Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

Project 1 Report

For this project, we are tasked with creating an ML pipeline responsible for classifying news articles. The first step of the pipeline is to extract features from the raw text data using TF-IDF representations. Then, we perform dimensionality reduction using Principal Component Analysis (PCA) and non-Negative Matrix Factorization (NMF). After that, we fit common classification models to the modified dataset. These models are the Support Vector Classifier (SVC), logistic classifier with L1 loss, logistic classifier with L2 loss and Gaussian naïve bayes model. We perform hyperparameter optimization by constructing a pipeline and utilizing grid search. Finally, we perform multi-class classification on the dataset and experiment with using GloVE embeddings to generate features instead of relying on TF-IDF. The entirety of the project is coded in Python using mainly the scikit-learn and pandas libraries.

Getting Familiar with the Dataset

The first step of the project was to load the data and perform our preliminary exploration. To do so, we utilized the read_csv function of the pandas library.

Question 1

After loading the dataset, it was observed that the set consisted of **2072** rows or samples and **9** columns or features.

Then, we examined how many alpha-numeric characters are present for each data sample in the dataset. To do so, we stored the amount of alpha-numeric characters in each sample and plotted the histogram of the results. We plotted the histogram by dividing the amounts to 50 bins.

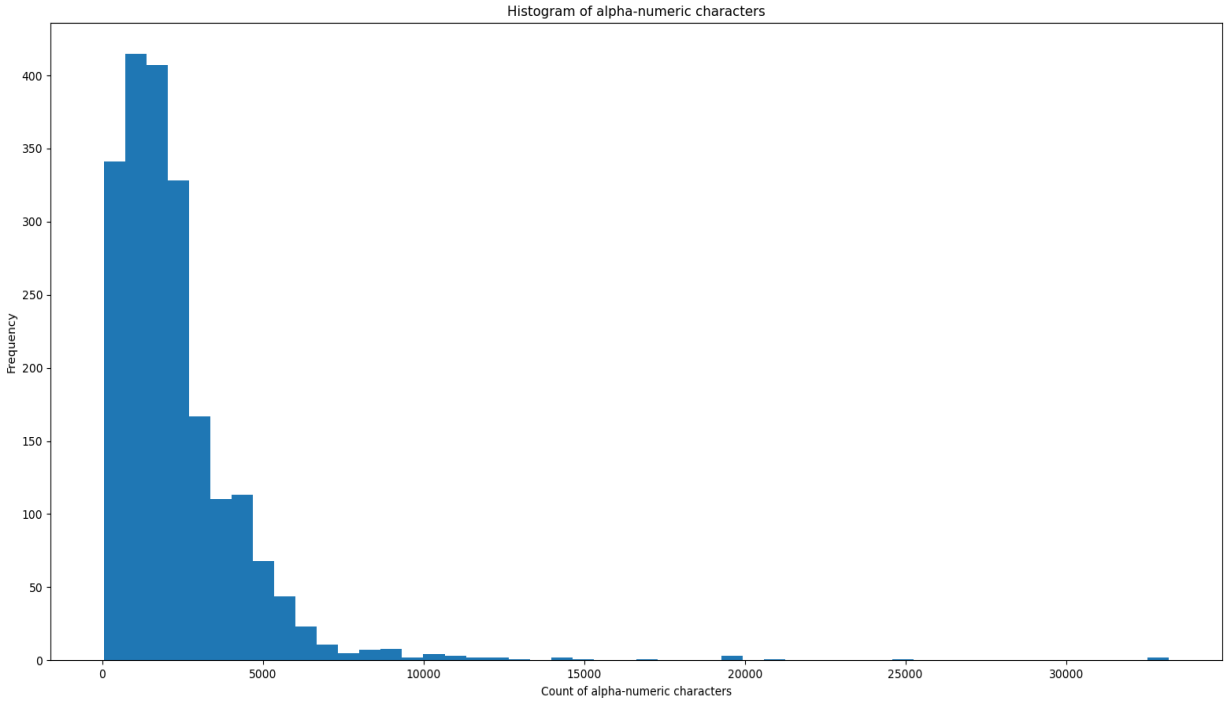

Figure 1: Histogram of alpha-numeric characters

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

As can be seen, the majority of data samples have an alpha-numeric count between 43 and 2700. This gives us an idea about the approximate length of each data sample. However, it can still be observed that there are some lengthy data samples with over 20000 alpha-numeric characters.

Then, we plotted the histogram of the labels in the dataset. The results are as follows:

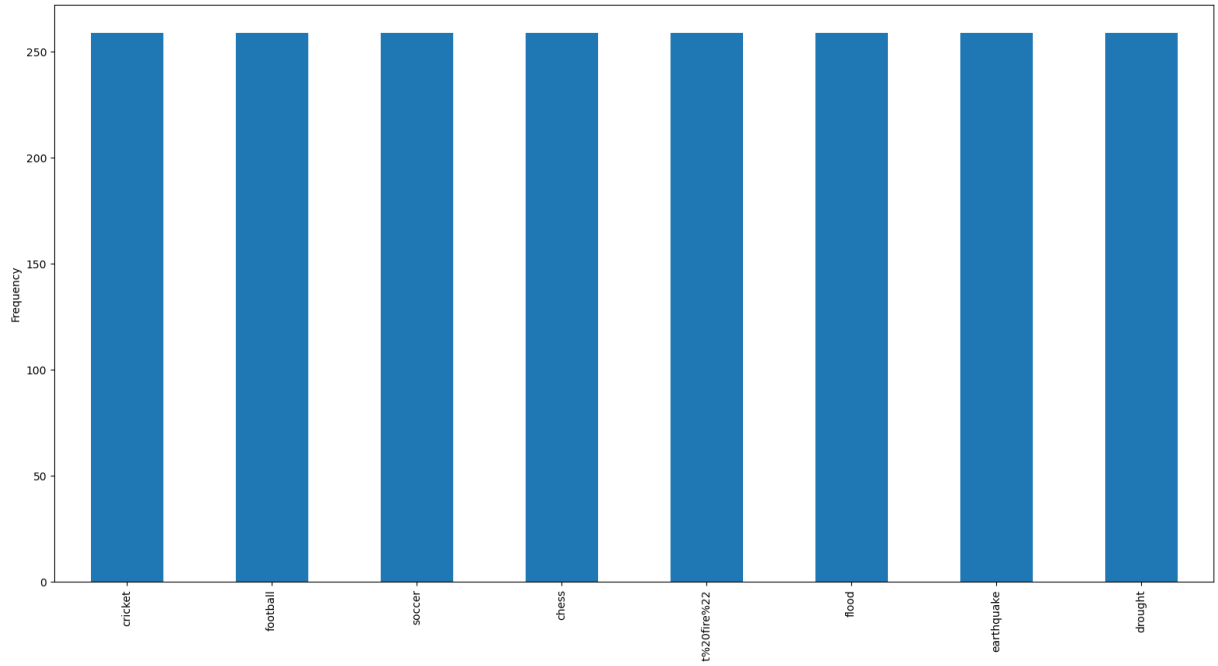

Figure 2: Histogram of leaf label elements

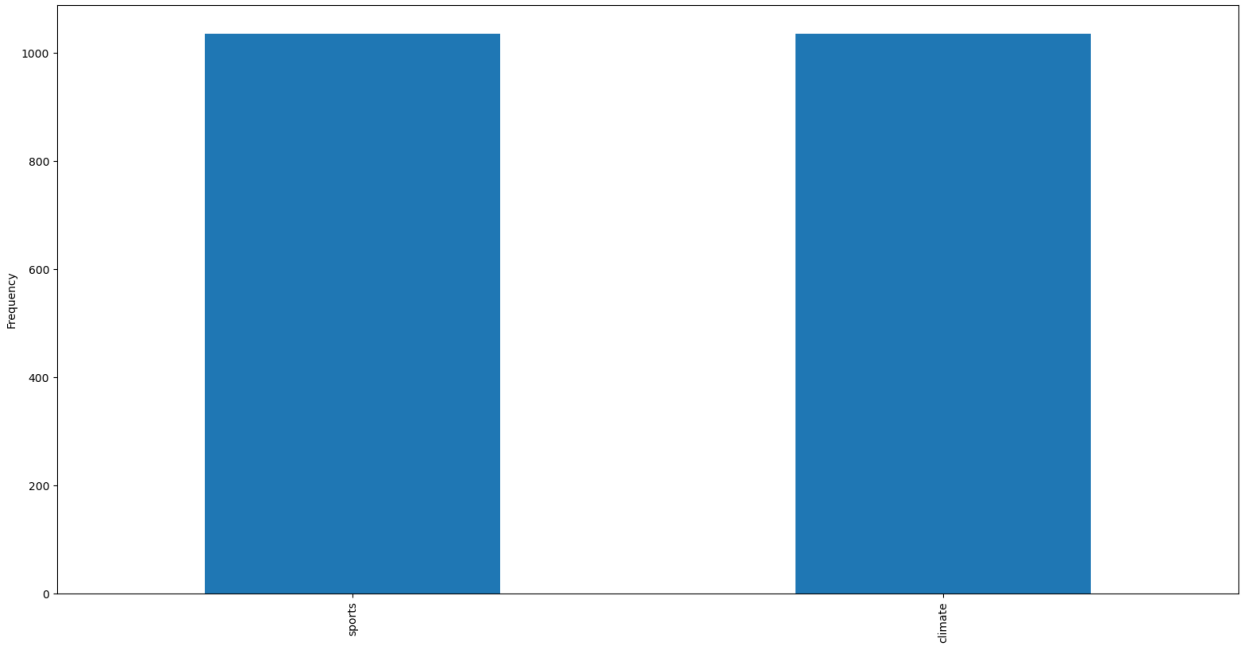

Figure 3: Histogram of root_label elements

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

As can be seen, for both leaf and root labels, there seems to be an equal number of datapoints for unique label values. This is beneficial since this means that our dataset does not suffer from the class imbalance problem.

Now that we have a better idea about the dataset, we can move on to the ML pipeline.

Binary Classification

In order to build our models, we need a training set and a test set. The training set is used to develop the model while the test set is used to get a better idea about the performance of the model. After setting the random seeds, the dataset was separated using the train_test_split function of the sklearn package.

Question 2

After a split with 80% of the data for training and 20% for testing, we had **1657** samples for training and **415** samples for testing.

Feature extraction

In order to fit a model to the data, we need to convert the text in our dataset to features that can be used in an ML model. In NLP tasks, one of the most common methods of doing so is to utilize the Bag-of-Words method, where a document is represented as a histogram of word frequencies. Although this method is fairly simple, a better approach would be to scale the count numbers by the number of documents or data samples the word appears in. This encourages the model to look at words that differentiate one sample from the next. TF-IDF method is based on this approach. The formula for TF-IDF is:

$$TF_{IDF(d,t)} = TF(t,d)xIDF(t)$$

where TF is the word frequency in document d of word t and IDF is:

$$IDF(t) = \log\left(\frac{n}{DF(t)}\right) + 1$$

Where n is the total number of documents and DF(t) is the number of documents that contain word t.

To perform feature extraction in our project, we first used the provided clean method to remove any HTML artefacts that are present in the data. Then, we defined our own lemmatization function. Here, we first separate a data sample to its sentences using the tokenize class of NLTK. Then, for each sentence, we separate the sentence to its words using the CountVectorizer class. After that, we check each word to make sure that it does not contain any digits and that it is not a part of the stop words of the CountVectorizer. The remaining words are lemmatized using NLTK's Wordnet lemmatizer by first getting the positional tag of the words using the pos_tag method and then feeding the positional tag and the word to the lemmatizer. This allows the lemmatizer to get the basic form of words based on the context they are used in.

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

We use our custom lemmatization function as the analyzer in the CountVectorizer class. We specify in the count vectorizer that we only intend to keep a word in the vocabulary after lemmatization if it is present in at least 3 documents and if it is not part of the stop words for the English language.

Question 3

Lemmatization is not the only way of reducing a word to its basic form. Another method of doing so is called stemming. While the lemmatizer uses positional knowledge and context to reduce a word to its basic form, stemming simply removes the last characters of a word to hopefully get the basic form of a word.

The main advantage of lemmatization is that it provides a more accurate way of transforming words. Consider the example of “caring”. After lemmatization, the base form of this word will be “care”, which would be correct. However, after stemming, the resulting word is “car” which is not related to “caring”.

However, stemming is easier to implement since it does not care about a word’s part-of-speech and only uses basic heuristics. In terms of performance, stemming is faster. However, thanks to modern computers the difference in time is negligible, so lemmatization is more popularly used.

The size of the dictionary is less for the stemmed result when compared with the result obtained after lemmatization. After lemmatization, the dictionary consists of 9954 words whereas the dictionary after stemming consists of 8707 words. This could be attributed to the fact that stemming maps multiple unrelated words to the same base form.

As min_df increases, infrequent words are eliminated from the vocabulary, which means that the amount of columns in the TF-IDF matrix decreases. On the contrary, as min_df decreases, there are more words in the vocabulary so the amount of columns in TF-IDF increases.

The stop words should be removed before lemmatization, since some stop words may be altered after lemmatization. As the altered stop words will not be present in the stop word list, they will not be removed from the generated dictionary, which is not desirable. Numbers and punctuation can also be removed before lemmatization, since these components can not be lemmatized and therefore, will not be modified after lemmatization. Thus, it is unnecessary to feed these components to the lemmatizer.

After using Count_vectorizer with mindf=3, the resulting TF-IDF matrix for the train set had 1657 rows and 9954 columns while the TF-IDF matrix for the test set had 415 rows and 9954 columns.

Dimensionality Reduction

Our current TF-IDF set has 9954 columns and only 1657 rows. When the number of features is greater than the amount of data points, our model can suffer from something called the Curse of Dimensionality. To get good performing models, we need to find a way to reduce the number of features. In the homework, we are given two methods to do so.

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

The first method is called Latent Semantic Indexing. Here, we calculate the Singular Value Decomposition of the data matrix. Then, we reconstruct the matrix using the top k columns of the U and V matrices. In our code, this is implemented using scikit-learn's TruncatedSVD function.

The second method is called non-negative matrix factorization. Here, the goal is to generate W and H matrices such that the Frobenius norm between the original matrix and the dot product of W and H is as small as possible. Then, W becomes the dimensionality reduced matrix. This part of the homework is implemented using sklearn's NMF function.

Question 4

In order to see how the explained variance ratio changes as the number of components used in LSI increases, we can fit different truncated SVDs with differing amounts of components and see how the explained variance ratio changes. The resulting graph is as follows:

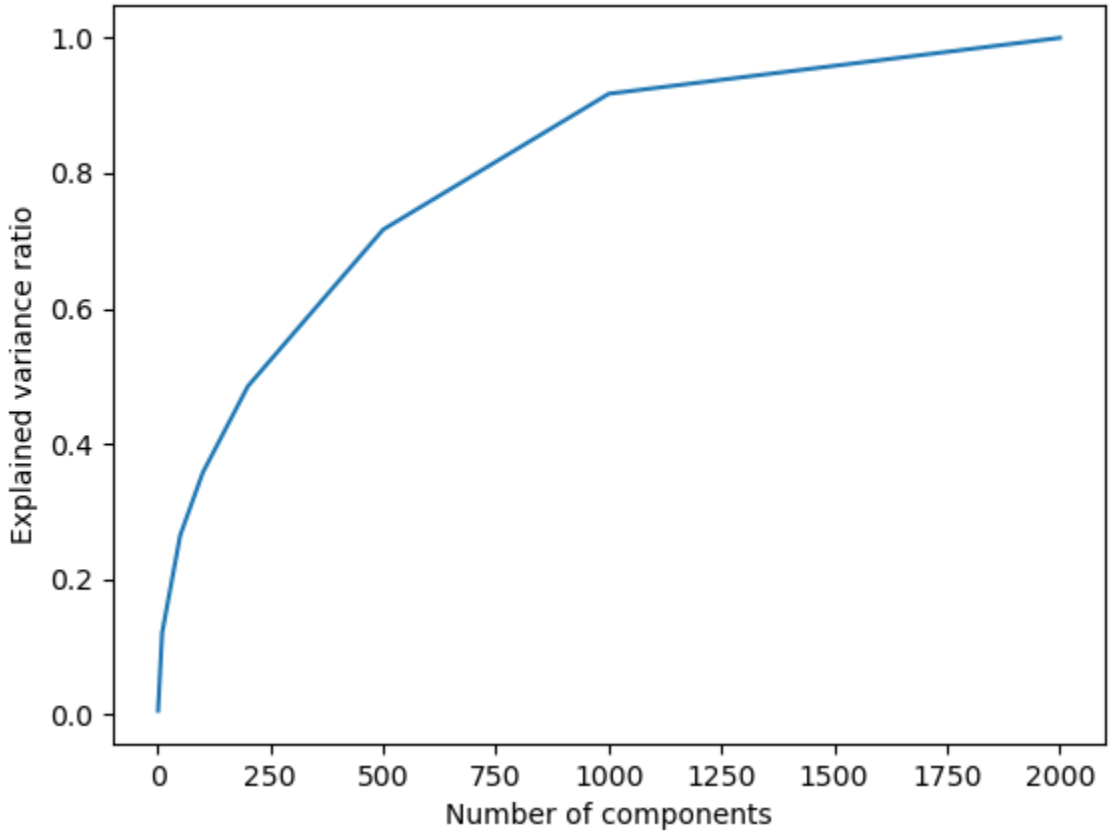

Figure 4: The change in explained variance vs number of components

The concavity of the plot suggests that the amount of explained variance contributed by each component decreases as the number of components increase. At first, using more components has a dramatic effect on the explained variance. However, later on adding more components does not change the explained variance that much.

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

Let us now compare the reconstruction results for both LSI and NMF. To get the reconstruction from LSI, we first performed singular value decomposition on the TF-IDF train set with 50 components. Then we calculated the dot product of U, SIGMA and V to get the reconstructed matrix. The Frobenius norm between this reconstruction and the original matrix was found to be **34.314**.

For the NMF, we first obtained W and H matrices using the NMF function with 50 components. After that, we multiplied the W and H matrices to get the reconstruction matrix. Then, we calculated the Frobenius norm between the original and the reconstructed matrices as **34.62**.

From the results, it appears that the Frobenius norm of NMF is greater than the Frobenius norm of LSI. This implies that more information is retained after the LSI process, which allows a more accurate reconstruction of the original data matrix.

We decided to use LSI with **50** components to reduce the dimensionality of our data. After this step, we were free to move on to fitting different classification algorithms to our created dataset.

Classification Algorithms

Now that we have our dataset that we can work on, it is time to fit different classification algorithms. But before doing that, let us define some metrics that can be used to evaluate the performance of our classifiers. For a binary classification task, the confusion matrix is as follows:

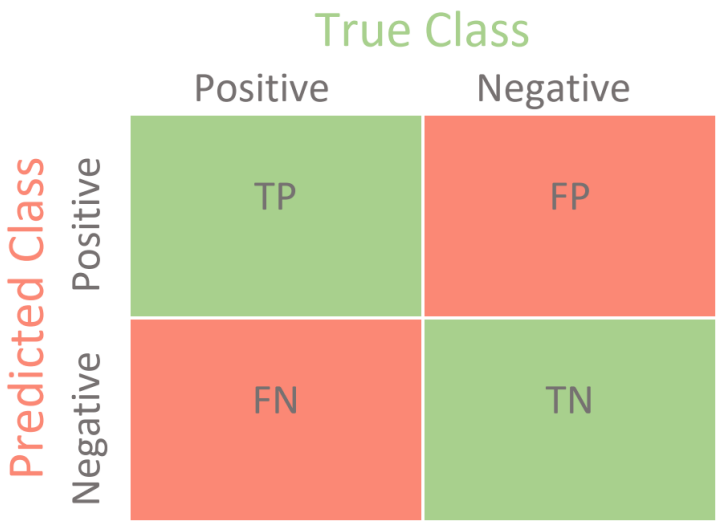

Figure 5: A confusion matrix for a binary classification task

From this confusion matrix, we can define several metrics for evaluation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN}$$

$$False\ Positive\ Rate = \frac{FP}{TN + FP}$$

By plotting the false positive rate and the true positive rate, we get what we call a Receiver Operating Characteristic (ROC) curve. This ROC curve gives us information about our model's performance based on varying decision thresholds.

SVM

The first model that we fit is the Support Vector Machine or SVM. The goal of the SVM is to find an optimal separating hyperplane between two classes. As such, SVM aims to optimize the following cost function:

$$\min_{w,b} \frac{1}{2}|w|^2 + \gamma \sum_{i=1}^{n} \varepsilon_i \text{ s.t. } y_i(w^T x + b) \geq 1 - \varepsilon_i$$

As the variable γ increases, more importance s placed on correctly classifying each data point. As it decreases, the model becomes free to make some mistakes. Therefore, changing γ equates to a form of regularization.

Question 5

In this part of the homework, we are asked to first train a total of three SVM models with different gamma values. As the gamma value increases, the model is penalized heavily for each misclassification, resulting in a stricter model. The gamma vales used in our experiments were: 0.0001,1000,100000. The results of our experiments can be summarized below. Here, it is important to note that the positive class was defined as the climate class.

Table 1: Performance of different SVMs

Gamma	Accuracy	Precision	Recall	F-1
1000	97.5%	98.5%	96.6%	97.5%
0.0001	49.6%	49.6%	100%	66.3%
100000	97.8%	98.5%	97.0%	97.7%

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

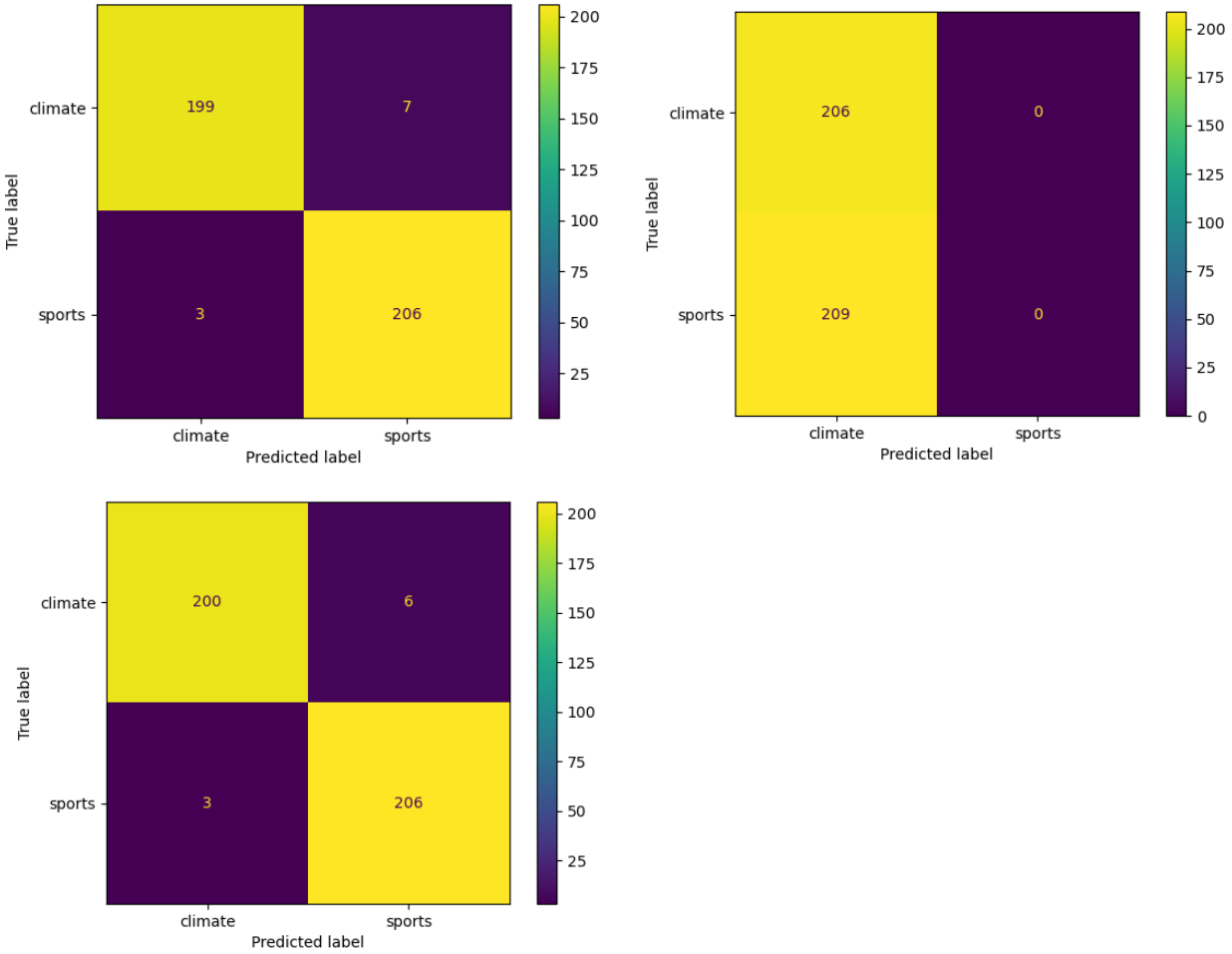

Figure 6: The confusion matrices of the trained SVMs. Top left for gamma = 1000, top right for gamma = 0.0001 and bottom left for gamma = 100000

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

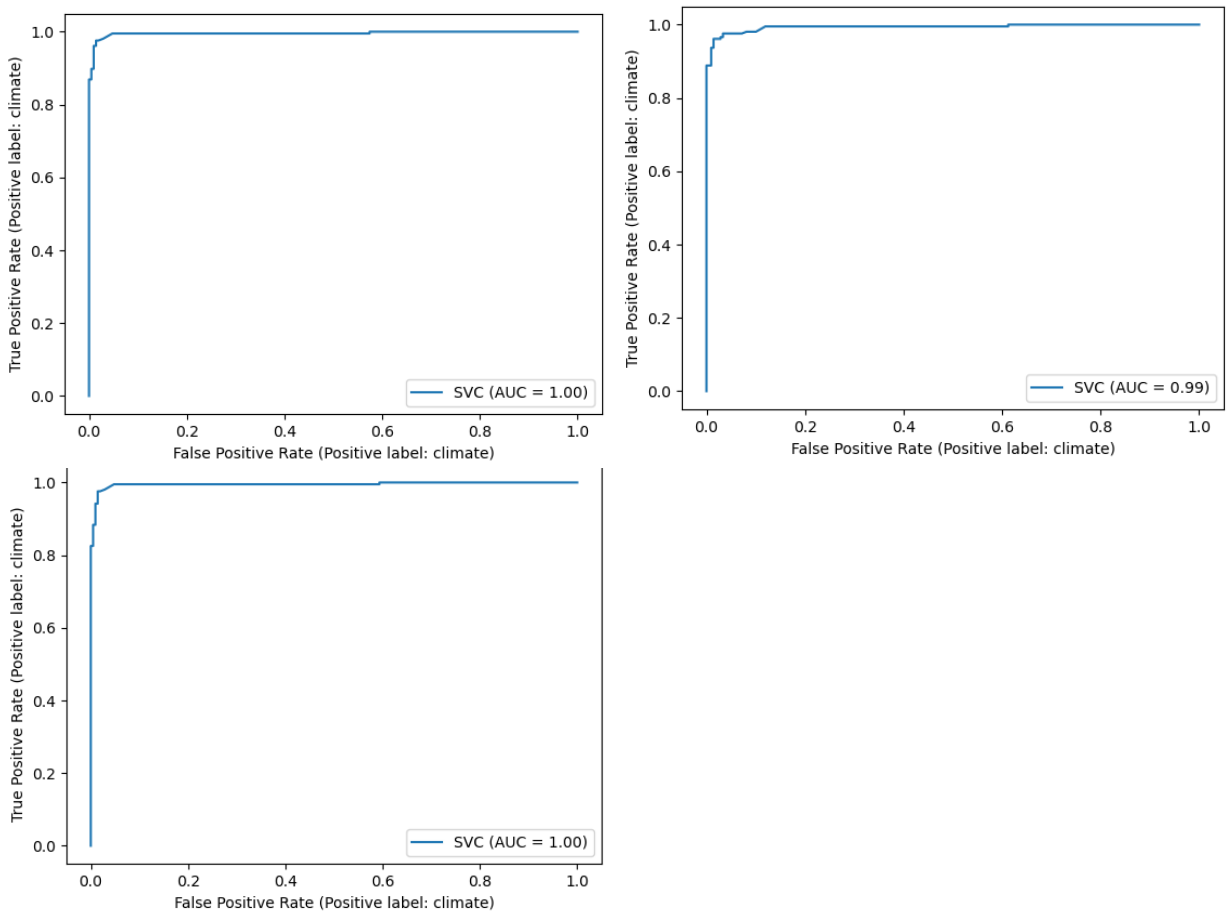

Figure 7: The ROC curves of the trained SVMs. Top left for gamma = 1000, top right for gamma = 0.0001 and bottom left for gamma = 100000

As can be seen, the best performance is obtained when the gamma value is 100000. From the results, the importance of the gamma parameter can be observed. A high gamma places emphasis on misclassified points, which allows the model to perform better on the test set. However, simply increasing the gamma to high values can easily lead to overfitting as well, so a high gamma value is not necessarily desirable.

For the soft margin SVM, it appears that the model only predicts one class no matter the data sample. All the samples are predicted as the climate class. This is possible since in the soft class SVM, there is no emphasis on the misclassified points, so the model does not care about getting a high accuracy score. Therefore, instead of learning a good margin, it simply focuses on minimizing w. Since there are 830 climate samples and 827 sports samples in the training set, the model only predicts the majority class.

However, the ROC curve of the classifier seems to be competitive, with an AUROC of 1. This is in contradiction with the accuracy and other performance metrics of the model. Let us attempt to explain why that could be. If there is not much difference in probability for the negative predictions and positive predictions, we can observe this ROC curve. Indeed, if we set the probability parameter in SVC to true, we can see that there is not much difference between the probability outputs for each class. Since the

probability of the positive class is slightly higher than the probability of the positive class when the sample is negative, the TPR value quickly jumps to 1 while the model is still capable of predicting the negative samples as negative. For example, if the probability of the positive classes is 52% and the negative classes is 48%, once the threshold is below 52%, TPR is immediately 1 while FPR is 0. However, as the threshold decreases, the model starts predicting everything as positive, which generates the point (1,1) in the ROC curve. If we connect the dots, we get the above figure even though the model has low accuracy. This implies that relying on a single metric for model comparison is not necessarily a good method and multiple metrics must be considered during model selection.

To choose the best gamma parameter, we can perform grid search through the possible combinations. The results of our search are as follows:

Table 2: Mean cross validation accuracies for SVMs with different gammas

Gamma	**Mean Cross Validation Accuracy**
0.001	50%
0.01	50%
0.1	94.2%
1	94.4%
10	95.0%
100	95.47%
1000	95.95%
10000	95.89%
100000	95.53%
1000000	95.53%

From the results, we can see that the best cross validation results are obtained when the gamma value is 1000.

The scores for this gamma parameter have already been calculated on the test set as follows:

Table 3: The performance of the best SVM on the test set

Gamma	**Accuracy**	**Precision**	**Recall**	**F-1**
1000	97.5%	98.5%	96.6%	97.5%

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

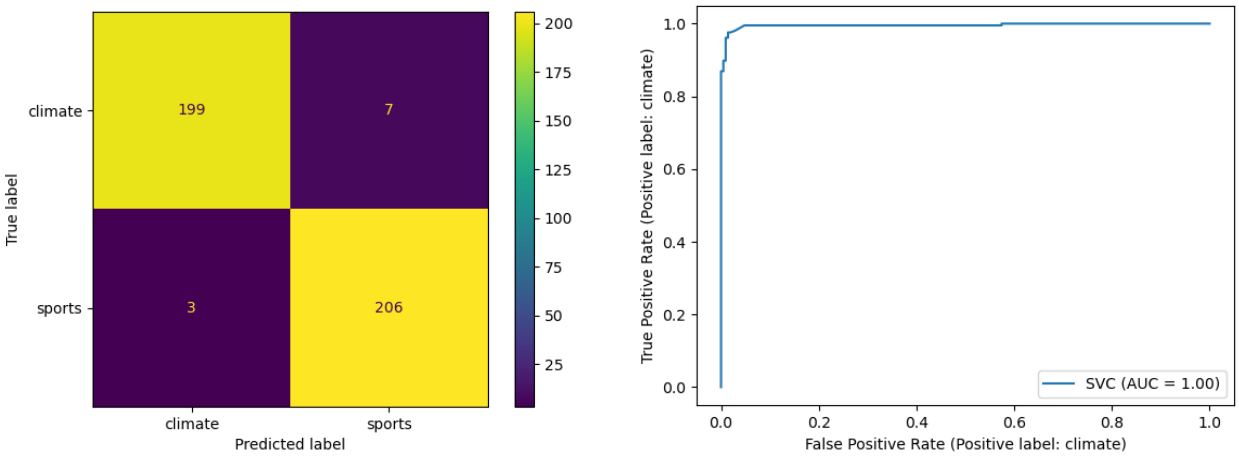

Figure 8: The confusion matrix and the ROC curve for the SVM classifier when the gamma parameter is 1000.

Logistic Regression

For this part of the assignment, we are tasked with fitting a logistic regression model to our dataset. The logistic regression model outputs the probability that a data sample is from the positive class using the following formula:

$$P(y = 1|x) = \frac{1}{1 + e^{-(wx+b}}$$

The w and b parameters are calculated by gradient descent so that they maximize the log likelihood.

We can also define a way to regularize the coefficients of the model. There are two main methods of doing so: **L1** and **L2 regularization.**

In L1 regularization, the following is added as a constraint during optimization:

$$\lambda|w|$$

In L2 regularization, the following is used:

$$\lambda w^2$$

As the λ value increases, the weights are more greatly penalized.

Question 6

If we train a logistic classifier without any penalty and test on the test data, we get the following results:

Table 4: The performance of the logistic regression model without penalty on the test set

Accuracy	**Precision**	**Recall**	**F-1**
97.8%	98.5%	97.0%	97.7%

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

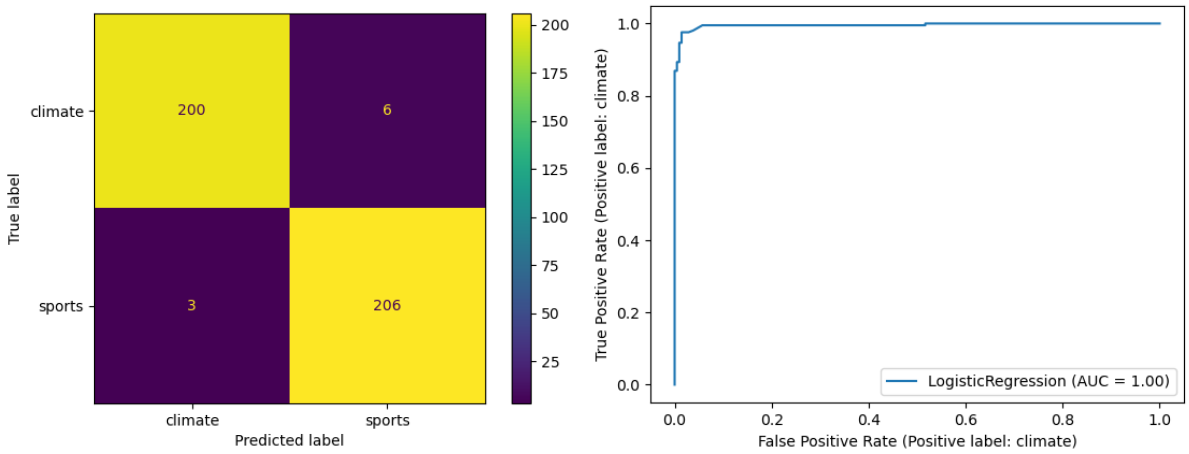

Figure 9: The confusion matrix and the ROC curve for logistic regression without regularization

Now, let us perform grid search to determine the optimal regularization strength for logistic regression with L1 and L2 losses. For L1 loss, our results are as follows:

Table 5: Mean cross validation accuracies for Logistic Regression with L1 penalty with different penalizations

Penalty	**Mean Accuracy**
0.0001	50%
0.001	50%
0.01	50%
0.1	90.8%
1	95.3%
10	95.9%
100	96.1%
1000	95.9%
10000	95.9%

From the results, we can see that the best penalty value for L1 loss is 100.

For L2 loss, we have:

Table 6: Mean cross validation accuracies for Logistic Regression with L2 penalty with different penalizations

Penalty	**Mean Accuracy**
0.0001	92.5%
0.001	94.4%
0.01	94.5%
0.1	94.7%
1	95.4%
10	95.3%
100	95.8%

1000	96.0766%
10000	96.0768%

From the above results, we can determine that the optimal penalty for L2 loss is 10000.

Now, let us train three classifiers (no regularization, L1 regularization and L2 regularization) with the determined optimal parameters and compare their performance on the test set.

Table 7: Performance of different logistic regression models

Model Type	**Accuracy**	**Precision**	**Recall**	**F-1**
No Regularization	97.83%	98.52%	97.08%	97.79%
L1 with penalty = 100	97.83%	98.52%	97.08%	97.79%
L2 with penalty = 10000	97.83%	98.52%	97.08%	97.79%

As can be seen, the change in loss when paired with parameters found through cross validation had little to no affect in the performance of our model.

However, if we observe the learned coefficients of each logistic regression, we can see that the coefficient of the L1 regularized model are mostly 0. Furthermore, the magnitudes of the coefficients of the L2 regularized model are smaller than those of the not regularized model. From here, it can be inferred that L1 penalty punishes the weights of coefficients more and sets the coefficients of not useful features to 0 while L2 offers a more even penalization across the weight coefficients. We may be interested in L1 regularization to determine important features and make our model simpler and L2 regularization when we want to shrink the parameters of our model more evenly instead of punishing certain weights severely.

SVM tries to find the decision boundary by finding the optimum hyperplane that separates the data points by maximizing the margin between the data points of different classes. On the other hand, logistic regression aims to find a separating hyperplane that works well using gradient descent. This is also apparent from the functions that the models are trying to optimize. In SVM, the goal is to increase the margin as much as possible and for logistic regression, the goal is to find the wight coefficients that maximize the log likelihood. The difference in performance can be attributed to the fact that while SVMs get the best hyperplane, logistic regression only attempts to find a good enough hyperplane through gradient descent. Therefore, the hyperplanes that the models learn are not equivalent, hence there is a difference in performance. However, as the difference in accuracy is only 0.33%, the difference is not that significant.

Naïve Bayes Model

On the final part of this section of the project, we are tasked with fitting a Naïve Bayes classifier on the training set. Naïve Bayes classifiers assume that the features of a dataset are independent and use Bayes rule when calculating its parameters. According to Bayes rule, we have:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

$$P(class|data) = \frac{P(data|class) * P(class)}{P(data)}$$

Maximizing the likelihood of $P(class|data)$ also implies maximizing $P(data|class)$. So, Naïve Bayes classifier attempts to find the probability distribution that maximizes this posterior estimation.

In our code, we used the GaussianNB class of sklearn to implement the Gaussian Naïve Bayes model.

After training the model, we tested the model on the test data. The results can be summarized as below:

Table 7: Performance of Gaussian Naïve Bayes on test set

Model	**Accuracy**	**Precision**	**Recall**	**F-1**
Gaussian NB	95.66%	94.76%	96.60%	95.67%

The ROC curve and the confidence matrix of this model are as follows:

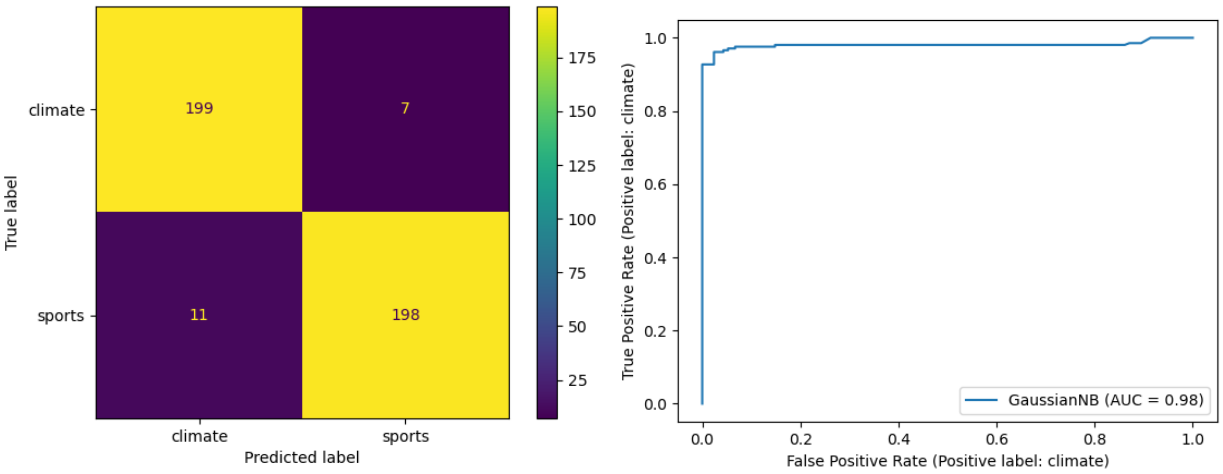

Figure 10: The confusion matrix and the ROC curve of the Gaussian Naïve Bayes model

Grid Search of Parameters

Question 8

Now that we see our models work and give satisfactory results, we can search the hyperparameter space using grid search. Here, we attempt to find the best combination among the following:

Table 8: Parameters that will be changed during grid search

Loading the data	Clean or not clean
Min_df	3 or 5
Feature Extraction	Lemmatization or Stemming or Nothing
Dimensionality Reduction	LSI or NMF
Dimension Size	5 or 50 or 500
Classifier	SVM(C=100) or LogisticRegressionL1(C=100) or LogisticRegressionL2(C=1000) or GaussianNB

Here, it is important to note that all feature extraction methods remove digits and stop words from the generated vocabulary. So, nothing is not just passing the text as it is. Lemmatization and stemming do their word manipulation after the word is checked for digits and stop words.

As can be seen, in total there are 288 possible combinations. Searching through all of them took approximately 18 hours to complete.

NOTE: This part of the project was completed before the discussion session that stated lemmatization must be performed sentence by sentence instead of on the entire data sample. Since this is the case, the operation of lemmatization and the parameters for best classifiers are different than what is stated in the earlier sections of the report. However, as also discussed with the TA, due to the lengthy running time of grid search it is infeasible to run the code again with the modified lemmatization function and the newly found best parameters for the models. Since the goal of this exercise is to see how to perform grid search and parameter optimization and since our experimental results are also satisfactory, we can consider our implementation sufficient.

After the grid search operation, the best 5 best parameter combinations in terms of mean cross validation performance are found to be as follows:

Table 9: Parameter combinations that have the best cross validation performance

	Clean?	**Min_df**	**Feature Extraction**	**Dimensionality Reduction**	**Dimension Size**	**Classifier**	**Mean Accuracy**
Comb 1	False	5	Nothing	LSI	500	LogisticRegL2(C=1000)	96.9812%
Comb 2	True	3	Stem	LSI	500	LogisticRegL2(C=1000)	96.921%
Comb 3	False	3	Nothing	LSI	500	LogisticRegL2(C=1000)	96.921%
Comb 4	True	5	Nothing	LSI	500	LogisticRegL2(C=1000)	96.9208%
Comb 5	False	5	Lemmatization	LSI	500	LogisticRegL1(C=100)	96.9202%

It is interesting to see that cleaning does not affect the performance much. Further investigation showed that this is because the CountVectorizer used to separate the document to its words automatically removes some HTML tags.

Let us now see the performance of these combinations on the test set. For these tests, the metrics are computed by taking the positive class as the climate class.

Table 10: Performance of the best combinations on the test set

	Accuracy	**Precision**	**Recall**	**F-1 Score**
Comb 1	98.31%	100%	96.60%	98.27%
Comb 2	98.31%	99.5%	97.08%	98.28%
Comb 3	98.07%	100%	96.11%	98.01%
Comb 4	98.07%	100%	96.11%	98.01%

Comb 5	98.31%	99.5%	97.08%	98.28%

As can be seen from the table above, the combinations give similar performances on the test set. The combination with the highest cross validation performance also performed the best on the test set, so it can be understood that cross validation in our project is an accurate method to estimate performance on the test set. As all of the combinations offer satisfactory performance, we can conclude that the grid search process was successful.

Multiclass Classification

In this part of the project, we are asked to perform multiclass classification of the data that we have. To do so, we utilize the same features used in the binary classification problem. However, we change the labels used to the leaf_label column.

Question 9

We first utilized the Gaussian Naïve Bayes classifier to perform multiclass classification. The advantage of using this model was that it was readily suitable for multiclass classification without any needed modifications. To calculate the metrics for multiclass classification (accuracy, precision, recall, etc.), we specified the functions to perform macro averaging. Here, the metrics were calculated individually for each class in the dataset. Then, the average of the results were calculated. This average became the reported result for the multiclass classifier. The result for the Gaussian Naïve Bayes is as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

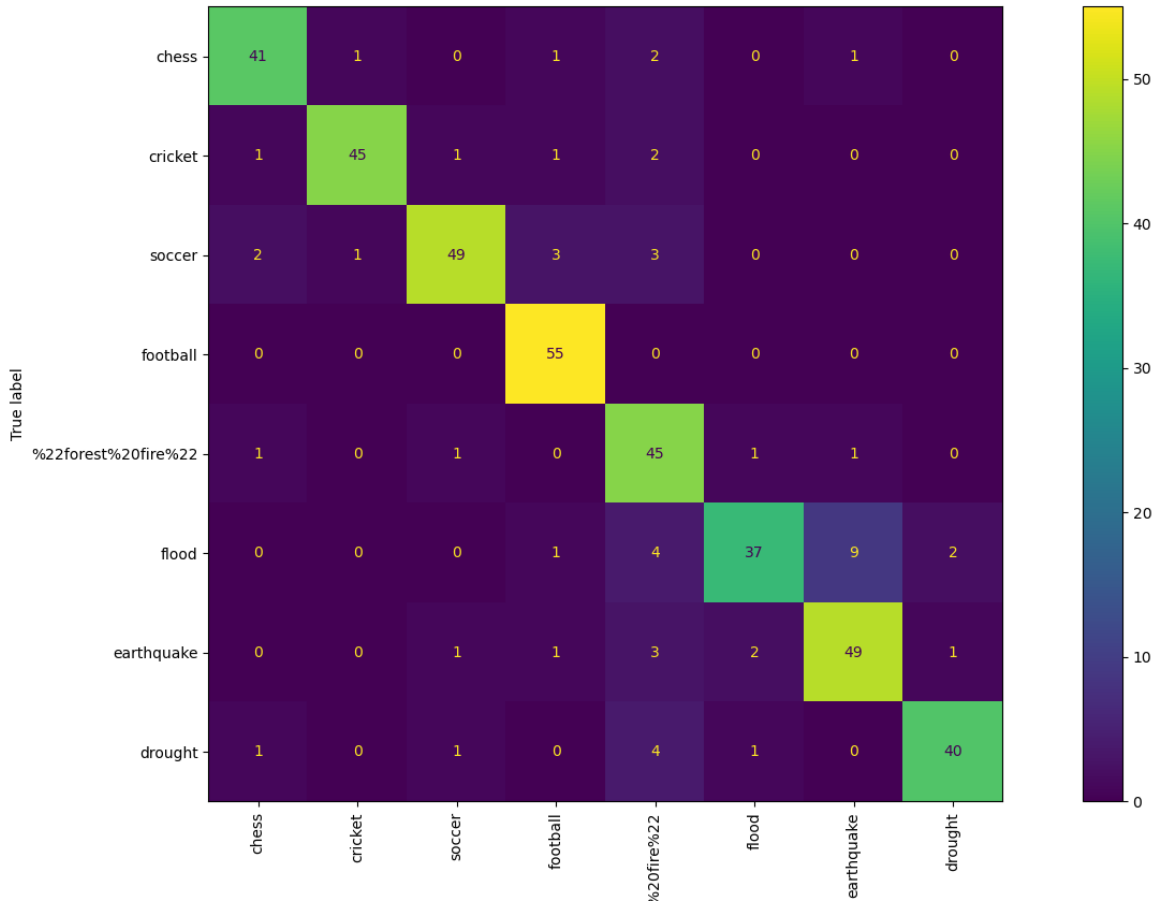

Figure 11: The confusion matrix of the Gaussian Naïve Bayes model

Table 11: Performance of the Gaussian Naïve Bayes model for multiclass classification on the test set

Model	**Accuracy**	**Precision**	**Recall**	**F-1**
Gaussian NB	86.98%	87.80%	87.04%	86.99%

After fitting this model, our task was to use the SVM model to perform multiclass classification. However, SVM in its default form is only suitable for binary classification. To overcome this, we used the One vs Rest (OvR) and One vs One (OvO) approach.

In the OvR approach, we fit multiple SVMs to our data. Here, each SVM is responsible for differentiating one class from all the other classes. For example, one SVM is responsible for determining if a data sample has label chess or not, the other determines if the label is flood or not, etc. In total, there are as many models as the number of classes for this approach.

However, it can be understood that in the OvR model, there is a huge class imbalance, since a single class is being compared to all the others for each SVM. In order to combat this problem, we used the “balanced” class weighting option in the SVC class to even out the imbalance. The “balanced” mode automatically adjust weights inversely proportional to the class frequencies in the input data as n_samples / (n_classes * bincount(class)). This leads to a higher penalization of the model for

misclassifying samples of less frequent labels during training. Doing so allows us to get over the class imbalance issue.

In the OvO approach, each SVM model differentiates between two classes. As such, there are as many models as the amount of possible class combinations. Since there are the same amount of samples for each class, this approach does not suffer from the class imbalance problem in OvR.

While creating the OvO and OvR classifiers, each SVM had a C value of 1000, as this value was found to be the optimal value for binary classification. The results for OvO and OvR approaches are as follows:

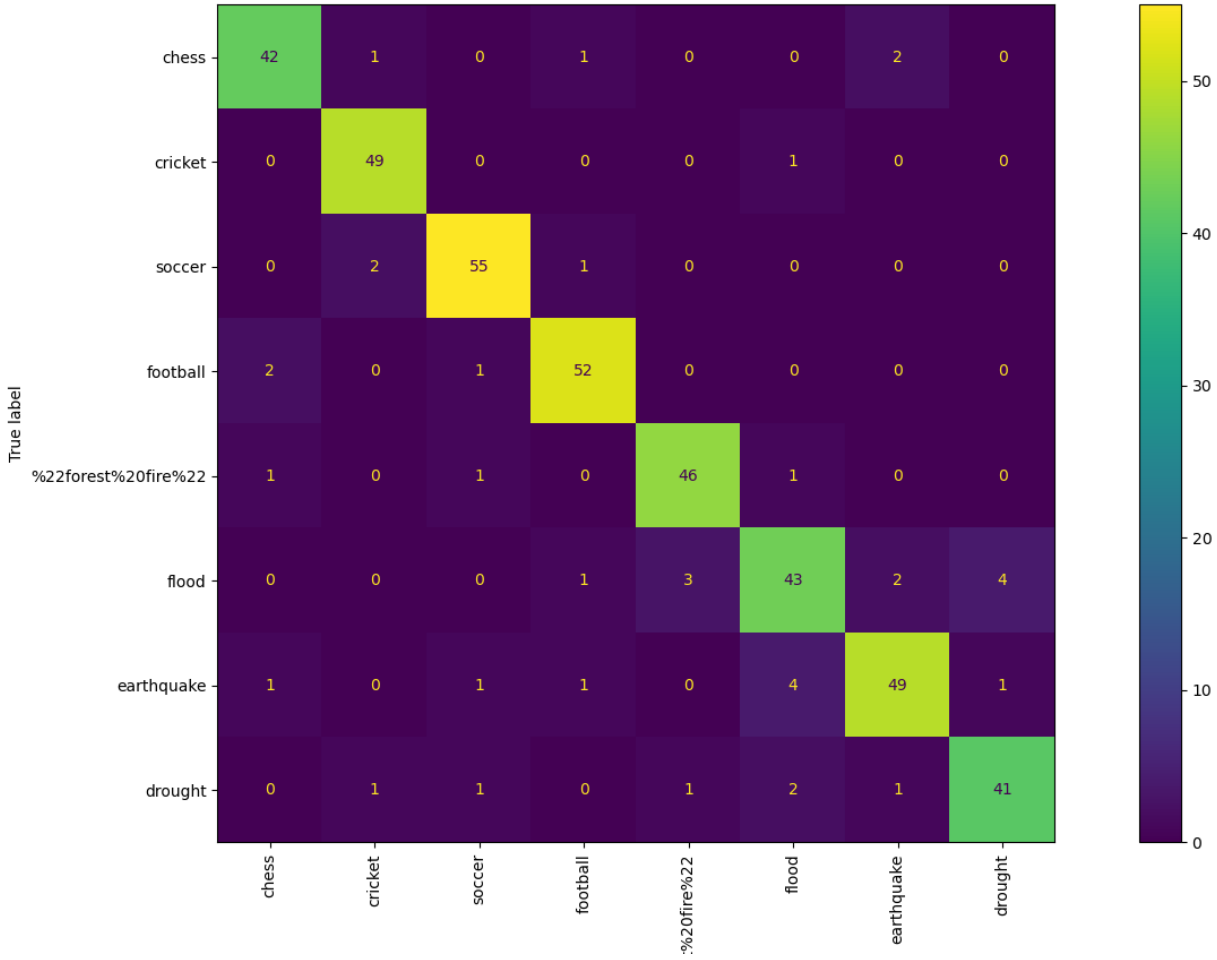

Figure 12: The confusion matrix of the One vs One model

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

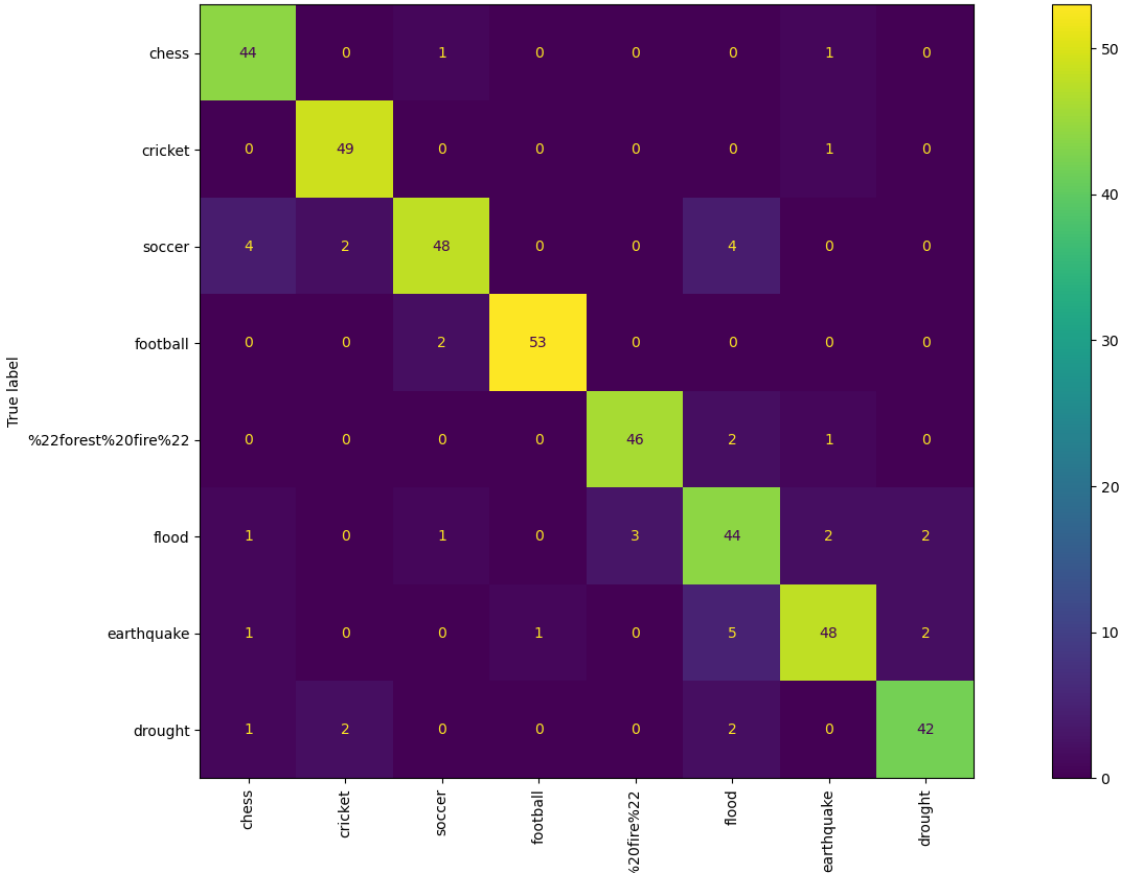

Figure 13: The confusion matrix of the One vs Rest model

Table 12: Performance of the OvO and OvR models for multiclass classification on the test set

Model	**Accuracy**	**Precision**	**Recall**	**F-1**
OvO	90.84%	90.75%	90.86%	90.78%
OvR	90.12%	90.26%	90.40%	90.23%

By examining the confusion matrices, especially for that of the Gaussian Naive Bayes model, there are distinct blocks along the major diagonal for flood and earthquake. This implies that there are a high number of datapoints that belong to one of the classes that is being misclassified as the other class. In short, we can understand that the model is confusing samples of the two classes.

Based on this observation, we can combine the flood and earthquake labels as a merged label. Doing so should give an increase in accuracy. The results of all three models after merging the labels are as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

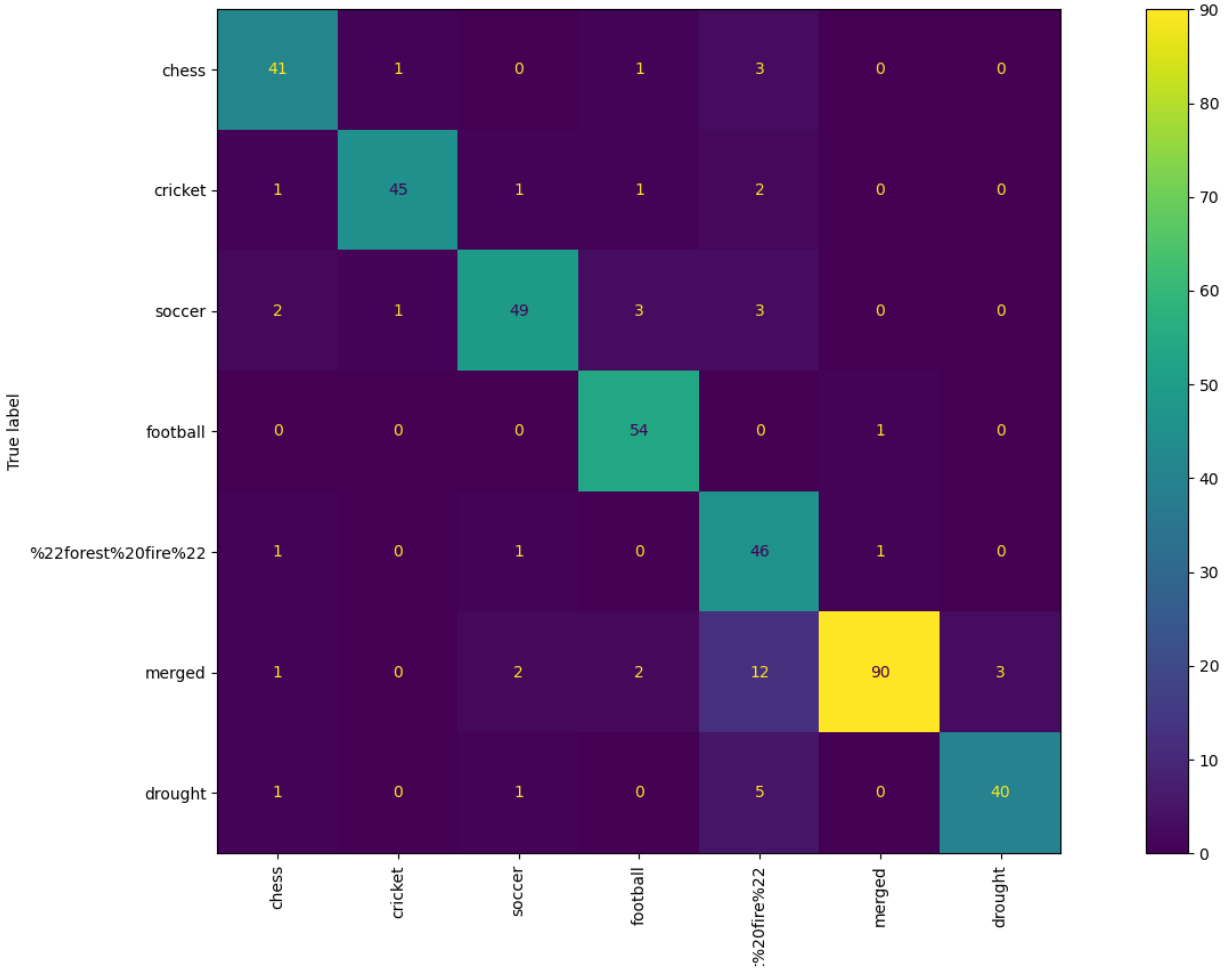

Figure 14: The confusion matrix of the Gaussian Naïve Bayes model after merging classes

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

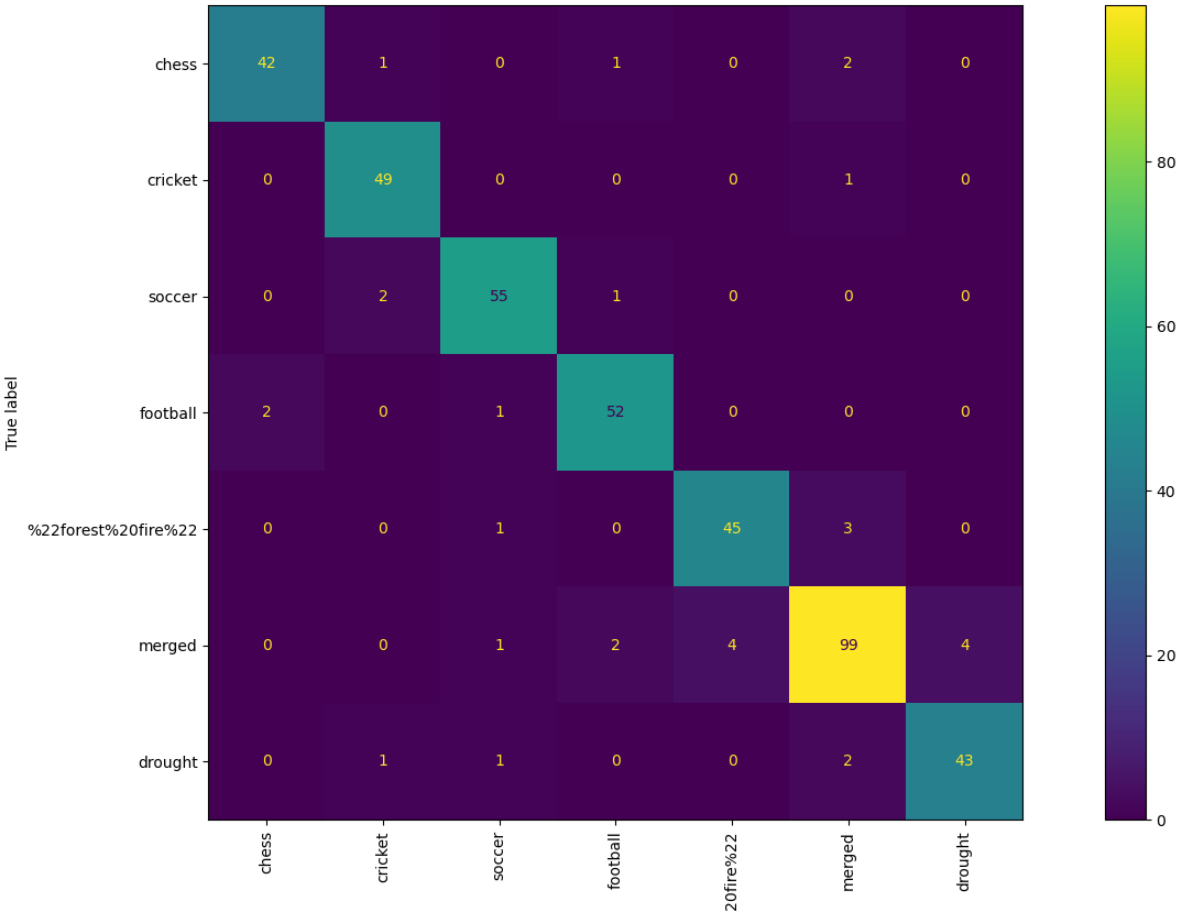

Figure 15: The confusion matrix of the One vs One model after merging classes

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

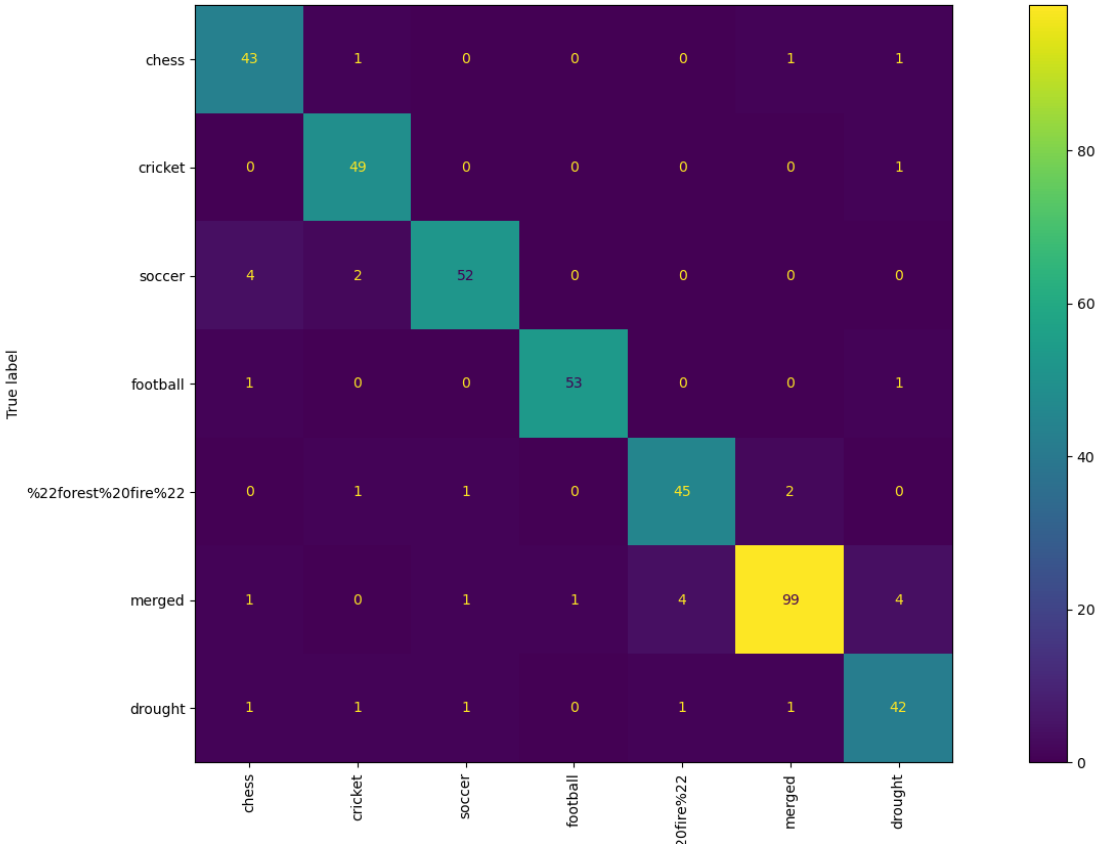

Figure 16: The confusion matrix of the One vs Rest model after merging classes

Table 13: Performance of the multiclass models after merging flood and earthquake classes

Model for merged label	**Accuracy**	**Precision**	**Recall**	**F-1**
Gaussian NB	87.5%	88.26%	88.94%	88.03%
OvO	92.77%	92.83%	93.14%	92.96%
OvR	92.28%	91.60%	92.67%	92.06%

As can be seen from the results, the accuracy of OvO increased by **1.93%** and the accuracy of OvR increased by **2.16%**.

For the previous part, we turned off class weighting based on class frequency. Normally, we expect both OvR and OvO to suffer from class imbalance, since now, some the SVMs in OvO compare the merged class with another class that has half the data samples. However, the class imbalance problem does not seem to affect the accuracy of the models too significantly. Nevertheless, let us now train the models with class frequency-based weighting enabled for the OvO and OvR models. The results are as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

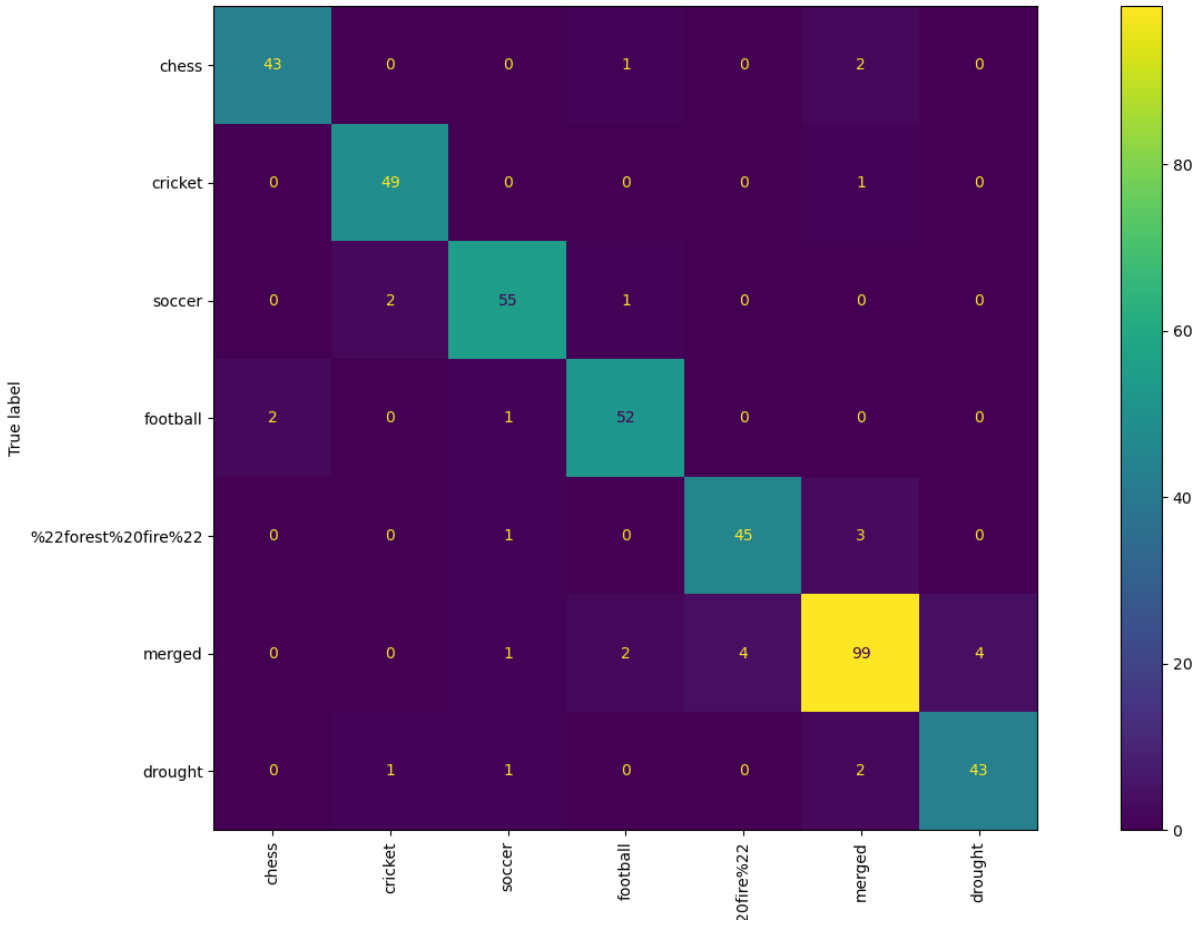

Figure 17: The confusion matrix of the One vs One model after merging classes and dealing with class imbalance

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

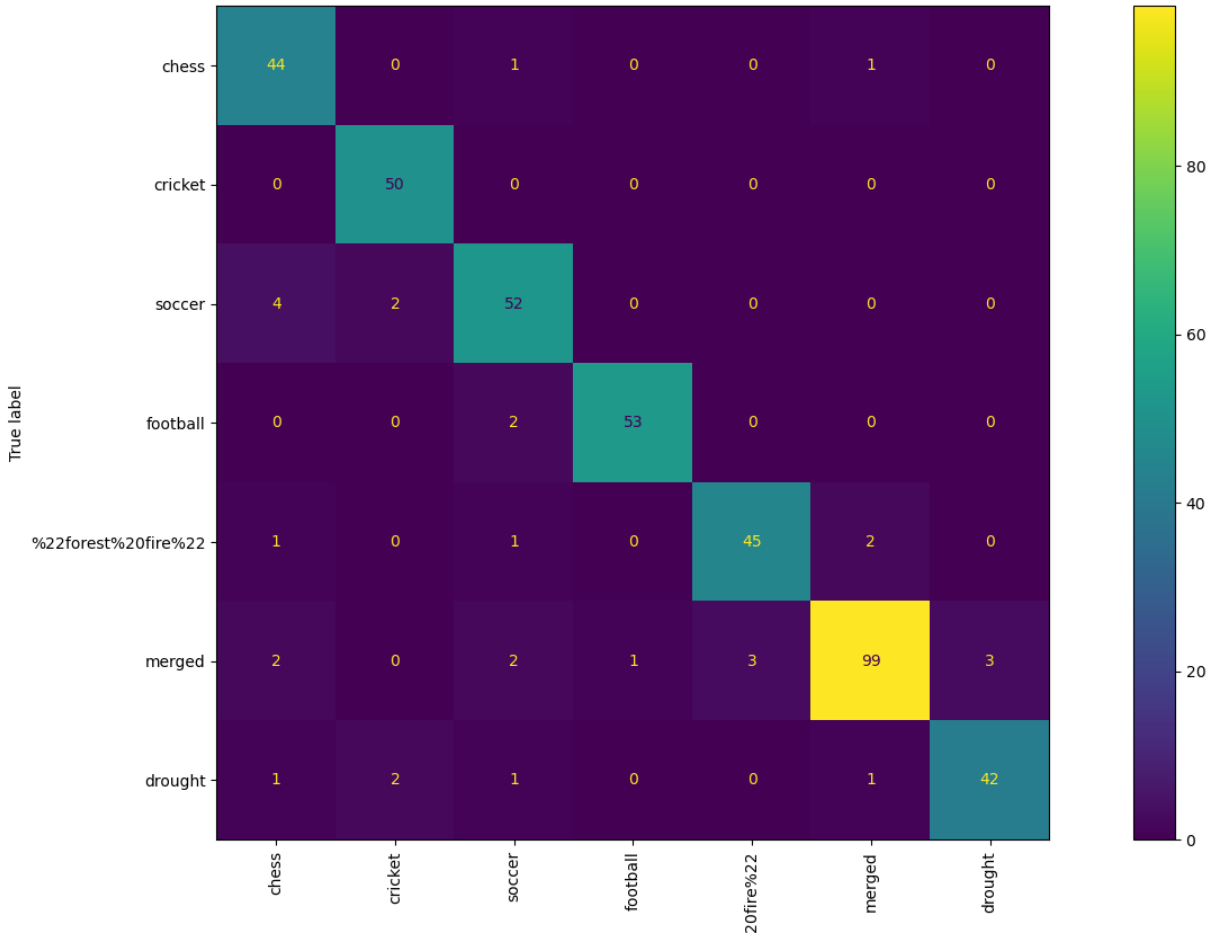

Figure 18: The confusion matrix of the One vs Rest model after merging classes and dealing with class imbalance

Table 14: Performance of the OvO and OvR models after merging classes and balancing misclassification

Model for merged label with class weighting	**Accuracy**	**Precision**	**Recall**	**F-1**
OvO	93.01%	93.10%	93.45%	93.26%
OvR	92.77%	92.38%	93.26%	92.73%

As can be seen, there is a slight increase of **0.24%** for OvO and **0.49%** for OvR in accuracy after frequency based class weighting.

Word Embedding

In the final section of the project, we are asked to use pretrained GLoVE embeddings to generate our features for each data sample. GLoVE is a method to turn words into vector forms. As these embeddings are generated based on sensical proximity of the words, they offer a useful tool to train NLP models.

Question 10

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

GLoVE embeddings are trained on the ratio of co-occurrence probabilities rather than the probabilities themselves because the co-occurrence gives a sense about the relationship between two words, which is why it is used instead of the probabilities on their own. This encoded relationship allows us to distinguish between irrelevant words and better discriminate relevant words.

GLoVE is context independent. For example, in sentences “James is running in the park.” and “James is running for the presidency”, GLoVE would return the same vector for running for both sentences. This is because the embeddings in GLoVE are pretrained and do not take into account the context of the word when generating the embeddings.

We expect the value of *|GLoVE['queen'] - GLoVE['king'] - GLoVE['wife'] + GLoVE[‘husband']|* to be close to 0. This is because the relationship between queen and king is the same relationship between husband and wife. By the same logic, we expect the values of *|GLoVE[‘queen'] - GLoVE['king']|* and *|GLoVE['wife'] - GLoVE[‘husband’]|* to be about the same as well. Indeed, after loading the GLoVE embeddings, the results of these operations are found to be as follows:

|GLoVE['queen'] - GLoVE['king'] - GLoVE['wife'] + GLoVE[‘husband']| = 6.16

|GLoVE[‘queen'] - GLoVE['king']| = 5.96

|GLoVE['wife'] - GLoVE[‘husband’] | = 3.15

As can be seen, the obtained results are similar to our predictions.

It is better to lemmatize the word instead of stemming before mapping the word to GLoVE embedding. This is because there is a high probability that stemming will result in a nonsensical word which will not be present in the dictionary of GLoVE.

Question 11

In our implementation, we used the keywords column of the dataset. For each data sample, first we stored the associated keywords in a list. Then, if a word in this list was part of the GLoVE dictionary, we got its embeddings. We summed all the obtained embeddings to get the final feature vector for a data sample or document. Doing this process for all the data samples in the dataset allowed us to convert our dataset into one that can be used by an ML model.

As for our classifier, we chose to use the Logistic Regression method with L2 penalty. The results after training this model on the features obtained by GLoVE embeddings to perform binary classification are as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

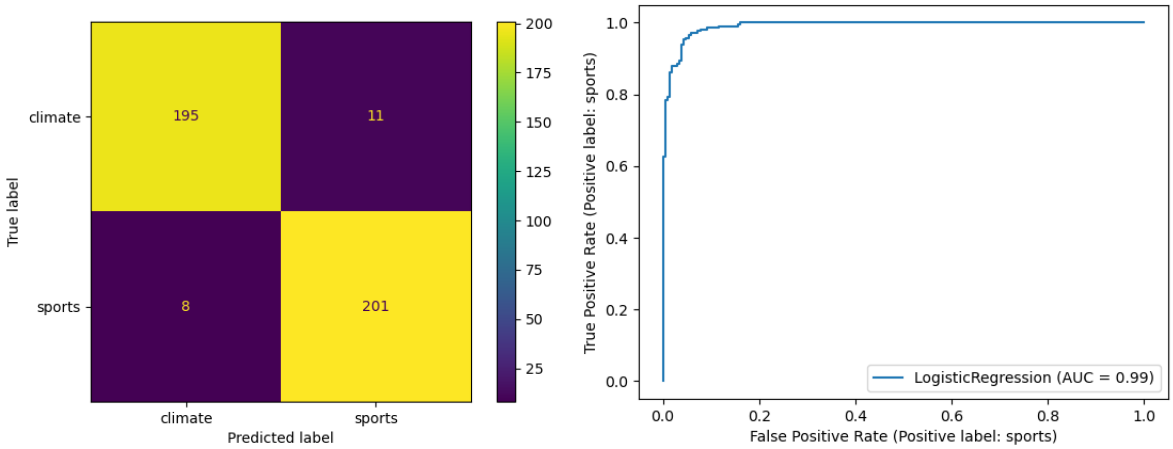

Figure 19: The confusion matrix and the ROC curve of the L2 logistic regression trained with features obtained from GLoVE embeddings

Table 15: Performance of L2 logistic regression trained with features obtained from GLoVE embeddings

Model	**Accuracy**	**Precision**	**Recall**	**F-1**
L2 Logistic Regression	95.42%	94.81%	96.17%	95.48%

Question 12

Let us know examine the relationship between the dimension of the GLoVE embeddings and the accuracy of the L2 logistic regression model on the test set. To do this, we loaded GLoVE embeddings of different length, used them to generate features, trained an L2 logistic regression model on these features then tested the model on the test set. The resulting graph is as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

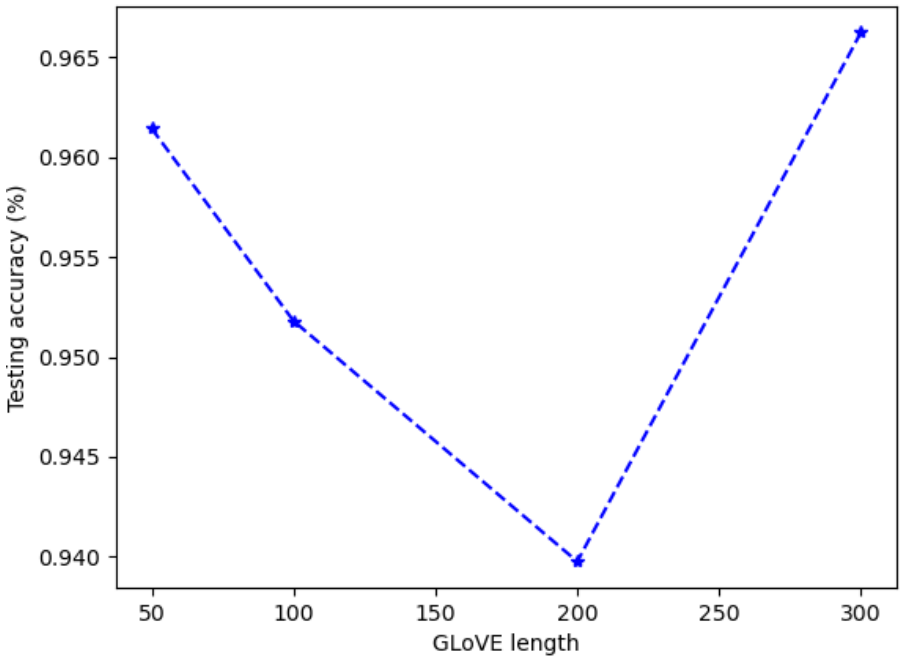

Figure 20: The change in test accuracy vs. the length of the GLoVE embeddings

As can be seen from the graph, the accuracy has a trend to decrease as the length of the GLoVE embeddings increase. However, there is a jump in accuracy when the length is 300. The decreasing trend is expected, since we are increasing the number of features without changing the number of data samples. Because of this, after a while having too many dimensions leads to a loss in performance due to the curse of dimensionality. However, the last jump in accuracy is not expected.

Question 13

In this part of the project, we are asked to visualize the features of the documents generated by GLoVE embeddings and a set of randomly generated features using UMAP. The results are as follows:

Deniz Orkun Eren – UID: 905624625
Matthew Waliman – UID: 605848839

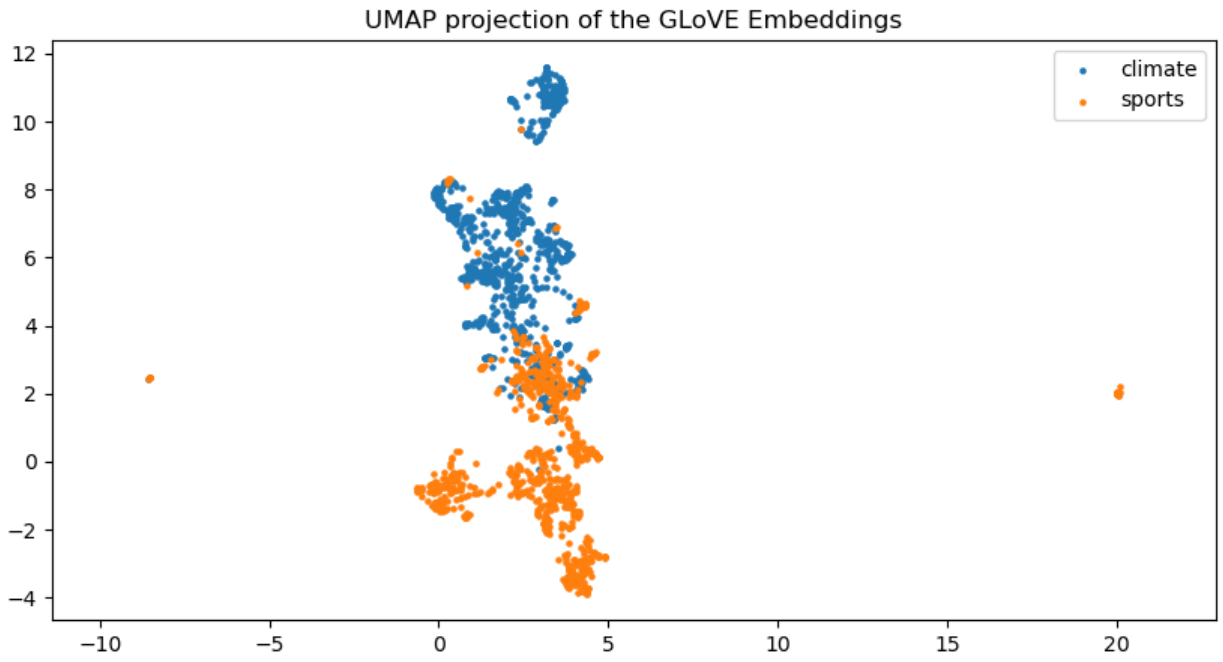

Figure 21: The UMAP projection of GLoVE embeddings

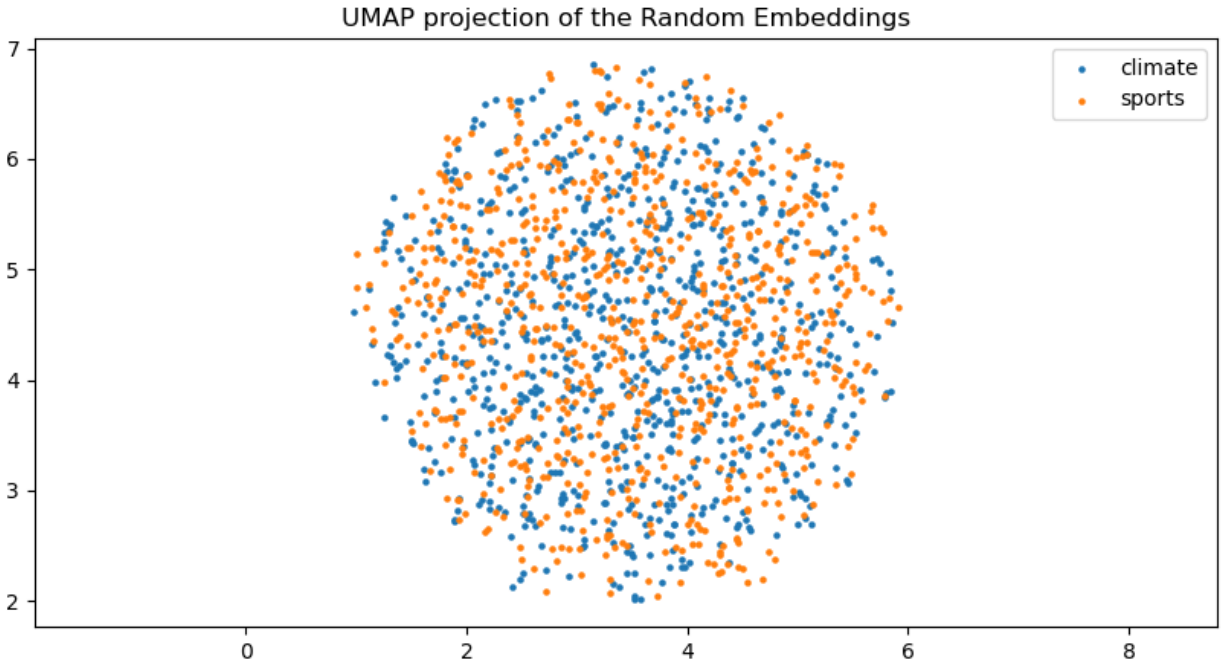

Figure 22: The UMAP projection of random embeddings

As can be seen from the plots, there are obvious clusters in the UMAP visualization when the features obtained from GLoVE embeddings are visualized. Here, there is a cluster for each class in the dataset. For the randomly generated features, there does not seem to be a cluster present. All the data seems to have been bundled together. This implies that the features generated from the glove embeddings are useful to separate the two classes.