EC ENGR 219: Project 1

End-to-End Pipeline to Classify News Articles

Yanfeng Guo (806073779)
Garvit Pugalia (504628127)
Hyosang Ahn (606073544)

Versions:

python	3.10
scikit-learn	1.2.0
pandas	1.5.2
numpy	1.23.5
matplotlib	3.6.3
nltk	3.8.1
umap-learn	0.5.3

Question 1

We use pandas.read_csv() method to read the dataset. It is easy to show the information of the original dataset through attributes such as shape and info.

Overview: How many rows (samples) and columns (features) are present in the dataset?

There are 3150 rows and 8 columns in the dataset.

Interpret Plots: Provide qualitative interpretations of the histograms.

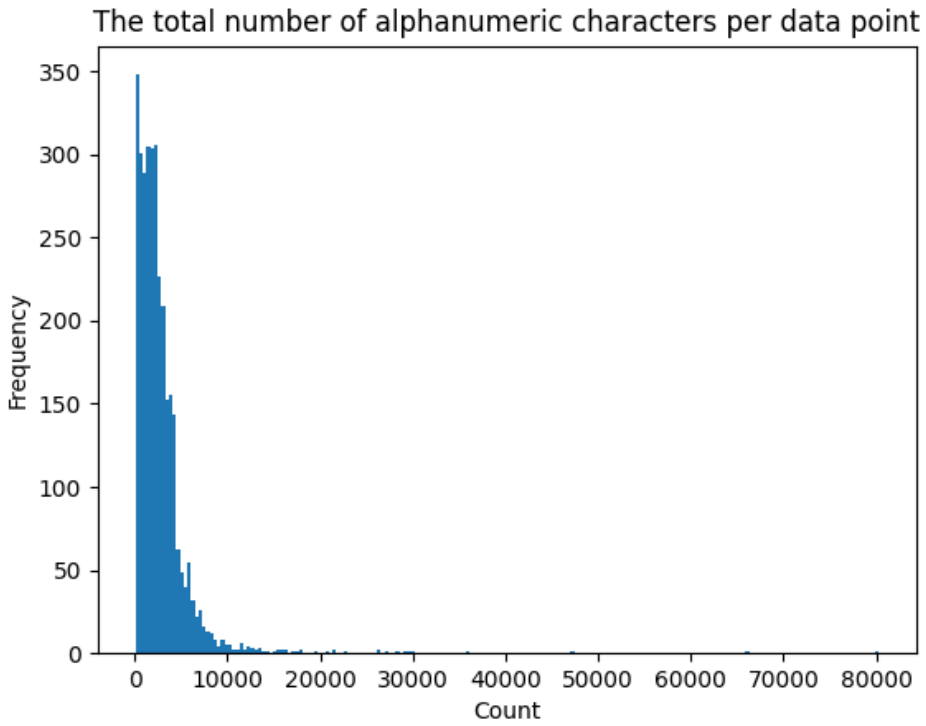


Fig. 1: The total number of alphanumeric characters per data point.

From Fig.1, we know that the data points have various alphanumeric characters. Most of them have less than 10000 characters. Some of them have up to 80000 characters. As far as the length of the article is concerned, the dataset has enough diversity.

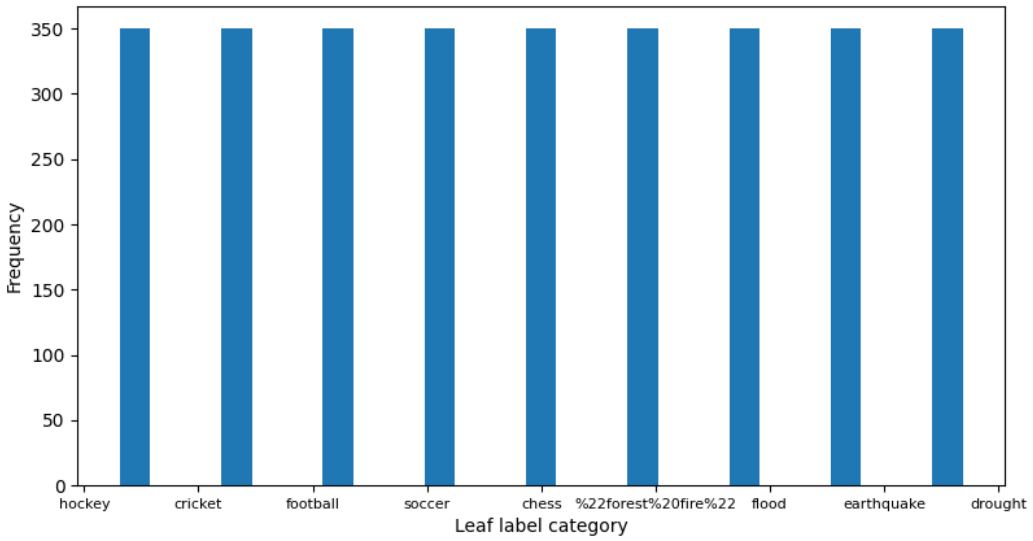


Fig. 2: The leaf label classes.

It is shown that the dataset has 9 leaf labels and the leaf labels are distributed equally.

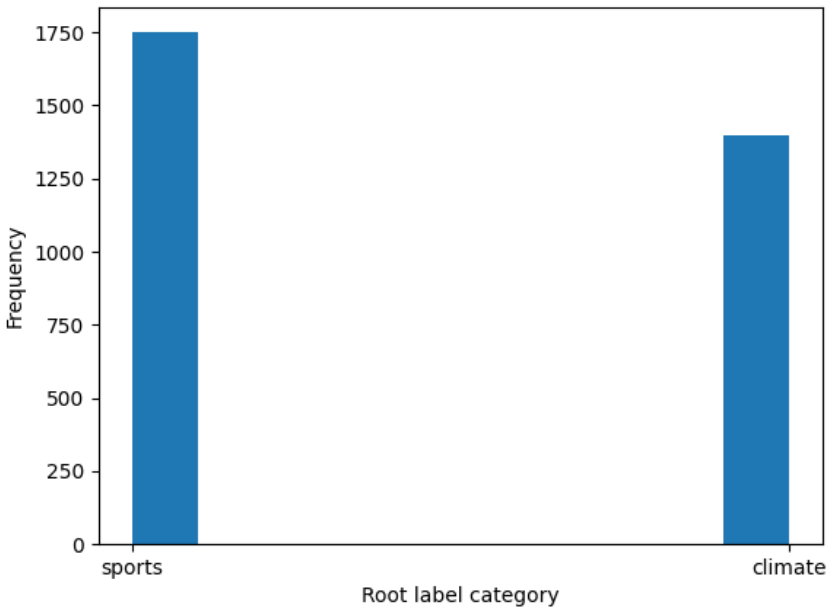


Fig. 3: The root label classes.

It is shown that the root labels are NOT distributed equally. Data with the label "sports" are more than those with the label "climate".

Question 2

Following the specs, we set the random seed as 42 and split the dataset into the training set and testing set. We print the shape of the training and testing set.

Report the number of training and testing samples.

There are 2520 training samples and 630 testing samples.

Question 3

In this section, we first clean the data through the provided function. To clear the items of numbers, the code is added to the provided function: texter = re.sub('[\d]', '', texter). Then we define the lemmatization function. Given a sentence, the function will first divide the sentence into separate words, and use nltk.pos_tag to get the property of a word. After that, we apply WordNetLemmatizer() as the lemmatizer. Then CountVectorizer() is used to count the lemmatized words. The stop words is 'english' and the minimum document frequency is 3. Finally, TfidfTransformer() is called to create a term frequency-inverse document frequency model.

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

Both lemmatization and stemming can reduce the number of words in the 'bag'. The advantage of lemmatization is that lemmatization can keep more information because words with the same root words may have different meanings. However, it is necessary to judge the property of words first, so the disadvantage is that lemmatization will cost more time. And it needs a dictionary to do the lemmatization, so it consumes more memory. By comparison, stemming is faster, but keeps less information, which may cause some errors. For instance, when given 'see', the algorithm simply returns 's', losing information.
The dictionary size is bigger after lemmatizing than stemming. It makes sense because more words become the same word after stemming.

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

Now the min df=3. If we make the parameter bigger, then only the words with higher frequency in the articles will not be removed, leading to fewer words in the bag of words, so the number of columns of the TF-IDF matrix will become smaller.

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

It is better to remove the stopwords before lemmatizing because it may save computational costs.
Punctuations should be removed after lemmatizing because they can help the lemmatizer decide the sentence structure and the accuracy of pos_tag() will increase. And we read the document of CountVectorizer() finding that punctuations will not be counted when the bag of words is constructed, so not removing punctuations should be fine.
Numbers should be removed before lemmatizing to save time and cost of lemmatization.

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 .

The shape of the TF-IDF-processed train matrix: (2520, 14598);
The shape of the TF-IDF-processed test matrix: (630, 14598).
The numbers of rows are identical to those of Question 2.

Question 4

In this section, we first train TruncatedSVDs with different numbers of components and call the attribute 'explained_variance_ratio_' of TruncatedSVD to get the explained variance ratio. Then we fix the number of components as 50 and call TruncatedSVD and NMF respectively to reduce the dimensionality of the data. Finally, we calculate the reconstruction residual MSE by comparing the original TF-IDF matrix and the reconstruction matrix.

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?

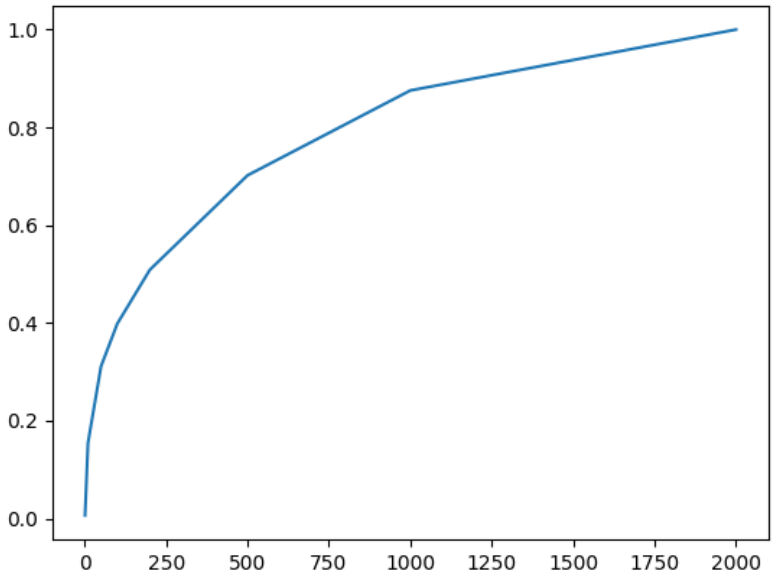


Fig. 4: The explained variance ratio across multiple different k. Different k on the x-axis and the corresponding explained variance ratio on the y-axis.

The plot looks like a classic concave function such as the logarithmic function. LSI chooses the top k largest singular values, representing the top k most important features. The explained variance ratio will increase as the number of components increases, which makes sense because more features are included. The ratio increases less quickly when k is big, suggesting that the components are chosen in the decreasing order of variance explanation ratio. The result shows that with the help of LSI, we can use a small number of components to reduce the dimensionality of data and save computational costs without losing the main information.

With k = 50 found in the previous sections, calculate the reconstruction residual MSE error when using LSI and NMF – they both should use the same k = 50. Which one is larger, and why?

LSI MSE: 1681.76
NMF MSE: 1705.49

The reconstruction residual MSE error of NMF is larger than that of LSI. The reason is that LSI uses SVD. Although SVD is slower, it can guarantee that the reconstruction residual MSE is the smallest. However, NMF is an algorithm that is related to the random seed. Given different random seeds, NMF may converge to different destinations with local minimums.

Question 5

In this section, we use the data after dimension reduction to train the SVM. It is important to get the ground truth label first. If the root label is ‘sports’, we set the true label as 1. If the root label is ‘climate’, we set the true label as 0. Then we call the SVM with different margins (soft, hard and harder margins) by setting different gamma values (0.0001, 1000, and 100000 respectively).

Train two linear SVMs: – Train one SVM with γ = 1000 (hard margin), another with γ = 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 γ = 100000?

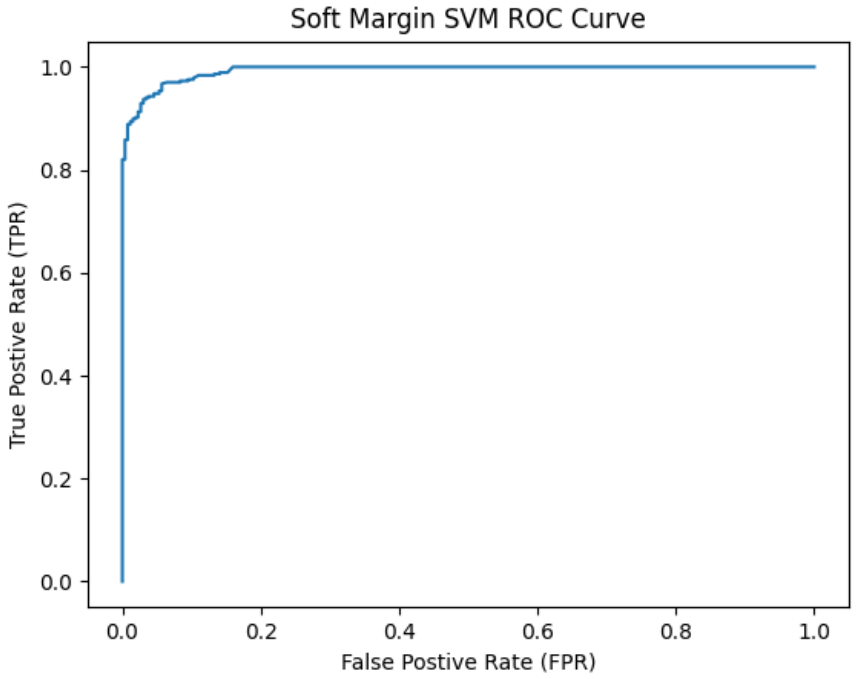


Fig. 5 The ROC Curve of the soft margin SVM (γ = 0.0001)

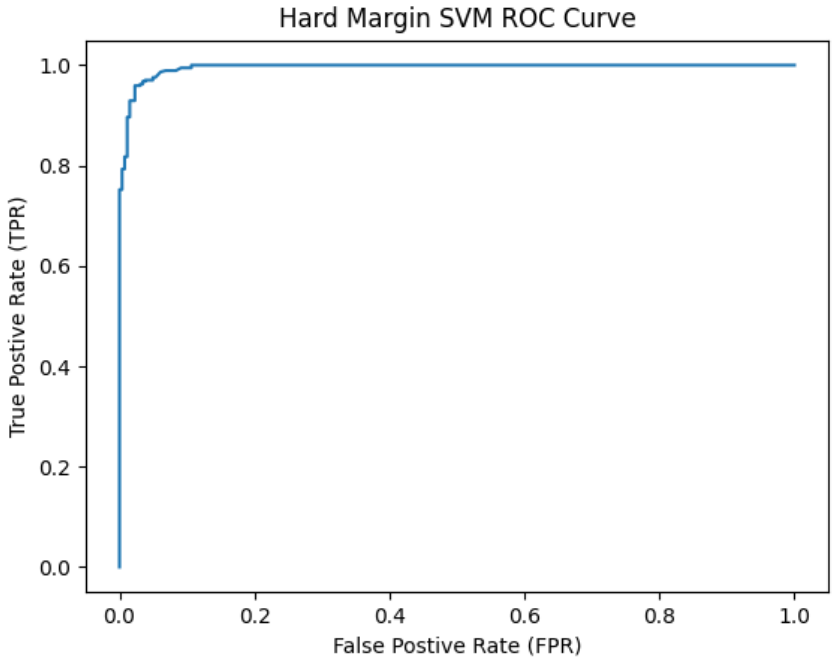


Fig. 6 The ROC Curve of the hard margin SVM (γ = 1000)

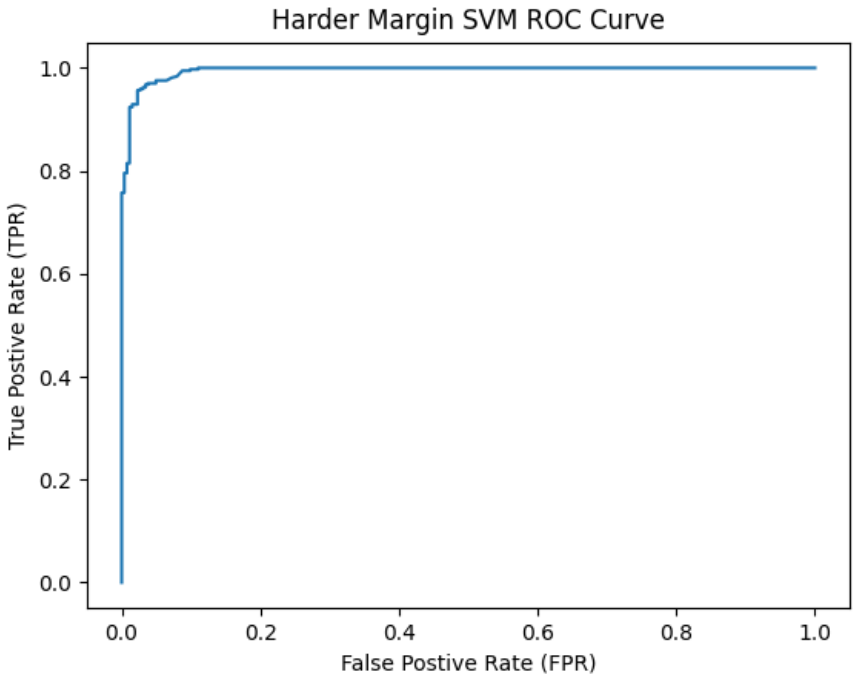


Fig. 7 The ROC Curve of the harder margin SVM (γ = 100000)

	Soft Margin SVM	Hard Margin SVM	Harder Margin SVM
Confusion Matrix	$\begin{bmatrix} 0 & 263 \\ 0 & 367 \end{bmatrix}$	$\begin{bmatrix} 257 & 6 \\ 16 & 351 \end{bmatrix}$	$\begin{bmatrix} 257 & 6 \\ 16 & 351 \end{bmatrix}$
Accuracy Score	0.5825	0.9651	0.9651

Recall Score	1.0	0.9564	0.9564
Precision Score	0.5825	0.9832	0.9832
F1 Score	0.7362	0.9696	0.9696

It is shown that the hard margin SVM performs better than the soft margin SVM. We can use the SVM with a hard margin to get a good result on the testing set. When γ=100000, it is a harder margin SVM. The performance of the harder margin SVM is the same as the hard margin SVM because the regularization strength is powerful enough.

What happens for the soft margin SVM? Why is the case? Analyze in terms of the confusion matrix. Does the ROC curve reflect the performance of the soft-margin SVM? Why?

The soft margin SVM has a poor performance. In fact, the confusion matrix shows that the predicted labels of all the testing samples are returned as 1. It is because the regularization strength is too weak, so the classifier allows more misclassifications. Weak regularization strength is very useful when the dataset can't be linearly divided perfectly, but in this case, the dataset can be linearly divided, so the result is very poor.

The ROC curve fails to reflect the performance of the soft margin SVM. To understand why the case is, we print the decision results of the soft margin SVM and observe what's going on. It turns out that the decision function result of each testing sample is almost 1. The threshold of each point of the ROC curve is between 0.987 and 2. When the threshold is high, such as 2, then most of the samples are predicted to be negative, then the true positive rate (TPR) and the false positive rate (FPR) are both close to 0. When the threshold goes down, TPR and FPR will grow rapidly because the decision function results of all samples are so concentrated around 1. The ROC curve suggests that to correctly evaluate the performance of a classifier, we need to use multiple metrics instead of relying on one metric.

Use cross-validation to choose γ (use average validation accuracy to compare): Using a 5-fold cross-validation, find the best value of the parameter γ in the range $\{10^k \mid -3 \leq k \leq 6, k \in Z\}$. Again, plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F-1 score of this best SVM.

K	-3	-2	-1	0	1	2	3	4	5	6
Score	0.5488	0.5492	0.9341	0.9409	0.9460	0.9520	0.9504	0.9524	0.9508	0.9512

We find the best value of the parameter is 10000. It is a hard-margin SVM.

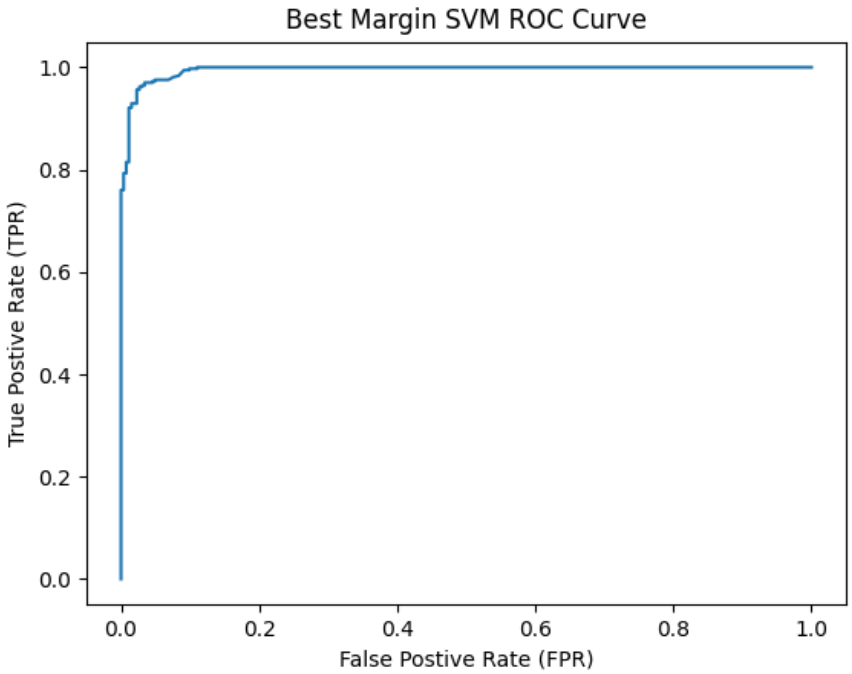


Fig. 8 The ROC Curve of the best margin SVM (γ = 10000)

Confusion matrix: $\begin{bmatrix} 257 & 6 \\ 16 & 351 \end{bmatrix}$; Accuracy Score: 0.9651; Recall Score: 0.9564; Precision Score: 0.9832; F1 Score: 0.9696.

Question 6

In this section, we call LogisticRegression() to train logistic classifiers without regularization, with L1 regularization, and with L2 regularization. 5-fold cross-validation is applied to find the optimal regularization strength. For the classifier without regularization, the solver is 'lbfgs'. For the classifiers with L1 regularization and with L2 regularization, the solver is 'liblinear'.

Train a logistic classifier without regularization; 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.

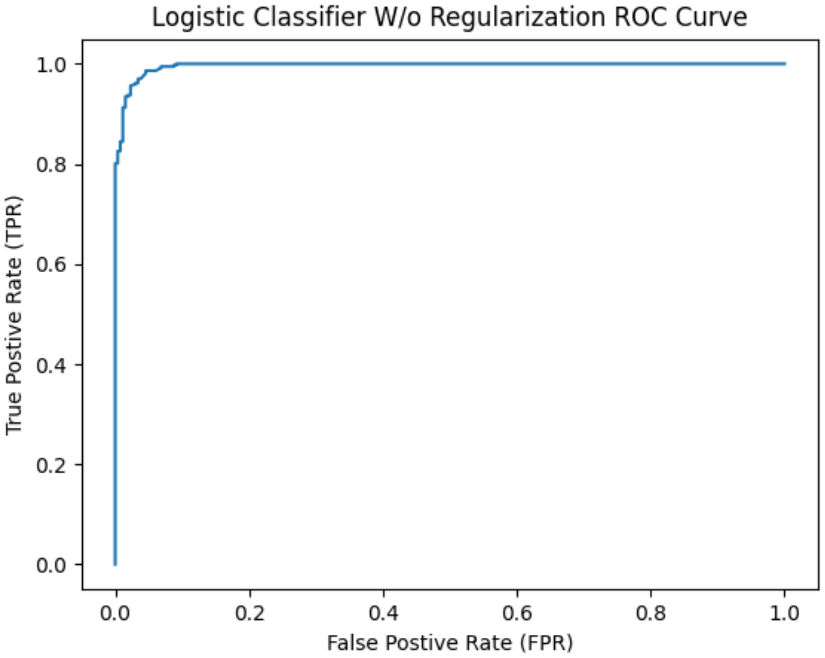


Fig. 9 The ROC Curve of the Logistic Classifier without regularization

Confusion matrix: $\begin{bmatrix} 257 & 6 \\ 17 & 350 \end{bmatrix}$; Accuracy Score: 0.9635; Recall Score: 0.9537; Precision Score: 0.9831; F1 Score: 0.9682.

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 {10k |−5 ≤ k ≤ 5, k ∈ Z} for logistic regression with L1 regularization and logistic regression with L2 regularization, respectively.

We use the 5-fold cross-validation to get the mean accuracy score of each model. The model with the highest average score is considered to be the best model.

K	-5	-4	-3	-2	-1	0	1	2	3	4	5
L1	0.451	0.451	0.451	0.451	0.920	0.941	0.947	0.953	0.952	0.952	0.952

	2	2	2	2	6	7	2	2	4	4	4
L2	0.5488	0.5488	0.5488	0.7310	0.9274	0.9437	0.9472	0.9480	0.9520	0.9523	0.9523

The best coefficient for L1 regularization: 100;
The best coefficient for L1 regularization: 10000.

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.

	w/o regularization	**L1 regularization**	**L2 regularization**
Accuracy Score	0.9635	0.9619	0.9635
Recall Score	0.9537	0.9510	0.9537
Precision Score	0.9831	0.9831	0.9831
F1 Score	0.9682	0.9668	0.9682

The performance of three models are very similar.

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?

During the experiment, we print the coefficients of the three models. It turns out that the coefficients of the models with L1 and L2 regularization are smaller than the coefficients of the model without regularization, especially the model with L1 regularization. Therefore, we conclude that the regularization parameter can restrict the magnitude of some coefficients through additional punishment. And it is observed that many coefficients in L1 regularization are close to 0, so L1 regularization may punish more on bigger coefficients than L2 regularization. We may be interested in L1 regularization if we want to have a simpler classifier with sparse coefficients. With the help of L1 regularization, many coefficients will be restricted to 0, and only the most important features will be kept. If we are concerned about how to avoid overfitting, we may be interested in L2 regularization.

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?

Linear SVM uses support vectors to find a separating hyperplane. It tries the maximize the margins between the points of different classes. Logistic regression is based on the

maximum likelihood estimation. It assumes that the data have a logistic distribution, and tries to maximize the likelihood that a data point is correctly classified.
Linear SVM is trained by solving a convex optimization problem, so it can always find the global minimum through reliable algorithms. However, a logistic classifier is often trained through coordinate descent (when L1 or L2 regularization is added). Sometimes the logistic classifier may find a local minimum point and stop there. In other words, a linear SVM model can always find the best hyperplane, but the logistic regression classifier can always find a good hyperplane, not necessarily the best. Moreover, Logistic regression assumes the distribution of data at the very beginning, but the data may not have such a distribution as assumed. However, the difference between the two models is NOT significant, if we compare the results of Question 5 and Question 6.

Question 7

In this section, we call GaussianNB() to train a naive Bayes classifier. The likelihood of the features is assumed to be gaussian.

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

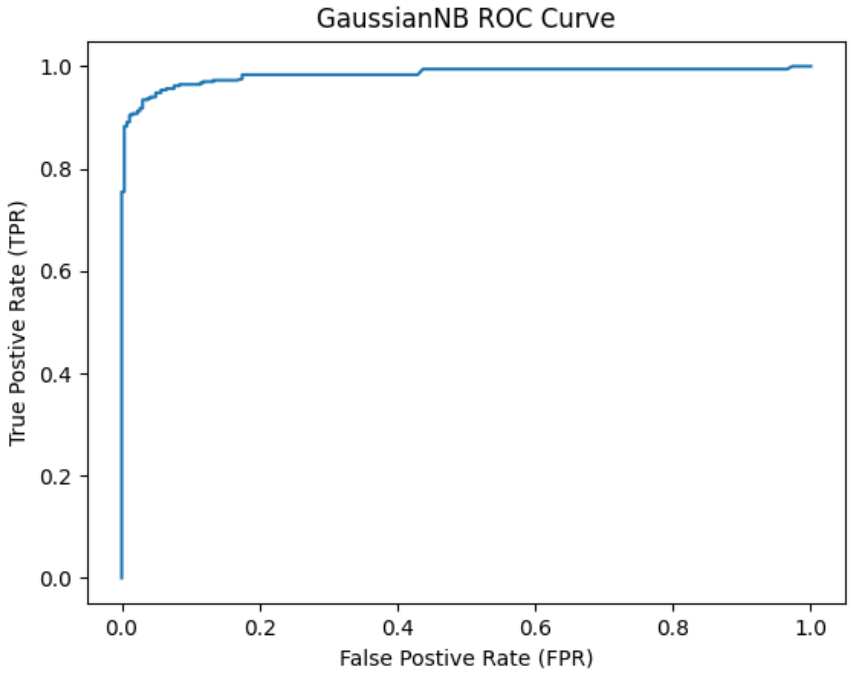


Fig. 10 The ROC Curve of the Gaussian Naive Bayes classifier

Confusion matrix: $\begin{bmatrix} 260 & 3 \\ 40 & 327 \end{bmatrix}$; Accuracy Score: 0.9317; Recall Score: 0.8910; Precision Score: 0.9909; F1 Score: 0.9383.

Question 8

In this section, we construct TWO pipelines to combine feature extraction, dimensionality reduction, and classification. The first pipeline takes the data after cleaning and lemmatizing as the input. The second pipeline takes the data after cleaning and stemming as the input. Both of the pipelines will take the following options when performing the grid search.

Module	Options
Feature Extraction	min_df = 3 OR 5 in CountVectorizer()
Dimensionality Reduction	LSI (k=5, 30, 80); NMF (k=5, 30, 80)
Classification	SVM (γ = 10000); GaussianNB(); Logistic Classifier with L1 regularization (coefficient for L1 regularization: 100); Logistic Classifier with L2 regularization (coefficient for L2 regularization: 10000)

It takes about 40 minutes to run the grid search codes.

What are the 5 best combinations? Report their performances on the testing set.

We save the grid search results of the two pipelines to 'grid_results_lemma.csv' and 'grid_results_stem.csv' respectively, and then combine the results and find the 5 best combinations manually. These models are as follows:

	Compression	min_df	Dim Reduction	Classification
1	stemming	3	LSI (k=80)	Logistic Classifier with L1 regularization
2	lemmatization	5	LSI (k=80)	Logistic Classifier with L1 regularization
3	stemming	3	NMF (k=80)	Logistic Classifier with L1 regularization
4	lemmatization	5	LSI (k=80)	Logistic Classifier with L2 regularization
5	lemmatization	3	LSI (k=80)	Logistic Classifier with L1 regularization

Their performance on the testing set are as follows:

	1	2	3	4	5
Confusion Matrix	$\begin{bmatrix} 256 & 7 \\ 15 & 352 \end{bmatrix}$	$\begin{bmatrix} 258 & 5 \\ 15 & 352 \end{bmatrix}$	$\begin{bmatrix} 260 & 3 \\ 15 & 352 \end{bmatrix}$	$\begin{bmatrix} 258 & 5 \\ 15 & 352 \end{bmatrix}$	$\begin{bmatrix} 258 & 5 \\ 14 & 353 \end{bmatrix}$

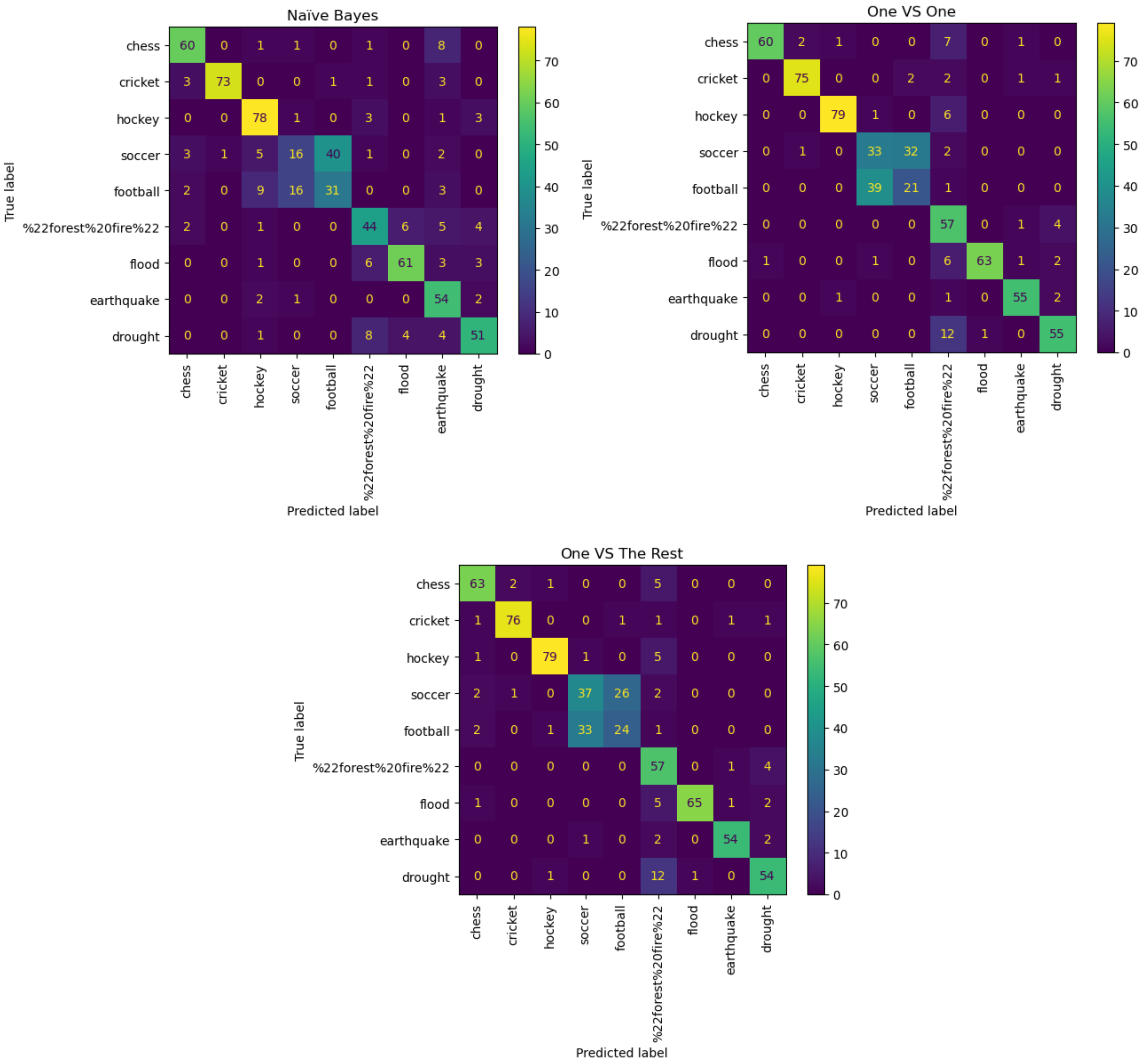
Accuracy Score	0.9651	0.9683	0.9714	0.9683	0.9698
Recall Score	0.9591	0.9591	0.9591	0.9591	0.9619
Precision Score	0.9805	0.9860	0.9915	0.9860	0.9860
F1 Score	0.9697	0.9724	0.9751	0.9724	0.9738

It is shown that all of the models can have a good performance on the testing set. The top model we've achieved in the cross-validation process is not the best one when it comes to the testing set, but the difference is NOT significant. It is normal that the best model in the validation is not the best model in the testing. The validation is aimed to choose a good model, not necessarily the best one.

Question 9

In this section, we experimented a similar approach of the Naïve Bayes classification as done above but using the Leaf Labels of the data as the multi-class classifier. We chose to use LSI and Gaussian NB method as we have observed that LSI produces smaller MSE error as experimented above.

Perform Naïve 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?



	Naïve Bayes	One VS One	One VS Rest
Accuracy	0.7429	0.7905	0.8079
Recall	0.7329	0.7812	0.7988
Precision	0.7260	0.7923	0.8036

F1	0.7219	0.7816	0.7973

In an attempt to improve class balancing in One VS the rest model, we utilized SKLearn's linear support vector classification method with the class weight set to 'balanced' to build the classifiers. This approach uses the y value–or the array of labels–to adjust the weight according to the frequency of each class. For the given data, the leaf labels are vaguely evenly distributed and the data are decently distinguished, so the label balancing method improved the recall score, precision score, and F1 score very marginally. Above results are produced with label imbalance issue addressed.

In the confusion matrix you should have an 9 × 9 matrix where 9 is the number of unique labels in the column leaf label. Please make sure that the order of these labels is as follows:

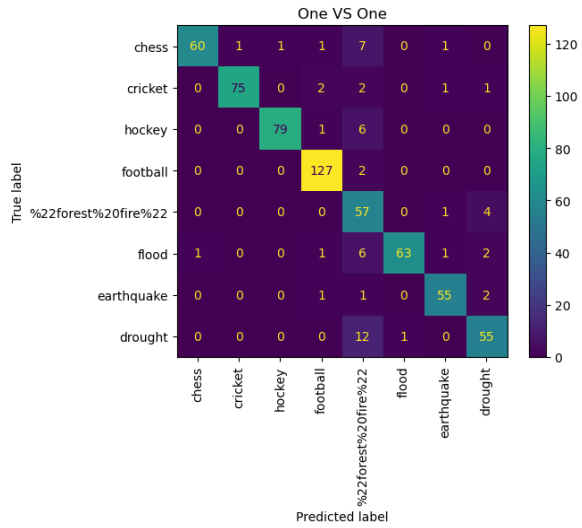
```
map_row_to_class = {0:"chess", 1:"cricket", 2:"hockey", 3:"soccer",
↪  4:"football", 5:"%22forest%20fire%22", 6:"flood", 7:"earthquake",
↪  8:"drought"}
```

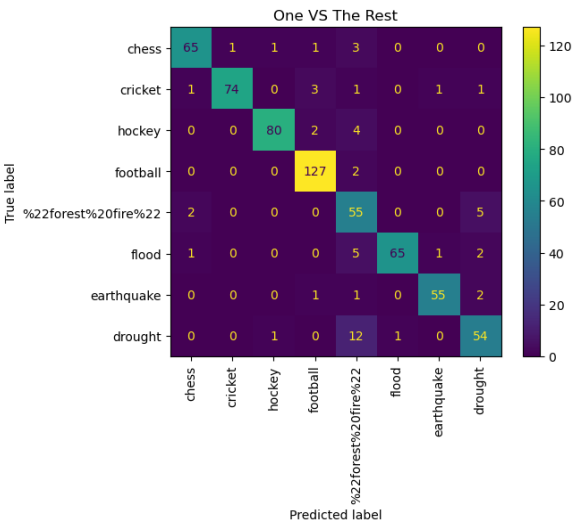
Do you observe any structure in the confusion matrix? Are there distinct visible blocks on the major diagonal? What does this mean?

The confusion matrix has its diagonal *roughly* concentrated. Its true positivity under labels 'soccer' and 'football' are not well concentrated, which causes the visible 2*2 square in the middle of the matrices. This may be due to the misconception between football and soccer, as those two words refer to the same or different sports depending on the region.

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

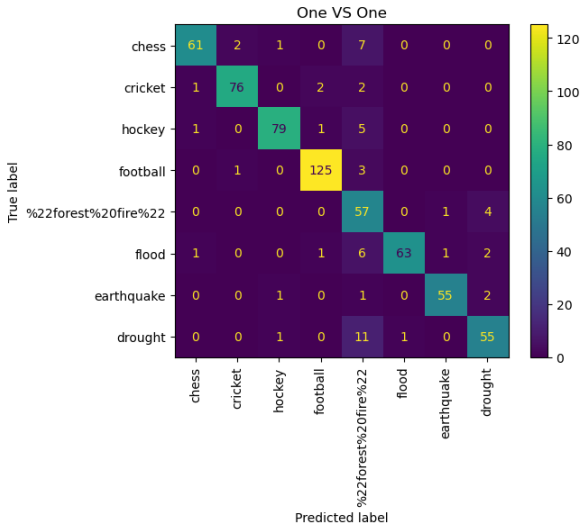
As a result of the above observation, merging 'soccer' and 'football' classes generates better accuracy. We merged them into a single label 'football'. The accuracy vastly increased after the merge for both One VS One and One VS The Rest models. The results are shown below:

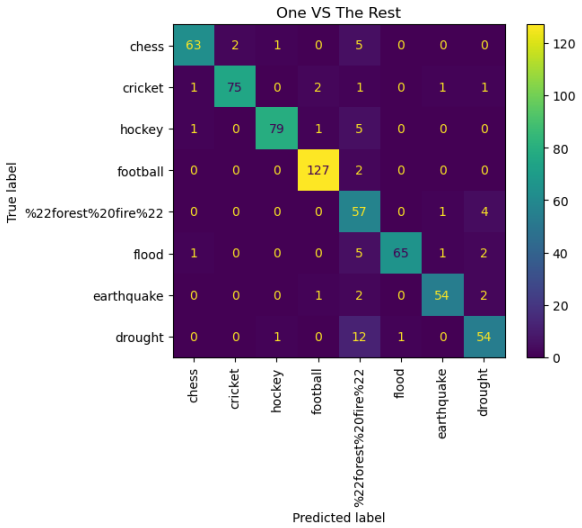




	One VS One	One VS Rest
Accuracy	0.9063	0.9127

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?.





	One VS One	One VS Rest
Accuracy	0.9063	0.9111

After merging the labels, the classifications are less evenly distributed but more well distinguished, so the class balancing does not make significant changes to the accuracy. Above results are produced after making the same changes as before the merge (LinearSVC with weighted average).

Question 10

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

The ratio can distinguish relevant words and discriminate among the relevant words. The paper shows an example of words “ice” and “steam” - two words that can represent the concept of thermodynamics. Four words that are either relevant to one, another, both, or neither are introduced. Using a token corpus, the probabilities and co-occurrence probabilities of word relevance are then collected. With probabilities themselves, the relevance was not clearly conveyed as there were no apparent patterns in the numbers at first glance. By using co-occurrence probabilities $\frac{P(k|ice)}{P(k|steam)}$, however, we can distinguish the relevance by the probability being much larger than 1 (relevant to ice), much smaller than 1 (relevant to steam), or close to 1 (relevant to both or neither).

(b) 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 for each word are pre-established as a result of training on a larger universal set of documents. Therefore, regardless of the context, the word ‘running’ would result in the same vector embedding from GLoVE. To expand on this idea further, the vector can be viewed as a weighted combination of all the occurrences of the word in the training corpus, so the co-occurence and contextual information (in this case, the two usages) will be stored inherently in the GLoVE embedding; however, when these embeddings are queried for the word ‘running’, it will be the same vector regardless of the usage.

(c) 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 these values.

Since ‘queen’ and ‘king’ have a similar relationship (and therefore similar patterns of co-occurrence in the training corpus) as ‘wife’ and ‘husband’ with the added condition of royalty, we can expect $||GLoVE[“queen”] - GLoVE[“king”]||_2$ and $||GLoVE[“wife”] - GLoVE[“husband”]||_2$ to have similar values, and thus $||GLoVE[“queen”] - GLoVE[“king”] - GLoVE[“wife”] + GLoVE[“husband”]||_2$ to be 0.

(d) Given a word, would you rather stem or lemmatize the word before mapping it to its GLoVE embedding?

While it is quick and simple, stemming is a naive way of finding base words as it simply truncates until a viable result. It can often return invalid words and single letters as the stem of the word, which works fine in terms of comparing different derivatives of the word but can lead to issues in querying the GLoVE embeddings (that only include valid English words).

In comparison, lemmatization is smarter and uses lexical knowledge to reduce words into their base form. When combined with other techniques such as part-of-speech tagging, this

process can analyze the morphology and context of words to output the most accurate, valid, English base word. Consequently, lemmatization would work better with GLoVE embeddings than stemming.

Question 11

In this section, we switched from TF-IDF vector representation of the text to using a larger set of GLoVE embeddings for each word. Before running a classification model, we needed to come up with a way to use the GLoVE embeddings to represent each document.

Describe a feature engineering process that uses GLoVE word embeddings to represent each document (*without exceeding the dimensions of the word embeddings itself*).

The cleaning and preprocessing of the textual data is done as before:

- Convert the text to lowercase and remove punctuations
- Remove common stopwords such as “i”, “the”, “an”, etc. that might skew the GLoVE representation of the document
- Lemmatize the words into their base form to cross-reference with the GLoVE embeddings

Instead of only relying on the full text, we appended the GLoVE representation of the ‘keywords’ column for each document to give it slightly more distinguishability. To do this, we put both the full text and the keywords for each document through the cleaning process as described above. Then we can:

- Get a dictionary mapping of the GLoVE embeddings to query the vector representation of each word
- For both “full_text” and “keywords”, we average across all the available word embeddings in the document. For each document, this gives us a vector of floats with the same length as the number of dimensions in the embeddings (e.g. 50, 100, 200, 300).
- Finally, with the normalized mean vectors, we average across the “full_text” and “keywords” columns of vectors. We take this average after normalizing each column in order to give slightly more weightage to keywords in the representation of the document.

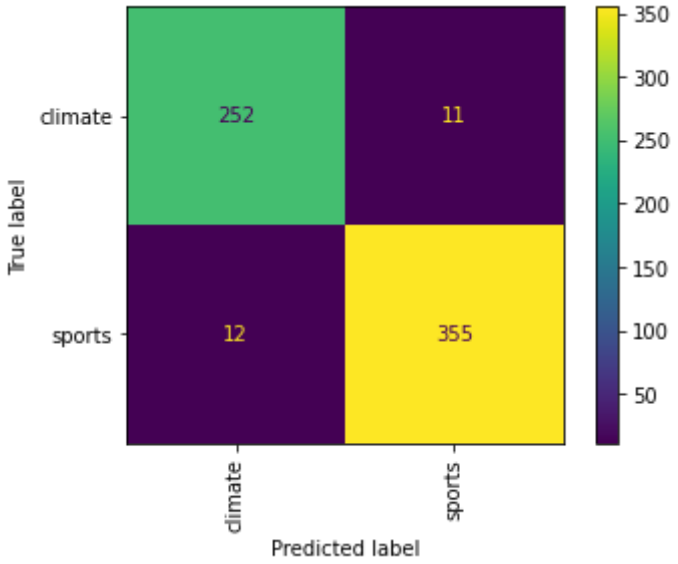
Once we have the documents represented as GLoVE vectors, we can perform classification with the binary labels and evaluate the model.

Select a classifier model, train and evaluate it with your GLoVE-based feature.

We tried Logistic Regression with different penalties and Linear SVM to fine-tune the model, however, the results were very similar. We decided to use the Linear SVM model for the classifier and performed 5-fold cross-validation to tune the parameter γ with the following results on the testing set:

γ	**Accuracy**	**Precision**	**Recall**	**F1 Score**
1	0.963492	0.962245	0.962739	0.962489

The confusion matrix of the model evaluated on the test set is:



We get a testing accuracy of 96.3%, which is similar to the best approaches using TF-IDF representation of the documents. The GLoVE representation doesn't provide a significant improvement - which could mean that there is room for improvement in the preprocessing and feature engineering pipeline, or could point to an issue in quality of data.

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.

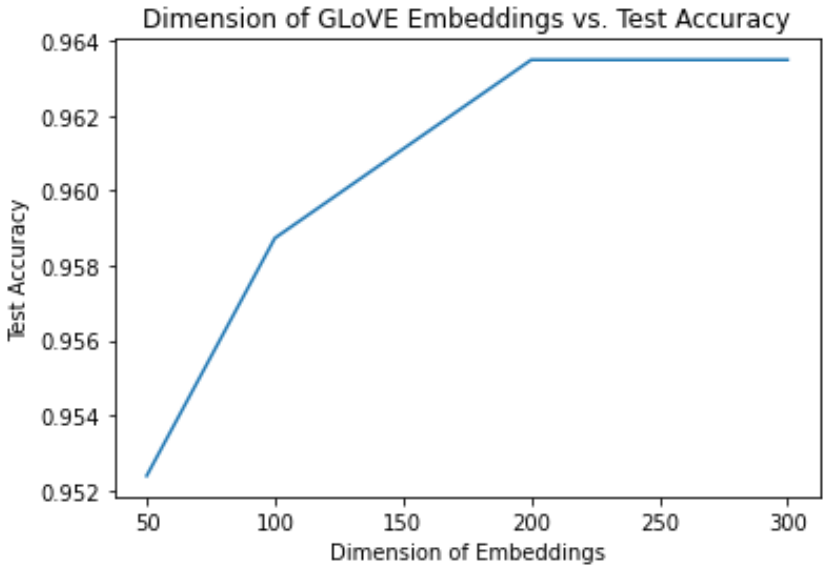


Fig. 11 Accuracy of classifiers with different GLoVE embedding dimensions

In the plot above, we observe that the accuracy of the classification model increases with the dimensions of the GLoVE embeddings with a plateau towards the end i.e. the increase in accuracy from 50-dimensions to 100-dimensions is more significant than the increase from 200-dimensions to 300-dimensions. At the same time, it is important to notice that the accuracy is pretty high for every embedding so there is a tradeoff between accuracy and time/space complexity as we increase dimensions.

Is this trend expected? Why or why not?

We expect the accuracy of the model to increase with the dimensions of the pre-trained GLoVE embeddings. Intuitively, the semantic and contextual information of a word captured by 300 floats (in a 300-dimension embedding) will be more detailed than if it was captured in a 50-dimension embedding. We can almost think of each float in the embedding as an extracted feature of the word; therefore, a higher dimension embedding will lead to more features extracted for the words (and consequently the document), better distinguishability from other documents, and ultimately a higher classification accuracy. The plateauing of the curve implies that any higher dimensions (more features) will not improve the accuracy as significantly as it reaches a saturation point.

Question 13

Visualize the set of normalized GLoVE-based embeddings of the documents with their binary labels in a 2D plane using the UMAP library.

For the training set, represented by GLoVE embeddings, we draw a labeled UMAP:

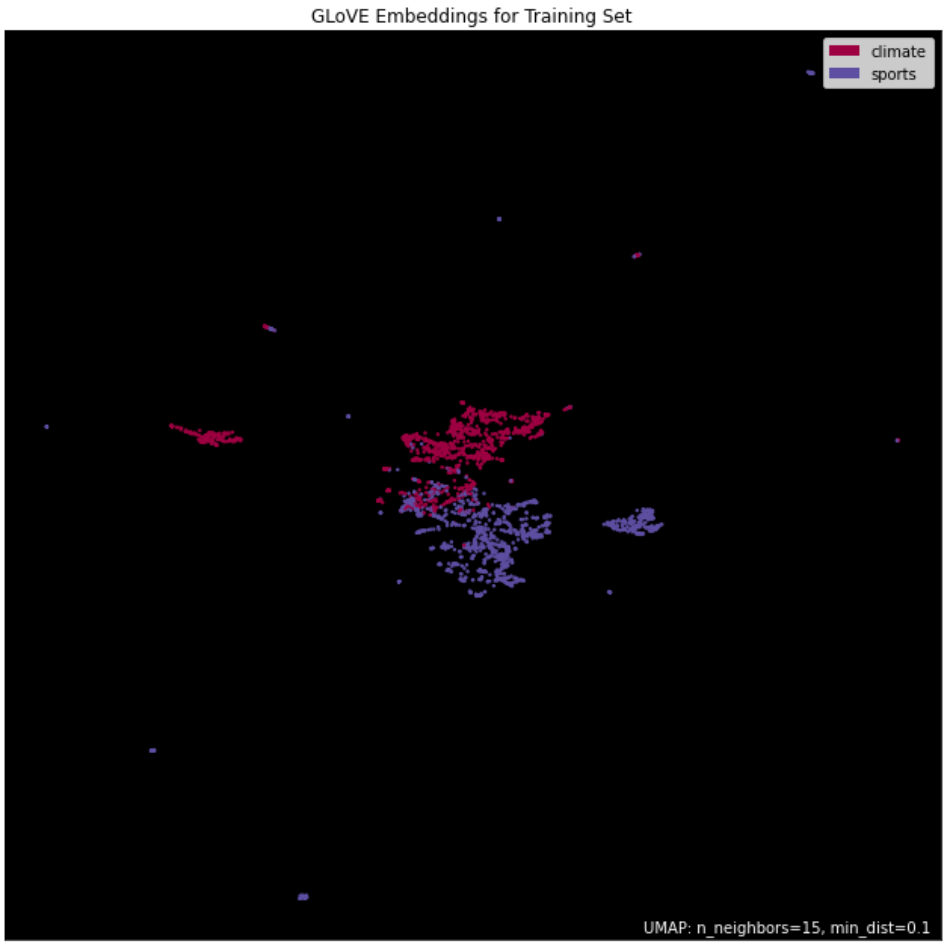


Fig. 12 Distribution of GLoVE representation of training set (n_dim = 300)

Similarly, for the testing set, we get:

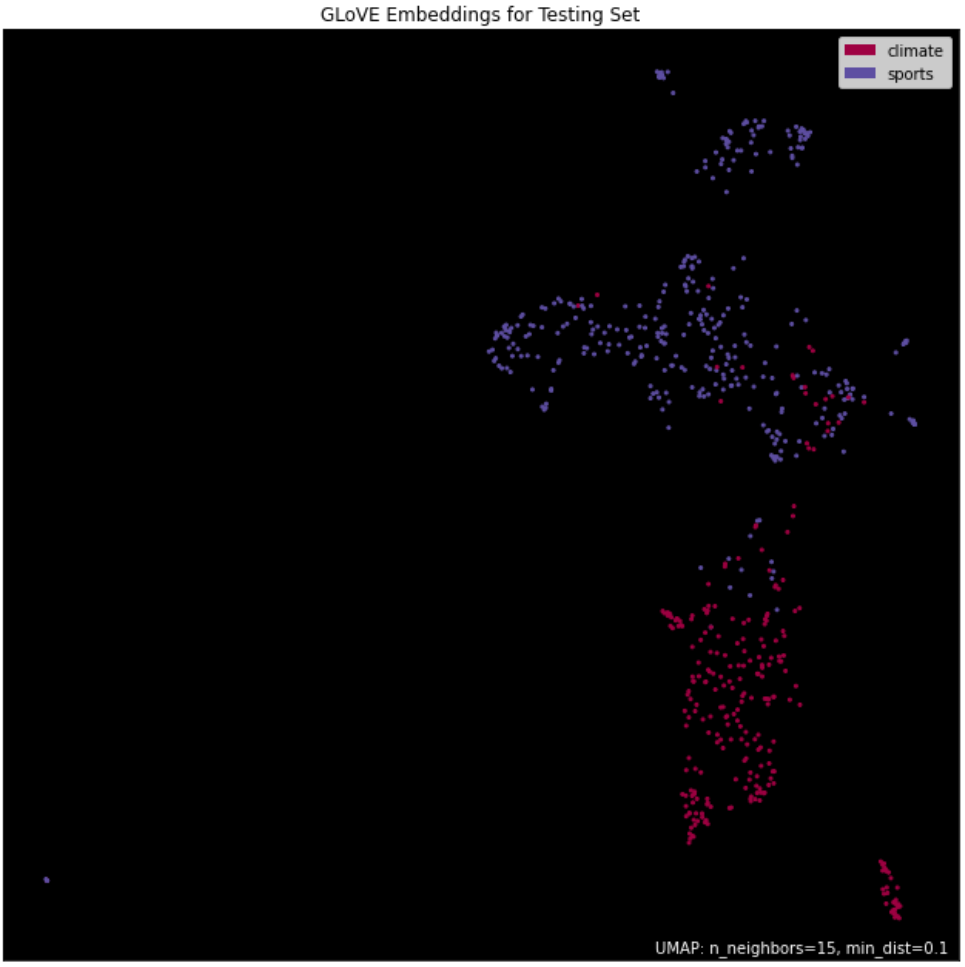


Fig. 13 Distribution of GLoVE representation of testing set (n_dim = 300)

Similarly generate a set of normalized random vectors of the same dimension as GLoVE and visualize those in a 2D plane using UMAP.

Using the numpy random function and normalizing the vectors, we get the following visualization for 300-dimension vectors labeled according to the document labels for comparison:

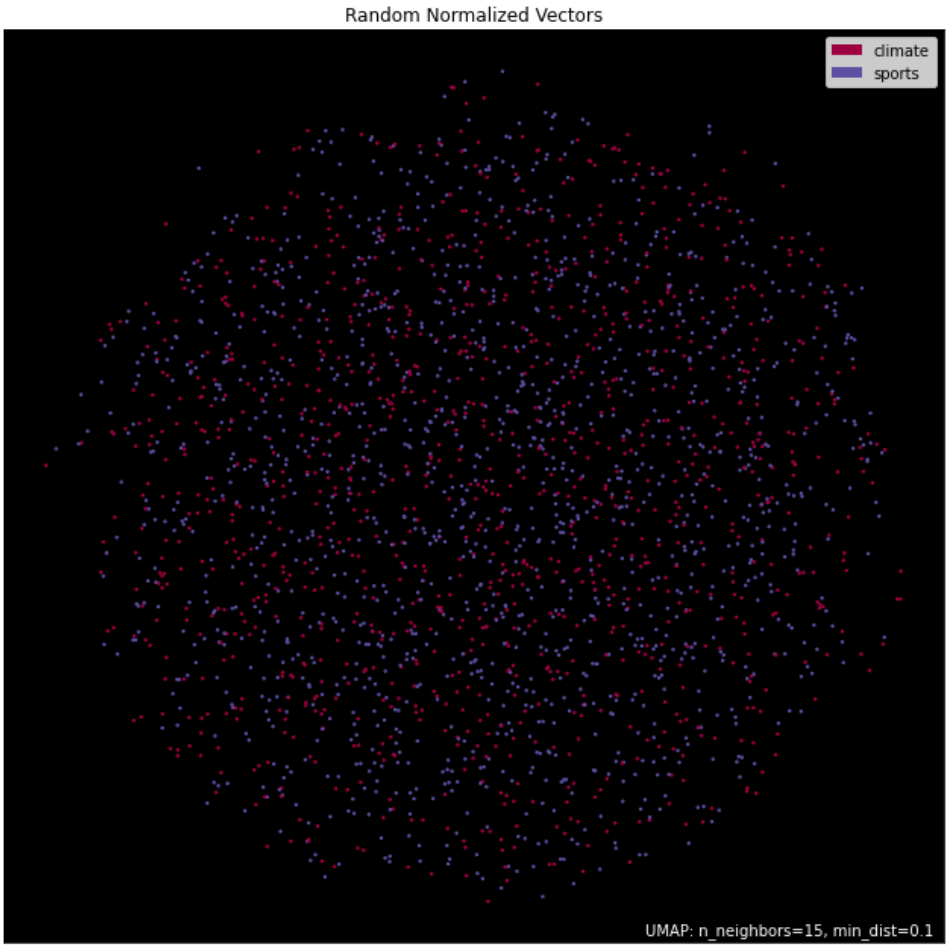


Fig. 14 Distribution of GLoVE representation of random normalized vectors (n_dim = 300)

Compare and contrast the two visualizations. Are there clusters formed in either or both of the plots?

In the UMAP visualization of the training and testing set, we can observe some clustering of the two different classes: sports, and climate. While there are some outliers, the UMAP for the training set displays significant clustering based on the two classes implying a distinguishability between the two types of documents that can be used for classification. There is some overlap between these clusters which can be an issue for the classifier - we can possibly fine-tune the GLoVE feature engineering process to further distance the clusters. In the testing set, there is very clear clustering of the two classes; however there are some data points in the incorrect clusters which correspond to the false classifications.

In the visualization of the normalized random vectors, created using the random function in the numpy library, there is no clustering and hence no distinguishability between the data points of the two classes. The data is evenly spread out regardless of label.

These UMAPs demonstrate the ability of the GLoVE embeddings to provide distinguishability between different document type representations that would not be seen in a randomized representation.