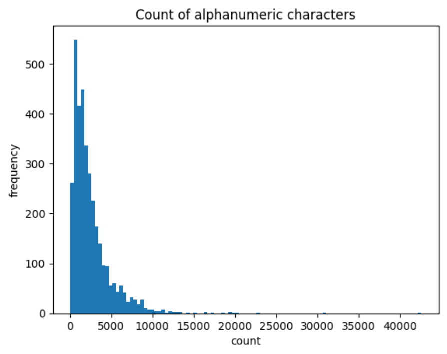
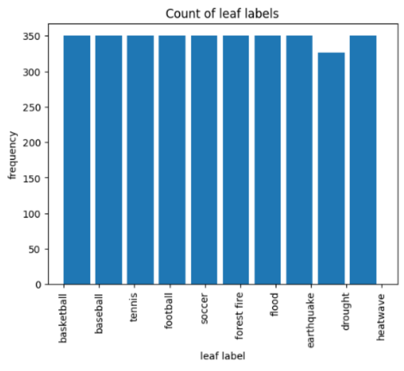
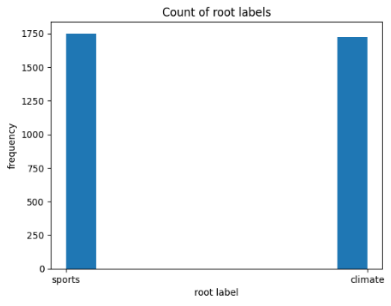
EC ENGR 219 Project 1

Q1.

Rows: 3476; Columns: 8







The "count of alphanumeric characters" histogram shows that majority of the data points have less than 5000 alphanumeric characters in the "full_text" feature, with most data points having around 2000 alphanumeric characters. The "count of leaf labels" histogram shows that there are generally equal number of data points for each leaf label, except for the "drought" leaf label that has slightly less data points than the other 9 labels. The "count of root labels" histogram similarly shows that there are about the same number of data points for each of the 2 root labels, with the "climate" root label having slightly less data points than the "sports" root label.

Q2.

Number of training samples: 2780

Number of testing samples: 696

Q3.

Lemmatization is more accurate than stemming because it actually analyzes the sentence structure using linguistic tools to determine the role of a word within a sentence, making it more accurate in finding the lemma of a word, while stemming simply chops off prefixes, suffixes and other common additions to a word, which can more easily result in erroneous changing of a word when trying to find the base form. However, because of the use of linguistic tools and a dictionary of words, lemmatization is more computationally expensive than stemming. Both reduce the dictionary size because words with the same base forms / lemma but with additional prefixes or suffixes will be reduced down to the same base forms / lemma, so number of unique words after both preprocessing techniques will decrease.

Increasing the min_df value means more terms that appear less frequently in the documents will be discarded (the filter for minimum occurrences of terms is more restrictive), so the TF-IDF matrix will be sparser and contain fewer non-zero values.

Stopwords, punctuations and numbers should be removed after lemmatization step, because these words (that will eventually be removed) are essential for the lemmatizer to decipher their role in the sentence. Keeping these words in when passing the sentence into the lemmatizer enables it to correctly tag the important words.

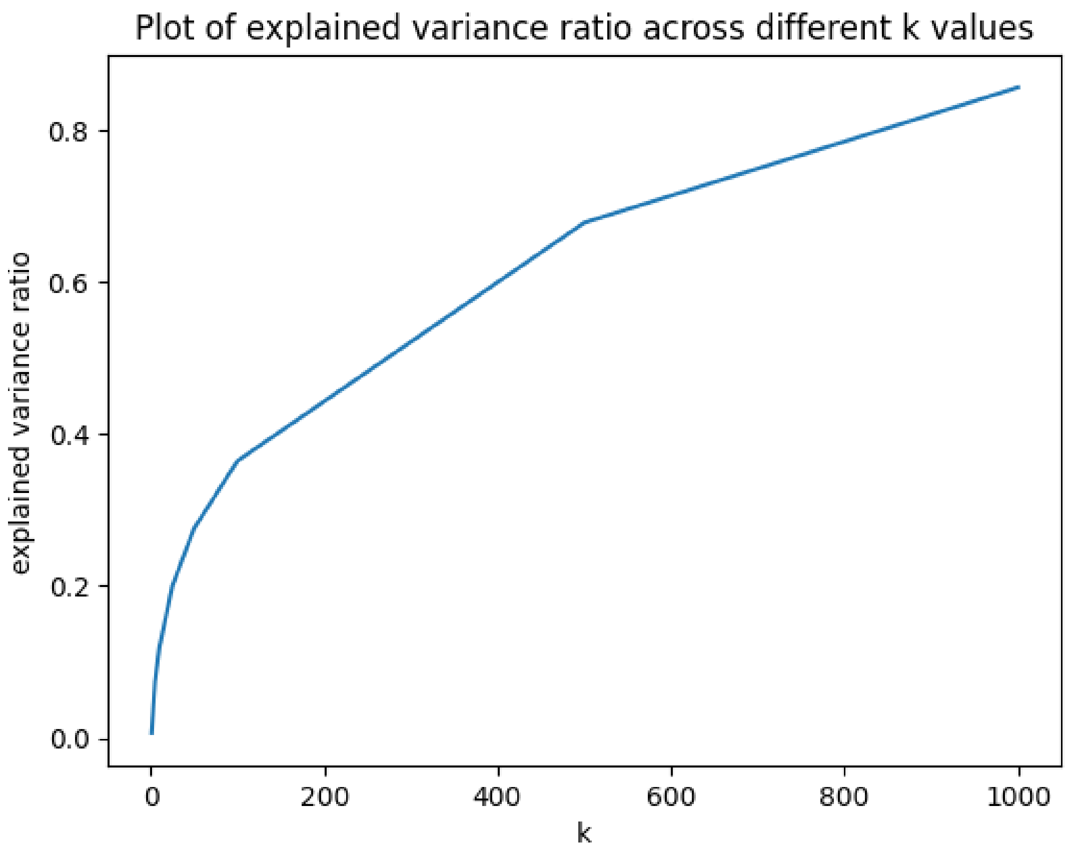
Train TFIDF shape:

Rows: 2780; Columns: 13288

Test TFIDF shape:

Rows: 696; Columns: 13288

Q4.



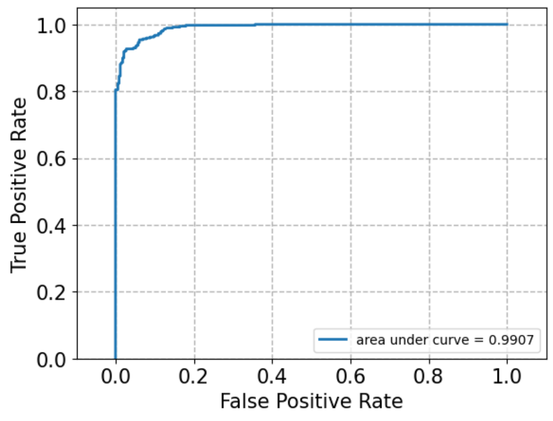
Concavity of the plot shows that with higher k values (or greater dimensionality / more principal components), the total variance explained by the principal components increases, albeit at a lower rate. This suggests that there is a certain number of principal components where the explained variance ratio is good enough such that we do not need more principal components.

NMF error: 2170.586363338054

#TODO: calculate LSI error and compare with NMF

Q5.

Using pipeline svm_hard



Confusion matrix:

```
[[309  19]
 [ 20 348]]
```

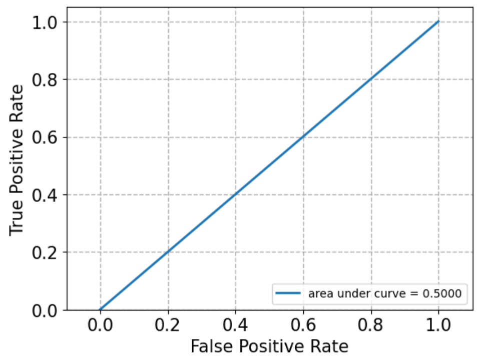
Accuracy score:0.9439655172413793

Recall score: 0.9456521739130435

Precision score: 0.9482288828337875

F1 score: 0.9469387755102041

Using pipeline svm_soft



Confusion matrix:

```
[[328   0]
 [368   0]]
```

Accuracy score:0.47126436781609193

Recall score: 0.0

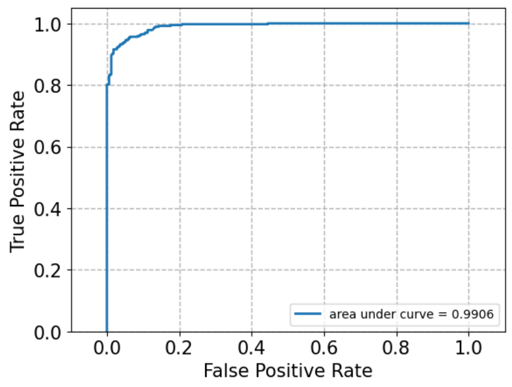
Precision score: 0.0

F1 score: 0.0

The hard margin SVM performed better, due to the better stats like accuracy, f1 score etc. For SVM with gamma = 100000, the stats are the same as that of the earlier hard margin SVM where gamma = 2000.

The soft margin SVM has a bad accuracy score, poor recall, precision, and f1 scores, and in fact predicts the same root label for all cases. From the confusion matrix, the true positive is very high, but the true negative is low. False positive is also very high. Since this is a binary classification (only two root labels), the soft margin SVM is predicting the same positive label for all cases, so only true positives and false positives are high. The ROC curve supports this, since the ROC curve for the soft margin SVM is on the diagonal line, representing the SVM is behaving like a random classifier.

Best gamma after 5-fold validation: 100



Confusion matrix:

```
[[309  19]
 [ 20 348]]
```

Accuracy score:0.9439655172413793

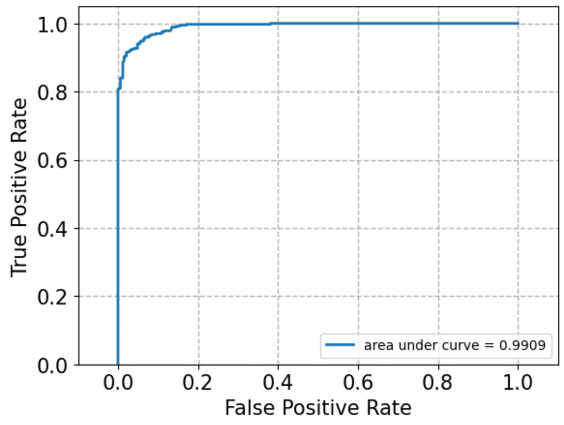
Recall score: 0.9456521739130435

Precision score: 0.9482288828337875

F1 score: 0.9469387755102041

Q6.

Logistic classifier without regularization:



Confusion matrix:

```
[[310  18]
 [ 20 348]]
```

Accuracy score:0.9454022988505747

Recall score: 0.9456521739130435

Precision score: 0.9508196721311475

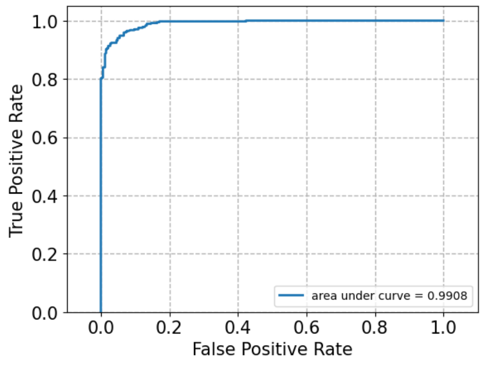
F1 score: 0.9482288828337875

After 5-fold cross-validation,

Best L1 regularization strength: 0.01

Best L2 regularization strength: 1e-05

Stats of logistic regression model with best L1 regularization strength:



Confusion matrix:

```
[[309  19]
```

```
 [ 19 349]]
```

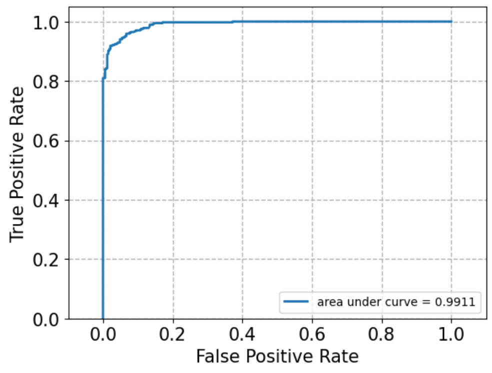
Accuracy score:0.9454022988505747

Recall score: 0.9483695652173914

Precision score: 0.9483695652173914

F1 score: 0.9483695652173914

Stats of logistic regression model with best L2 regularization strength:



Confusion matrix:

```
[[310  18]
 [ 20 348]]
```

Accuracy score:0.9454022988505747

Recall score: 0.9456521739130435

Precision score: 0.9508196721311475

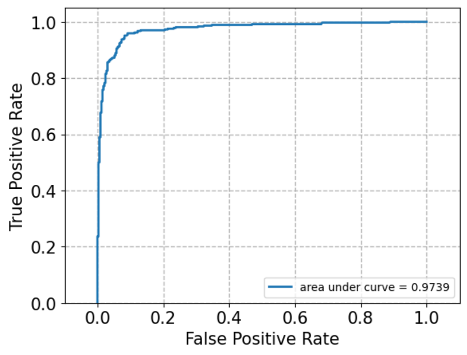
F1 score: 0.9482288828337875

By comparing accuracy, recall, precision and f1 scores, the logistic regression model with L1 regularization (strength=0.01) has the best performance.

Regularization parameter increases training error, which tends to reduce learnt coefficients (or weights) and prevents overfitting and improves generalizability. This tends to reduce test errors and make the models more robust. Different kinds of regularization leads to different effects on the model, so the appropriate regularization technique should be employed based on what is needed. L1 regularization performs more feature selection by making more weights become zero, causing greater sparsity and thus reducing dimensionality. L2 regularization is better for distributing the impacts of more correlated variables across different features, without eliminating those features altogether, thus capturing more information while still reducing overfitting.

Linear SVMs and logistic regression use different loss functions (maximizing margins through hinge loss for linear SVM, and estimating probabilities through log loss for logistic regression), so naturally they have different ways to find boundaries and will usually arrive at different decision boundaries. Performances may differ based on the intrinsic assumptions of the data, where a certain loss function may be more suited to find a clearer boundary for one set of data, but not for the other dataset. Empirically-speaking, they have similar performance and do not have statistically significant differences.

Q7.



Confusion matrix:

[[280 48]

 [11 357]]

Accuracy score:0.9152298850574713

Recall score: 0.970108695652174

Precision score: 0.8814814814814815

F1 score: 0.9236739974126779

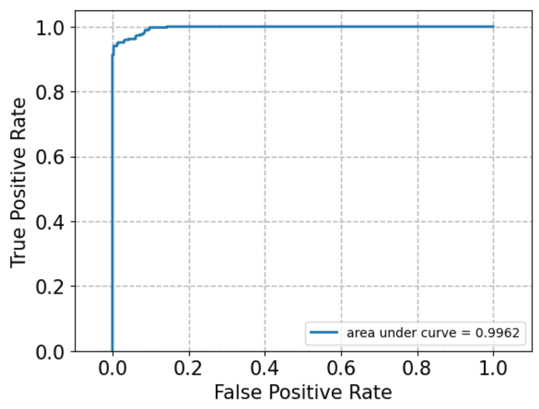
Q8.

Top 5 combinations (determined by best average validation accuracy)

Ranking	Feature Extraction	Dimensionality Reduction	Classifier	Average validation score
1	Stemming, mindf=2	NMF, k=100	Logistic Regression, L1 strength=0.01	0.960431654676259
2	Stemming, mindf=5	NMF, k=100	Logistic Regression, L1 strength=0.01	0.9597122302158274
3	Lemmatization, mindf=5	NMF, k=100	SVM, gamma=100	0.9597122302158272
4	Lemmatization, mindf=2	LSI, k=100	SVM, gamma=100	0.9593525179856115
5	Lemmatization, mindf=5	NMF, k=100	Logistic Regression, L1 strength=0.01	0.9593525179856115

Performances on test set:

Ranking 1:



Confusion matrix:

[[318 10]

 [16 352]]

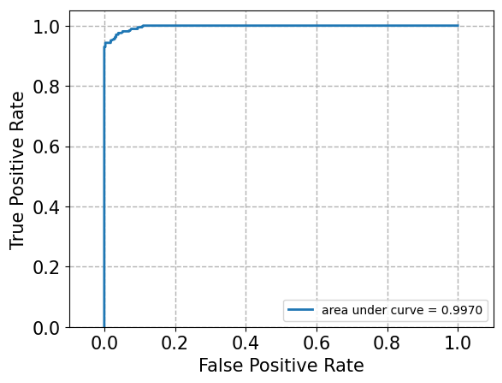
Accuracy score:0.9626436781609196

Recall score: 0.9565217391304348

Precision score: 0.9723756906077348

F1 score: 0.9643835616438357

Ranking 2:



Confusion matrix:

[[320   8]

 [17 351]]

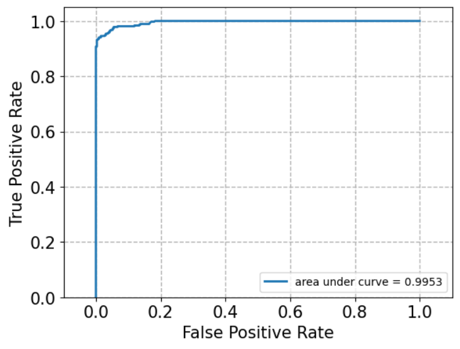
Accuracy score:0.9640804597701149

Recall score: 0.9538043478260869

Precision score: 0.9777158774373259

F1 score: 0.9656121045392022

Ranking 3:



Confusion matrix:

```
[[315  13]
 [ 16 352]]
```

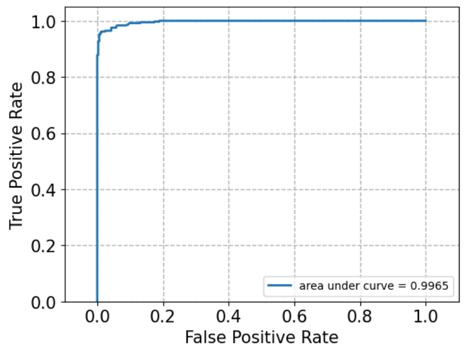
Accuracy score:0.9583333333333334

Recall score: 0.9565217391304348

Precision score: 0.9643835616438357

F1 score: 0.9604365620736699

Ranking 4:



Confusion matrix:

```
[[324   4]
 [ 14 354]]
```

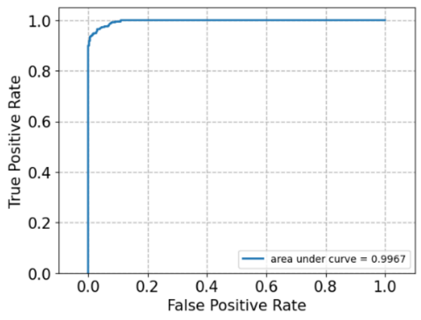
Accuracy score:0.9741379310344828

Recall score: 0.9619565217391305

Precision score: 0.9888268156424581

F1 score: 0.9752066115702479

Ranking 5:



Confusion matrix:

```
[[319   9]
 [ 17 351]]
```

Accuracy score:0.9626436781609196

Recall score: 0.9538043478260869

Precision score: 0.975

F1 score: 0.9642857142857143

Q9.

Naïve Bayes classifier:

Confusion matrix:

```
[[47  2  4  4  1  0  0  0  0  0]
 [ 0 41 17  0  1  0  2  2  0  0]
 [ 0  2 58  0  0  0  1  3  0  1]
 [ 0  9  1 62  0  0  0  1  0  0]
 [ 0  4 10  1 59  0  0  1  0  0]
 [ 0  1  4  0  0  2  1  3  2 50]
 [ 0  0  2  0  0  2 64  1  2  1]
 [ 0  2  3  0  0  0  1 77  0  0]
 [ 0  1  3  0  0  0  1  6 45  5]
 [ 0  4 11  0  0 10  4  9  0 45]]
```

```
Accuracy score: 0.7183908045977011

Recall score: 0.7183908045977011

Precision score: 0.7183908045977011

F1 score: 0.7183908045977011


Multiclass SVM (one vs one):

Confusion matrix:

[[58  0  0  0  0  0  0  0  0  0]

 [ 1 53  4  3  0  0  2  0  0  0]

 [ 0  4 56  0  1  2  1  0  0  1]

 [ 1  0  0 71  0  0  0  1  0  0]

 [ 0  0  3  1 70  1  0  0  0  0]

 [ 0  0  1  0  0 33  1  0  2 26]

 [ 0  1  0  0  0  3 65  2  1  0]

 [ 0  3  0  0  0  2  1 77  0  0]

 [ 0  1  1  0  1  8  1  3 45  1]

 [ 0  3  5  0  0 53  3  1  1 17]]

Accuracy score: 0.7830459770114943

Recall score: 0.7830459770114943

Precision score: 0.7830459770114943

F1 score: 0.7830459770114943


Multiclass SVM (one vs rest):

Confusion matrix:

[[55  1  1  0  0  0  0  1  0  0]

 [ 0 55  2  2  0  0  2  2  0  0]

 [ 0  5 54  1  3  1  0  0  0  1]

 [ 0  2  0 70  0  0  0  1  0  0]

 [ 0  0  1  0 73  1  0  0  0  0]

 [ 0  3  2  0  0 21  1  0  3 33]
 [ 0  1  0  0  0  2 65  2  2  0]
 [ 0  1  1  0  0  2  0 79  0  0]
 [ 0  2  3  0  1  3  1  3 44  4]
 [ 0  5  9  0  1 34  3  2  1 28]]
Accuracy score: 0.7816091954022989
Recall score: 0.7816091954022989
Precision score: 0.7816091954022989
F1 score: 0.7816091954022989
```

The imbalance issue in the Multiclass SVM one vs rest model is solved using the "class_weight='balanced'" parameter in SVC. This ensures that class weights are automatically changed, where the weights are inversely proportional to frequency of the class appearing. Thus the model can adjust weights when there are much more negative classes than positive classes, and when this SVC model is passed into OneVsRestClassifier, class imbalance issue can be solved.

Across all 3 classifiers, assuming we start counting row number from 0, row 5 (and row 9 for the multiclass SVMs) have exceptionally low true values along the diagonal. This suggests that datapoints where the leaf_label is "forest fire" (and sometimes for "heatwave") is exceptionally hard to predict correctly. For the multiclass SVMs, these 2 are oftentimes confused with each other, as shown by the higher false prediction rates of data labeled as "forest fire" but the classifier predicting them to be "heatwave", and vice versa.

I suggest for "forest fire" and "heatwave" labels to be merged together. Class 9 (heatwave) is merged into class 5 (forest fire) so all existing labels for "heatwave" have been changed to "forest fire". The new performance is shown below.

Naïve Bayes classifier:

```
Confusion matrix:
[[ 47   2   4   4   1   0   0   0   0]
 [  0  41  17   0   1   0   2   2   0]
 [  0   2  58   0   0   1   1   3   0]
 [  0   9   1  62   0   0   0   1   0]
 [  0   3   9   1  59   2   0   1   0]
 [  0   5  13   0   0 109   5  12   2]
 [  0   0   2   0   0   4  63   1   2]
 [  0   2   3   0   0   1   1  76   0]
 [  0   1   3   0   0   5   1   6  45]]
```

```
Accuracy score: 0.8045977011494253
```

Multiclass SVM (one vs one):

```
Confusion matrix:
[[ 58   0   0   0   0   0   0   0   0]
 [  1  50   4   3   0   5   0   0   0]
 [  0   3  55   0   1   5   1   0   0]
 [  2   0   0  71   0   0   0   0   0]
 [  0   0   3   1  70   1   0   0   0]
 [  0   4   2   0   0 136   2   0   2]
 [  0   1   0   0   0   5  65   0   1]
 [  0   1   0   0   0   5   1  76   0]
 [  0   1   0   0   0  13   1   1  45]]
Accuracy score: 0.8994252873563219
```

Multiclass SVM (one vs rest):

```
Confusion matrix:
[[ 55   1   1   0   0   0   0   1   0]
 [  0  55   2   2   0   0   2   2   0]
 [  0   5  54   1   3   2   0   0   0]
 [  0   2   0  70   0   0   0   1   0]
 [  0   0   1   0  73   1   0   0   0]
 [  0   6  10   0   1 120   4   2   3]
 [  0   1   0   0   0   4  63   2   2]
 [  0   1   1   0   0   3   0  78   0]
 [  0   2   3   0   1  12   1   2  40]]
Accuracy score: 0.8735632183908046
```

As we can see, accuracy scores for both multiclass SVMs increased.

Since there are now more data points with the label of "forest fire", there is class imbalance where there are more data samples for class 5. The earlier method of mitigating class imbalance (using class_weight='balanced') can similarly be employed to solve this challenge. After using this class imbalance solution, here are the new performance results for the multiclass SVMs.

Multiclass SVM (one vs one):

Confusion matrix:

```
[[ 58   0   0   0   0   0   0   0   0]
 [  1  53   4   3   0   0   2   0   0]
 [  0   5  57   0   1   1   1   0   0]
 [  1   0   0  71   0   0   0   1   0]
 [  0   1   3   1  69   1   0   0   0]
 [  0   5   6   0   0 125   4   2   4]
 [  0   1   0   0   0   3  65   2   1]
 [  0   3   0   0   0   2   1  77   0]
 [  0   2   1   0   1   8   1   3  45]]
```

Accuracy score: 0.8908045977011494

Multiclass SVM (one vs rest):

Confusion matrix:

```
[[ 55   1   1   0   0   0   0   1   0]
 [  0  55   2   2   0   0   2   2   0]
 [  0   5  54   1   3   2   0   0   0]
 [  0   2   0  70   0   0   0   1   0]
 [  0   0   1   0  73   1   0   0   0]
 [  0   6  10   0   1 120   4   2   3]
 [  0   1   0   0   0   4  63   2   2]
 [  0   1   1   0   0   3   0  78   0]
 [  0   2   3   0   1  12   1   2  40]]
```

Accuracy score: 0.8735632183908046

Q10.

a) According to section 3 of the paper, co-occurrence probabilities help to "distinguish relevant words … from irrelevant words", and "discriminate between the two relevant words". This is because co-occurrence probabilities uses conditional probabilities to find out how likely both words appear together, which serves to highlight the relationship between relevant words in the same context, and also devaluing the other irrelevant words from the selected relevant words.

b) It would not return the same vector. While both words are identical, both are verbs of the sentence, and both are of the same verb type (continuous action), GLoVE tries to extract information about how related the words are to other words in the same context. For example, GLoVE might assign stronger co-occurrence probabilities for the words "running" and "park" in the first sentence, but for the second sentence the probabilities will be stronger for the words "running" and "presidency". Since co-occurrence probabilities do not depend on just the word itself, but also the relevant words around it, there will be different embeddings assigned to the word "running" in the 2 sentences.

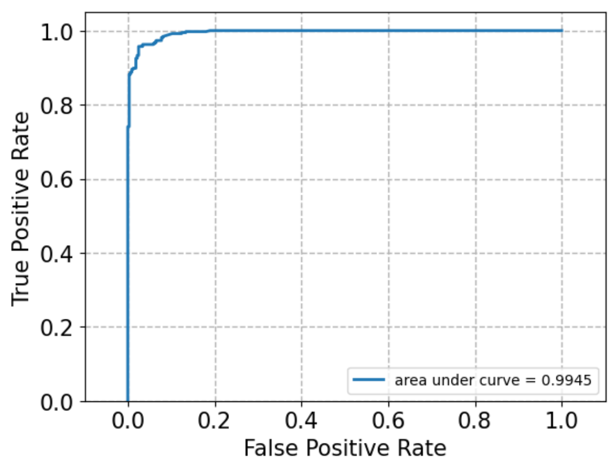
c) "Left" and "right" are extremely relevant to each other as both describe directions, so I expect their GLoVE embeddings to be more similar and thus the magnitude of their differences will be less. Similarly, "wife" and "husband" are also very relevant to each other as they are both related to family terms and marriage, so their embeddings differences should be similar to that between "left" and "right", and the magnitude of the vector difference is similarly small. However, "wife" and "orange" are very far apart from each other in terms of semantics, so their GLoVE embeddings should be vastly different and so the magnitude of their differences is expected to be much larger than the first 2 vector differences.

d) Lemmatization should be better, because compared to stemming, it extracts the lemma from each word more accurately, so the chances of the extracted word form being closer to the actual lemma form of the word is higher, and GLoVE can be performed on the more accurate lemma version of the word to give an accurate embedding.

Q11.

a) After lemmatizing full_text, we made a Word2VecVectorizer that has the same format as the vectorizers from sklearn, but the transform() function goes through each word in a datapoint, checks if it's in the GLoVE embedding dictionary, and adds the corresponding vector to an array if a match is found. After going through all words in the text segment, we find the mean of the vectors in the array. Those vectors are aligned in a way such that each row contains 1 vector of dimension=GLoVE embedding dimension (which is 300), and each row is for each datapoint.

b) Using a custom lemmatizer, LSI with k=25, and Logistic Regression with L1 regularization strength of 0.01 and maximum of 500 iterations, we reached an accuracy of 0.95. Here are the detailed performance:



Confusion matrix:

```
[[312  10]
 [ 16 358]]
```

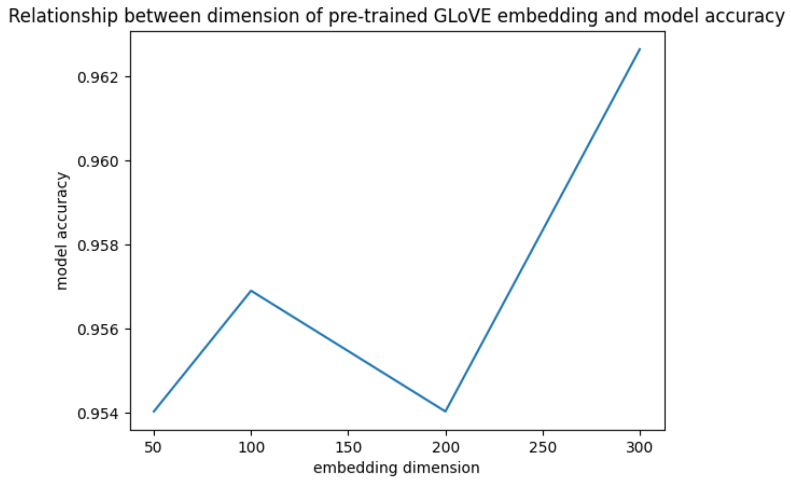
Accuracy score:0.9626436781609196

Recall score: 0.9572192513368984

Precision score: 0.9728260869565217

F1 score: 0.9649595687331537

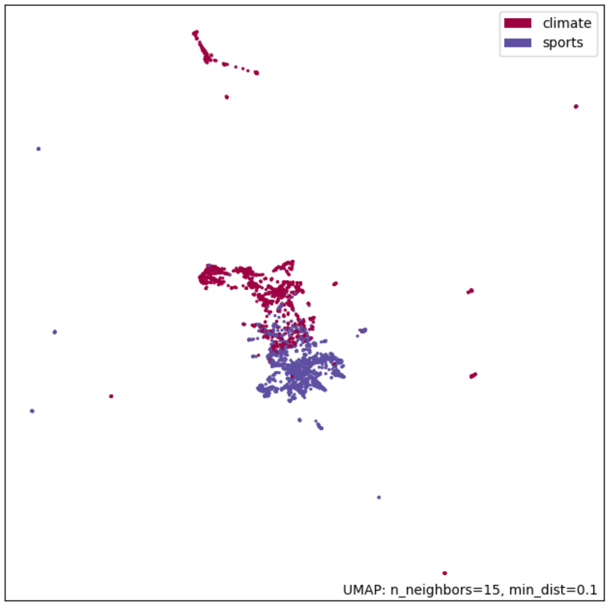
Q12.

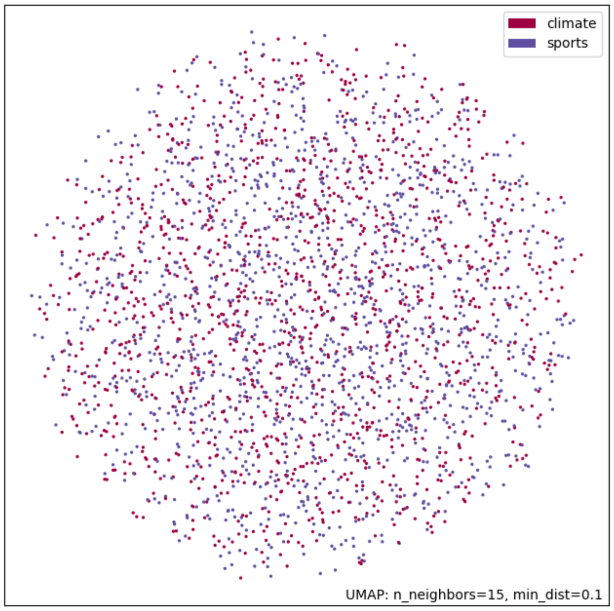


The observed trend is the accuracy increases as dimension increases from 50 to 100, then the accuracy dips when dimension increases to 200, then the accuracy increases again as dimension increases to 300.

This is not expected, as I expect the accuracy to increase as dimension increase, since more information can be contained with a higher embedding dimension that can lead to more features learnt in the model. I also anticipated the accuracy to first increase, then drop as dimension increases, because it's possible that too much information in the embedding can cause overfitting and decrease testing accuracy of models. However, neither of these 2 scenarios occurred. I feel it might simply be a case of small uncertainties, because the dip in model accuracy between dimension=100 and dimension=200 is very small, so it might simply be due to other factors.

Q13.





The left visualization is from the GLoVE embeddings, while the right visualization is from a normalized random-generated matrix of the same shape. We can clearly see that there are clusters being formed for the left visualization, where the embeddings representing 'climate' are grouped together above the cluster of embeddings representing 'sports'. Even though the clusters are not well-separated, we can see some form of clustering being formed. The right visualization shows no cluster whatsoever and is purely random, which is expected of a randomly-generated matrix.