ECE 219 Project 1

Tanner Dulay, Loïc Maxwell, Zoë Tucker, Alex Wasdahl *

January 20, 2021

1 Question 1

To begin, we need to determine whether the data in our dataset is evenly distributed. To check this, we plot a bar chart showing the number of documents with each available label in the "test" subset.

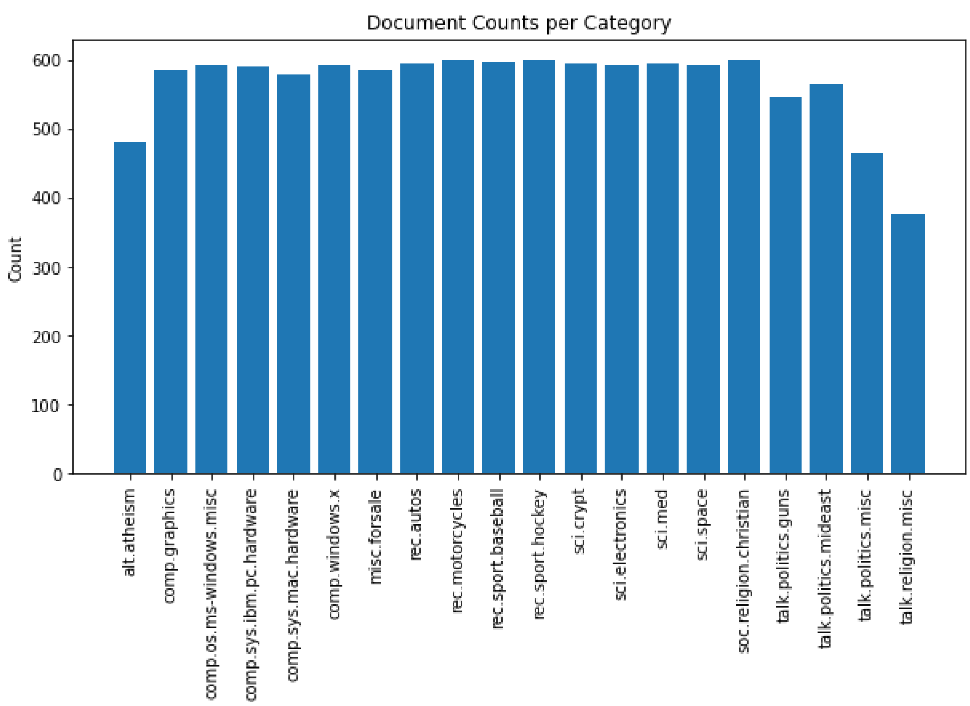


Figure 1: Histogram of the data

*Alex Wasdahl was a member of our team until 1/20/21, and contributed much of the text of Section 8. Alex's ID is not included in the assignment submission metadata.

As shown in Figure 1, the data is balanced, especially considering only the labels identified as "Computer Technology" or "Recreational Activity" as defined in the project statement.

2 Question 2

Next, we need to make a representation of our dataset to use in our classification algorithms using `CountVectorizer`. Excluding "`english`" stopwords, numerals, and punctuation, lemmatizing with `nltk.wordnet.WordNetLemmatizer` and `pos_tag`, and using `min_df` = 3, we create TF-IDF matrices for the test and train data subsets.

The resulting training data matrix is 4732×16466, and the test data matrix is 3150×16466.

3 Question 3

Before we apply classification algorithms, we reduce the dimensionality of the data to map each document to a 50-dimensional vector. We attempted this with two methods: LSI and NMF.

For LSI, the total (Frobenius) error is 4099.6. For NMF, it is 4141.4. As the error for the LSI-reduced matrices is lower, LSI is the better method for our case and we use the LSI matrices in subsequent questions. The reduced error for LSI may be due to the fact that NMF is forced to use non-negative matrices in its decomposition, limiting the range of potential solutions.

4 Question 4

In this section, we use linear Support Vector Machines (SVMs) as our first classification algorithm. We begin by training a "hard margin" SVM, with tradeoff parameter $\gamma = 1000$ representing a high penalty for individual misclassifications, and a "soft margin" SVM with $\gamma = 0.0001$ representing a higher prioritization of separation of most data points.

The below table summarizes the performance statistics for each of the SVMs, and the confusion matrices and ROC curves are given in Figures 2, 3, 4, and 5.

SVM	Accuracy	Recall	Precision	F-1 Score
Soft Margin	0.6711	0.6679	0.8027	0.6286
Hard Margin	0.9721	0.9720	0.9722	0.9721

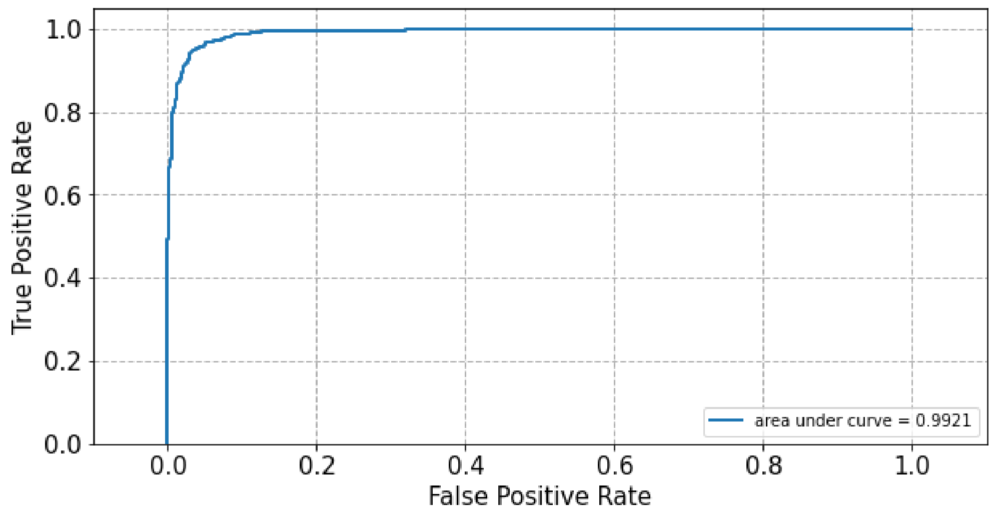


Figure 2: ROC for Soft Margin SVM

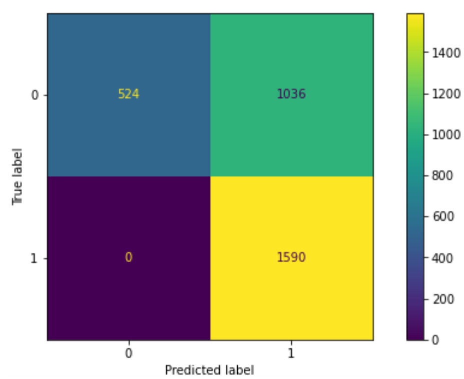


Figure 3: Confusion Matrix for Soft Margin SVM

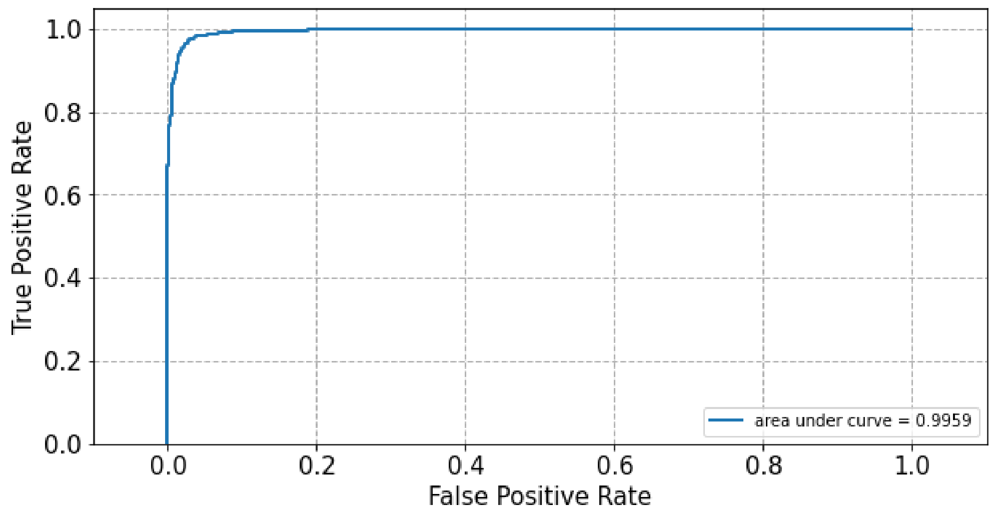


Figure 4: ROC for Hard Margin SVM

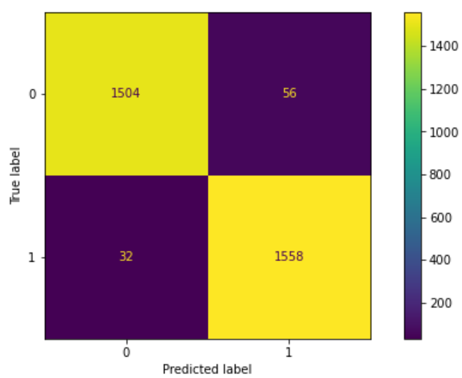


Figure 5: Confusion Matrix for Hard Margin SVM

From these statistics, especially the vastly improved accuracy of .97 over .67, we conclude that the hard margin SVM performs much better.

Looking more closely at the soft margin results, we see that the ROC curve is not very different than the hard margin ROC, despite the very different accuracy and other metrics. This indicates that the model has chosen a poor threshold for separating the data.

Next, we want to find the best possible SVM. To do this, we use cross-validation in testing to find the optimal value for γ. In particular, we use 5-fold cross-validation and look for a $\gamma \in \{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$.

Testing each parameter, we find that the best value is $\gamma = 1$, which gave an average accuracy of 0.9727 during our cross-validation trial. We then use this value to train a new SVM with the full training dataset, which gives the scores in the below table and in Figures 6 and 7.

SVM	Accuracy	Recall	Precision	F-1 Score
$\gamma = 1$	0.9727	0.9726	0.9730	0.9727

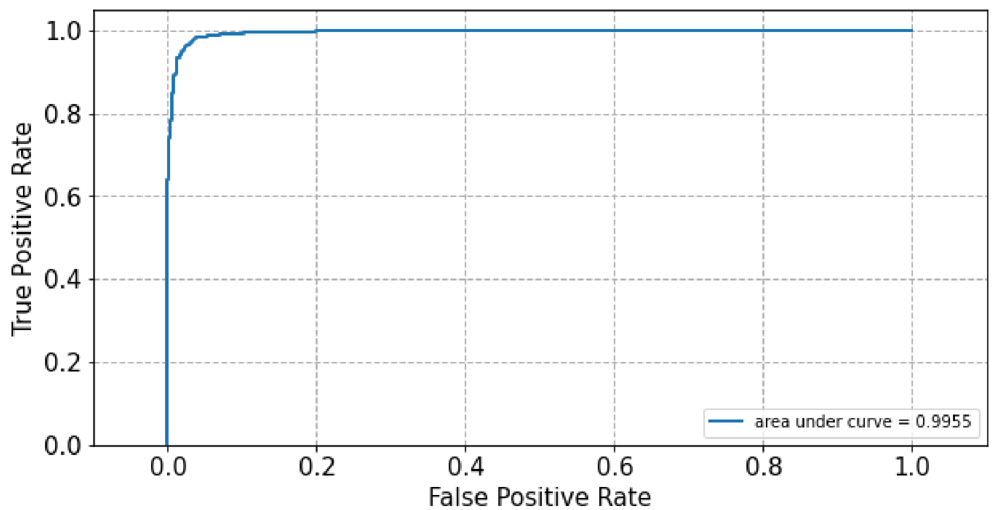


Figure 6: ROC for Optimal SVM

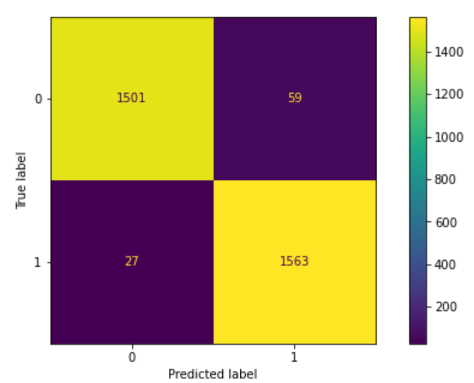


Figure 7: Confusion Matrix for Optimal SVM

5 Question 5

Next, we analyze logistic classifiers for this task using the same methods as in section 4. We begin with a non-regularized classifier. Since we are using `sklearn`'s implementation, we set C to a very large number to approximate an unregularized classifier. The statistics for this classifier are found in the table below, and the ROC and confusion matrix are shown in Figures 8 and 9.

Log. Classifier	Accuracy	Recall	Precision	F-1 Score
Unregularized	0.9717	0.9635	0.9647	0.9717

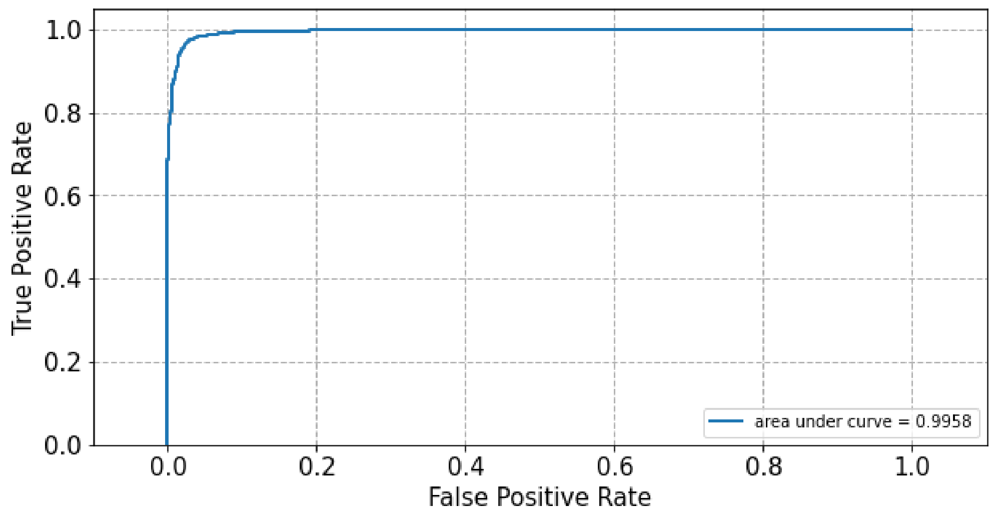


Figure 8: ROC for Unregularized Logistic Regression Classifier

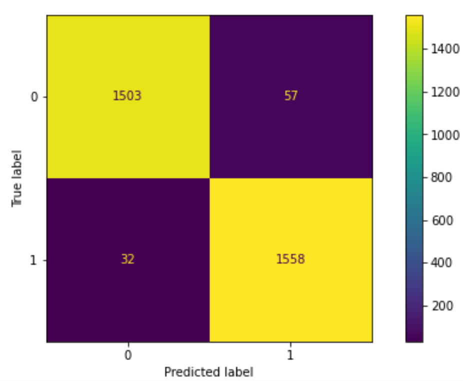


Figure 9: Confusion Matrix for Unregularized Logistic Regression Classifier

As before, we use five-fold cross-validation to find the optimal regularization parameter C. However, we now have two options for regularization type: L1 and L2. Thus, we must run the cross-validation twice, once for each type.

We find that for both regularization types, the optimal value of C is 10. The performance metrics for these optimal classifiers are given in the tabe below, and the ROC and confusion matrix for each are shown in Figures 10 - 13.

Log. Classifier	Accuracy	Recall	Precision	F-1 Score
L1, $C = 10$	0.9717	0.9717	0.9721	0.9717
L2, $C = 10$	0.9717	0.9716	0.9720	0.9717

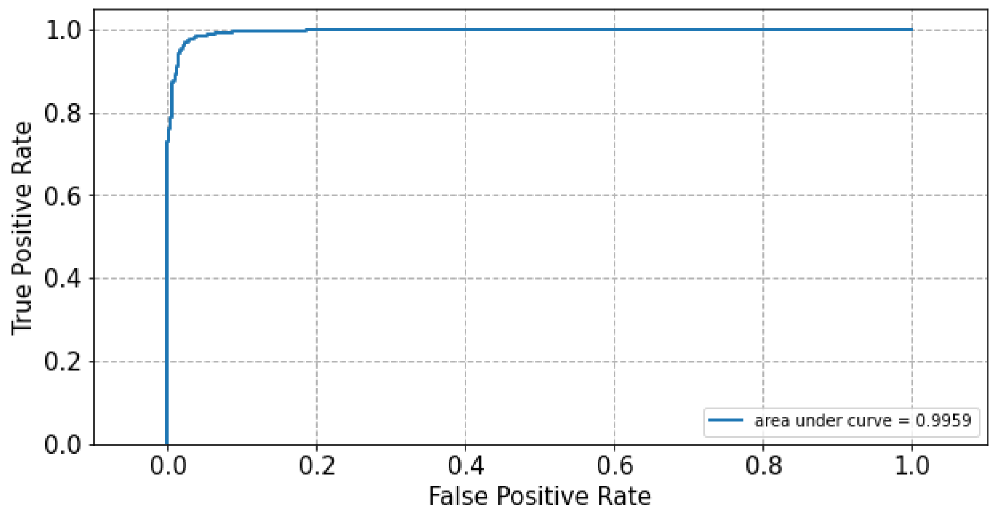


Figure 10: ROC for L1 Logistic Regression Classifier, $C = 10$

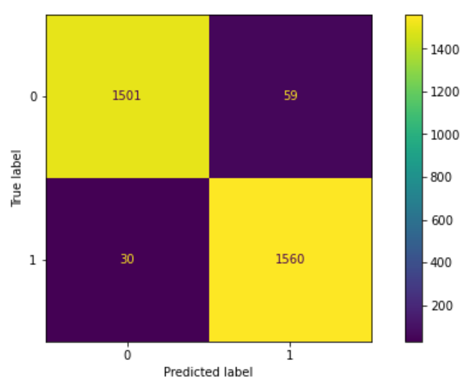


Figure 11: Confusion Matrix for L1 Logistic Regression Classifier, $C = 10$

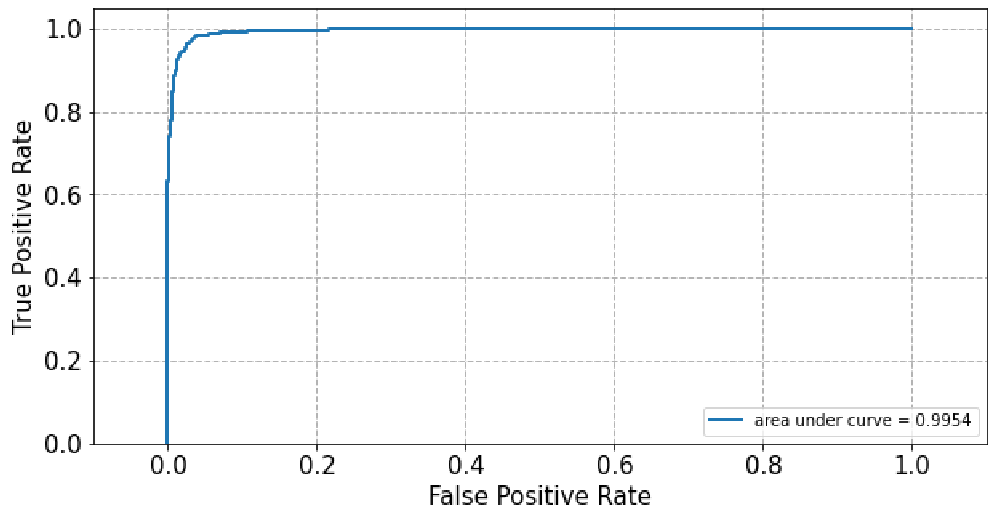


Figure 12: ROC for L2 Logistic Regression Classifier, $C = 100$

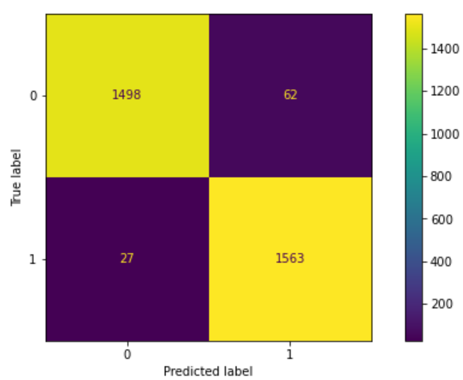


Figure 13: Confusion Matrix for L2 Logistic Regression Classifier, $C = 100$

As we can see, the logistic regression classifier with L1 regularization was slightly better than the others when comparing recall and precision across the models. The unregularized classifier had the worst performance.

Looking specifically at the learned coefficients, we see that they are fairly large for the unregularized classifier, with an average absolute value of around 13. This may suggest that the model is overfitted to the training data.

While L2 regularization is computationally easier and produces good results often, L1 regularization is useful when the feature space is not too large.

In comparison to the previous section, we see that the logistic regression classifiers do not perform as well as the optimal SVM classifier. This is likely due to the ways that the two models attempt to create a decision boundary. SVMs learn a feature vector by looking at all of the training data points and minimizing a loss function with regularization. In contrast, logistic regression takes a probabilistic approach by iteratively attempting to find the optimal decision boundary. For structured data like our text documents in this problem, SVMs have better performance.

6 Question 6

Finally, we train a Naïve Bayes classifier on our dataset and find the same metrics, demonstrating worse performance than either of the two previous models. The below table summarizes the information, and the ROC curve and confusion matrix are shown in Figures 14 and 15.

Classifier	Accuracy	Recall	Precision	F-1 Score
Naïve Bayes	0.89	0.89	0.90	0.89

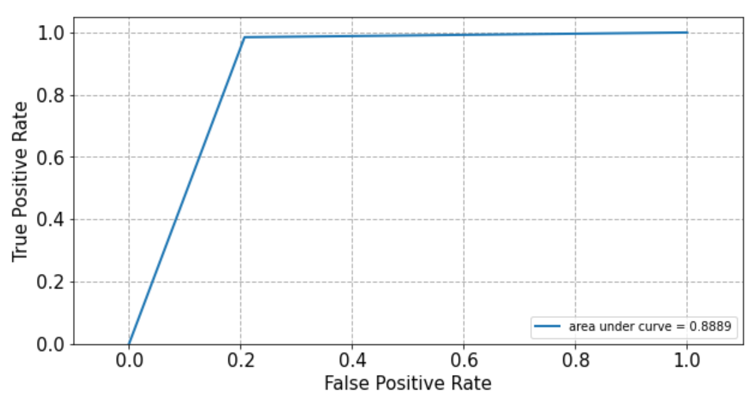


Figure 14: ROC for Naïve Bayes Classifier

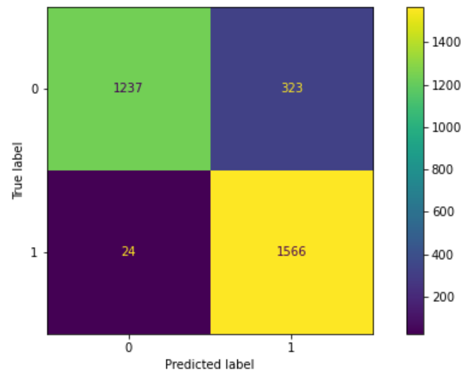


Figure 15: Confusion Matrix for Naïve Bayes Classifier

7 Question 7

Now that we have encountered each of these classifiers individually, we perform a grid search to find the optimal combination of parameters. We compare differences in data loading (removing or leaving "headers" and "footers"), feature extraction (`min_df` = 3 or 5, using lemmatization or not), dimensionality reduction (NMF or LSI), and classifier type (SVM, Logistic Regression, or Naïve Bayes with previously found optimal parameters). Two sets of training/test data were used: one with headers/footers and another without. `GridSearchCV` was run twice to tune the (hyper)parameters for each set of data, and the grid search that had the best performance on the test set was selected. The results are shown below:

Dataset Type	Lemmatization	min_df Value	LSI/NMF	Classifier	Test Accuracy
With Headers/Footers	Yes	3	LSI	Logistic Regression w/ L1 reg and C=10	0.9721
Without Headers/Footers	Yes	3	LSI	Logistic Regression w/ L1 reg and C=10	0.9670

As can be seen in the table, the best model parameters were actually the same for both data types. However, because of its better performance on the test set, the best combination includes data with headers and footers. This is why it's highlighted in the table.

8 Question 8

Next, we consider GLoVE embeddings of text data. GLoVe embeddings are trained on the ratio of co-occurrence probabilities rather than the probabilities themselves because the ratio of co-occurrence probabilities produces more meaning. The probabilities themselves can be high or low depending on the contextual relationship of the words in question, but the ratios produce numbers that cancel out much of the "noise" from probe words that either relate to both words of interest or neither. Using the example from the paper, the co-occurrence probabilities for the words of interest $i =$ ice and $j =$ steam differ based on the probe word k used. In the case of a word like "solid" that correlates strongly with ice but poorly with steam, the ratio provides a number much greater than one that gives meaning to the relationship between the two probabilities. In contrast, for a probe word like "water," the probabilities themselves are high for both correlations water—ice and water—steam, but the ratio is close to 1, which provides more contextual meaning about the relative relationship of the probe word water to both ice and steam. Similarly, a probe word like "fashion" which has low probabilities for both correlations fashion—ice and fashion—steam will also have a ratio close to 1. The probabilities are different but the ratios add more meaning regarding the relevance or irrelevance of the probe words.

For the sentences "James is **running** in the park" and "James is **running** for the presidency", GLoVE will **not** return the same vector for the word "running". This is because GloVe incorporates global statistics (word co-occurrence) to obtain word vectors rather than just relying on local statistics (local context information of words).

For all of the expressions

$$||\texttt{GLoVE["queen"] - GLoVE["king"] - GLoVE["wife"] - GLoVE["husband"]}||_2,$$

$$||\texttt{GLoVE["queen"] - GLoVE["king"]}||_2,$$

$$||\texttt{GLoVE["wife"] - GLoVE["husband"]}||_2,$$

we expect a value of zero.

Given a word, lemmatization is preferable to stemming before mapping to its GLoVE embedding. Stemming operates on a single word without knowledge of the context, and therefore cannot discriminate between words which have different meanings depending on part of speech. Lemmatization considers the roles of the words in the sentence.

9 Question 9

Our feature engineering process that leverages GLoVE embedding is as follows. First, for each document, we grab all characters that follow the strings: "Subject:" and "Keywords:" up until the next newline character and store them in a new string. We also remove the strings "Re:" and "Fwd:" if they can be found in this string, since they are common terms (found in the subject line) that don't give us any information about the document's class. We then tokenize the string and lemmatize each of the words. For each of the words, if an embedding exists in the GLoVE embeddings dictionary, we grab the corresponding vector and normalize it. We then average all of these vectors to form a single vector that represents the entire document. For the documents where none of the lemmatized words have corresponding embeddings, we decided to not include them in the dataset to be fed into the classifier. There were about 50 cases of this per dataset, which is still small compared to the number of total documents in each dataset.

For our classifier, we picked the same best parameters from Q7 (Linear Regression with L1 Regularization, C=10) and got a test accuracy of 0.9160, which is a good result. This isn't as good as our previous pipeline, but with more hyperparameter tuning we can expect some better results.

10 Question 10

Next, we plot the relationship between the dimension of GLoVE embeddings and the accuracy of our model. The results are shown in Figure 16.

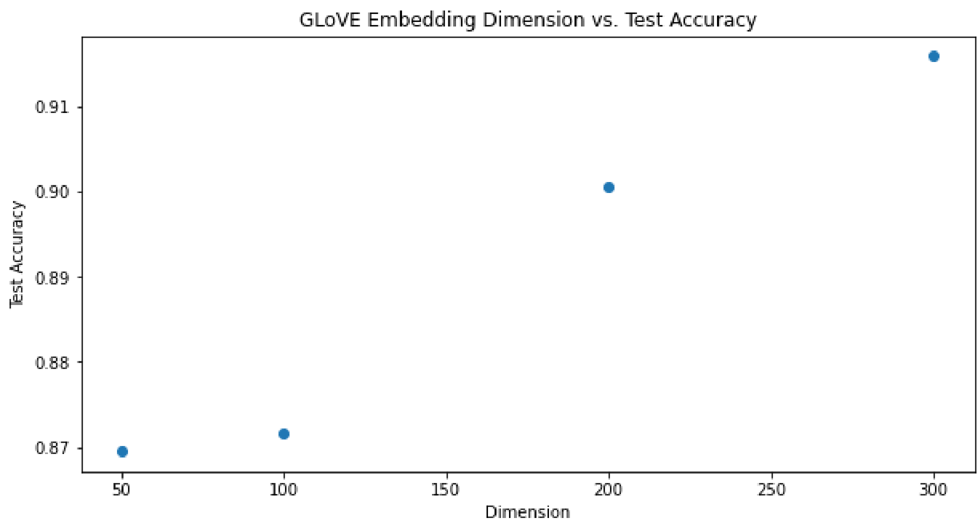


Figure 16: Dimensionality of GLoVE embeddings vs. accuracy of classifiers

Overall, the higher dimensional embeddings gave better results. This is as expected, as it is in line with the results in the original paper. The original authors reported diminishing returns after the dimension exceeded 200, which can be seen in our plot. By having a higher dimensional embedding, we are able to capture more defined literal/contextual information about a word, which allows us to better differentiate between text data belonging to one of two classes. However, if the dimension is too large, then we're "stretching" a word too thin; each feature in the vector becomes less representative of the word, making it more difficult to train (and easier to overfit) and resulting in a lower performance overall. In our case, we got better results with 300 dimensions than with 200 dimensions, but if we keep increasing this number than we would expect our test accuracy to decrease.

11 Question 11

Using UMAP, we project our GLoVE embeddings for the test dataset into the 2D plane, as shown in Figure 17. For comparison, we also generated a random set of normalized 300-dimensional vectors and projected them using the same UMAP methodology (Figure 18).

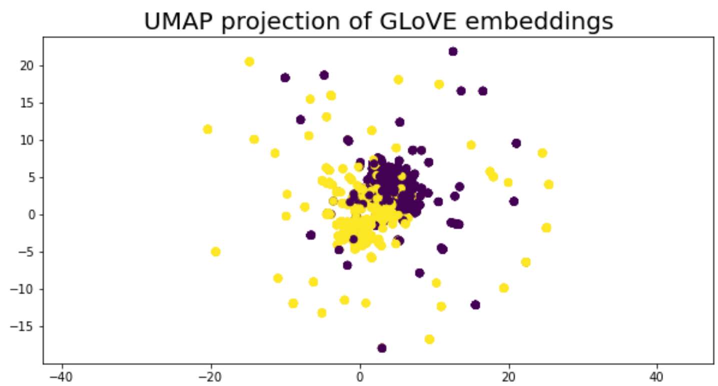


Figure 17: GLoVE Embeddings: Purple is "Computer Technology", Yellow is "Recreational Activity"

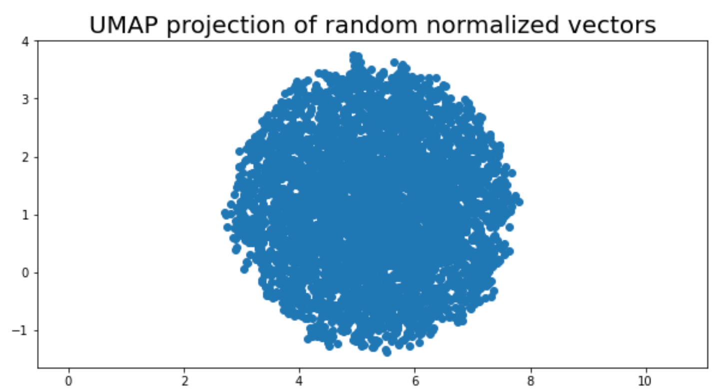


Figure 18: Random projected vectors

In both plots, we see that the points are largely clustered around the origin. However, the clustering on the random vectors is much more uniform, and there are many more "outliers" (points further from the origin - note the different axis scales between the two figures) in the GLoVE plot. In addition, we can see that the data is roughly partitioned, with the purple “Computer Technology” points falling roughly to the upper-right of the center, and the yellow “Recreational Activity” points to the lower-left. This indicates that meaningful information is contained in the GLoVE embeddings, while the random vectors contain no information.

12 Question 12

Finally, we perform multiclass classification on our data. We test Naïve Bayes, One vs. One SVM, and One vs. Rest SVM classifiers to classify data with labels `comp.sys.ibm.pc.hardware`,`comp.sys.mac.hardware`,`misc.forsale`, and `soc.religion.christian`. The below table summarizes the performance of the classifiers, and Figure 19 shows the confusion matrix for each classifier. Comparing accuracy scores for each method, we see that the One vs. Rest SVM has the best performance, and the Naïve Bayes classifier has the worst.

Classifier	Accuracy	Recall	Precision	F-1 Score
Naïve Bayes	0.69	0.69	0.70	0.68
One vs. One SVM	0.8652	0.8652	0.8663	0.8656
One vs. Rest SVM	0.8773	0.8773	0.8775	0.8773

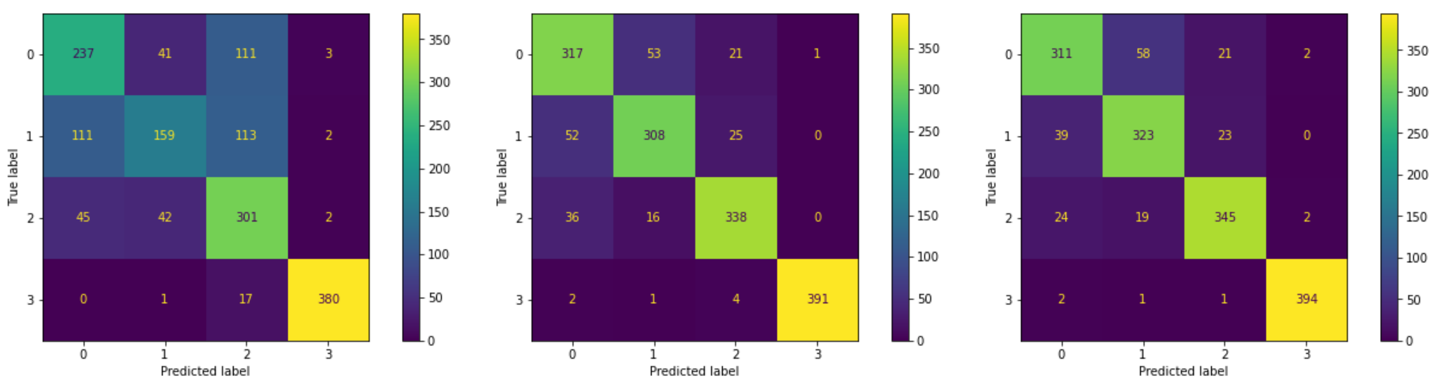


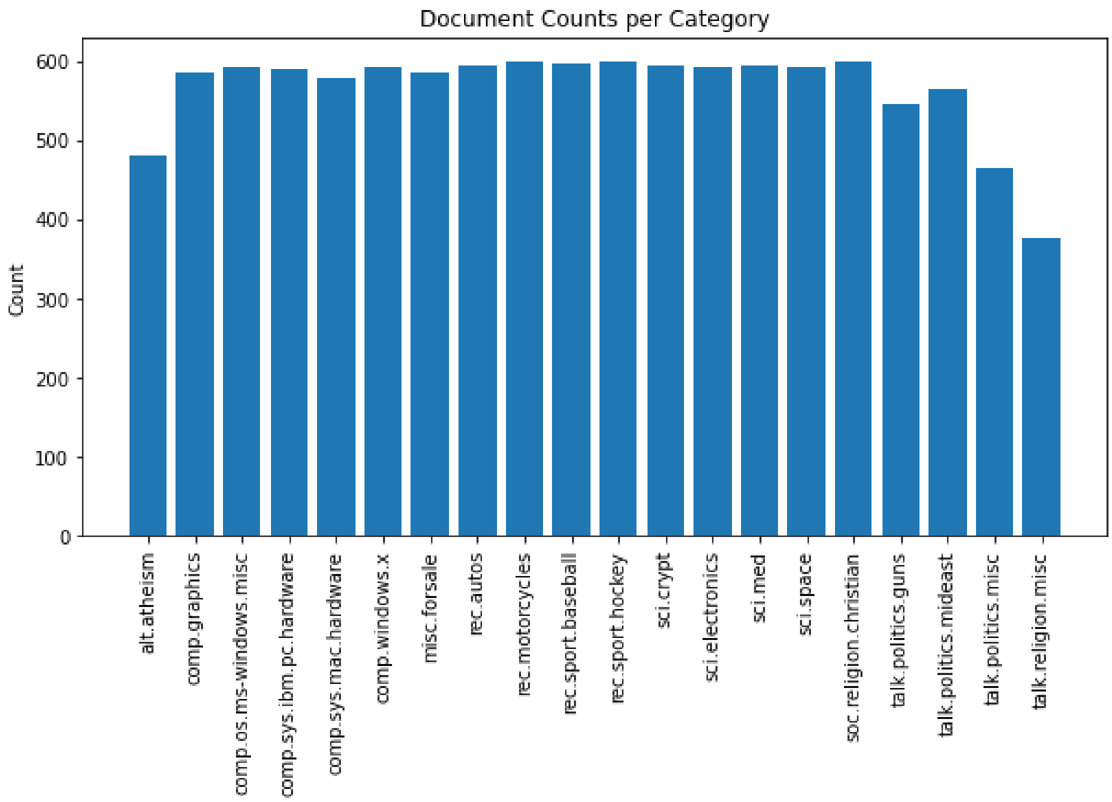
Figure 19: Confusion matrices for multiclass classifiers: Naïve Bayes (Left), One Vs. One SVM (Center), and One Vs. Rest SVM (Right)

As with our binary-classed data, the Naïve Bayes classifier was the least accurate for multiclass data. However, we found all of our classifiers to be less accurate overall when applied to multiclass data. This is likely because our classifiers are now over-fitting the training data. When we divide our dataset to more classes, the sample size for each class is reduced, leading the classifier to do a poorer job of generalizing so that it can correctly classify novel data.

Project_1

January 21, 2021

[1]:
```
#Q1
import numpy as np
np.random.seed(42)
import random
random.seed(42)

from matplotlib import pyplot as plt
from sklearn.datasets import fetch_20newsgroups

plt.rcParams['figure.figsize'] = [10,5]

twenty_plot = fetch_20newsgroups(subset='train')
labels = twenty_plot.target_names

y = list(range(20))
for i in range(20):
    y[i] = np.count_nonzero(twenty_plot.target == i)

plt.bar(labels,y)
plt.xticks(rotation=90)
plt.ylabel("Count")
plt.title("Document Counts per Category")
plt.show()
```



[3]:
```
#Q2
#Questions for TA:
#random_state for dataset fetch? None or 42? 42 used in discussion but project␣
 ↪doc says None

import nltk
from nltk import pos_tag
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from string import punctuation
from nltk.corpus import stopwords

categories = ['comp.graphics', 'comp.os.ms-windows.misc',
'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
'rec.autos', 'rec.motorcycles',
'rec.sport.baseball', 'rec.sport.hockey']

lemur = nltk.wordnet.WordNetLemmatizer()

#Vectorizer
analyzer = CountVectorizer().build_analyzer()
```

```
tfidf_transformer = TfidfTransformer()

#Download dataset
train_dataset = fetch_20newsgroups(subset = 'train', categories = categories,␣
↪shuffle = True, random_state = 42) #should 42 be None?
test_dataset = fetch_20newsgroups(subset = 'test', categories = categories,␣
↪shuffle = True, random_state = 42)

combined_stopwords = set.union(set(stopwords.words('english')),set(punctuation))

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

def lemmatize_sent(text):
    return [lemur.lemmatize(word.lower(), pos=penn2morphy(tag)) for word, tag in␣
↪pos_tag(text)]

#Updated analyzer to avoid counting digits and punctuation
def stem_rmv_punc(doc):
    return (word for word in lemmatize_sent(analyzer(doc)) if word not in␣
↪combined_stopwords and not word.isdigit())

count_vect = CountVectorizer(min_df=3,analyzer=stem_rmv_punc,␣
↪stop_words='english')

corpus = [
    'This is the first document.',
    'This is the second second document.',
    'And the third one.',
    'Is this the first document?',
]

#X_train_counts = count_vect.fit_transform(corpus)

X_train_counts = count_vect.fit_transform(train_dataset.data)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

X_test_counts = count_vect.transform(test_dataset.data)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)

print("X_train shape: ", X_train_tfidf.shape)
print("X_test shape: ", X_test_tfidf.shape)
```

```
X_train shape:  (4732, 16466)
X_test shape:  (3150, 16466)
```

[7]:
```
#Q3
#Questions for TA:
#random_state for nmf and trunc svd? 0 or 42?

from sklearn.decomposition import NMF, TruncatedSVD
from sklearn.utils.extmath import randomized_svd

trunc_svd = TruncatedSVD(n_components=50, random_state=42)
X_train_lsi = trunc_svd.fit_transform(X_train_tfidf)
X_test_lsi = trunc_svd.transform(X_test_tfidf)

#Get trunc svd matrices
#https://stackoverflow.com/questions/31523575/
↪get-u-sigma-v-matrix-from-truncated-svd-in-scikit-learn
U, Sigma, VT = randomized_svd(X_train_tfidf, n_components=50, random_state=42)
Sigma = np.diag(Sigma) #Make sigma values a diag matrix rather than nx1

nmf_model = NMF(n_components=50, init='random', random_state=42, max_iter=1000)
X_train_nmf = nmf_model.fit_transform(X_train_tfidf)
X_test_nmf = nmf_model.transform(X_test_tfidf)
H = nmf_model.components_
error_lsi = np.sum(np.array(X_train_tfidf - U.dot(Sigma.dot(VT)))**2)
error_nmf = np.sum(np.array(X_train_tfidf - X_train_nmf.dot(H))**2)

print('LSI Train error: ', error_lsi)
print('NMF Train error: ', error_nmf)

print('LSI Train shape: ', X_train_lsi.shape)
print('LSI Test shape: ', X_test_lsi.shape)
print('NMF Train shape: ', X_train_nmf.shape)
print('NMF Test shape: ', X_test_nmf.shape)
```

```
LSI Train error:  4099.627761986766
NMF Train error:  4141.402934293383
LSI Train shape:  (4732, 50)
LSI Test shape:  (3150, 50)
NMF Train shape:  (4732, 50)
NMF Test shape:  (3150, 50)
```

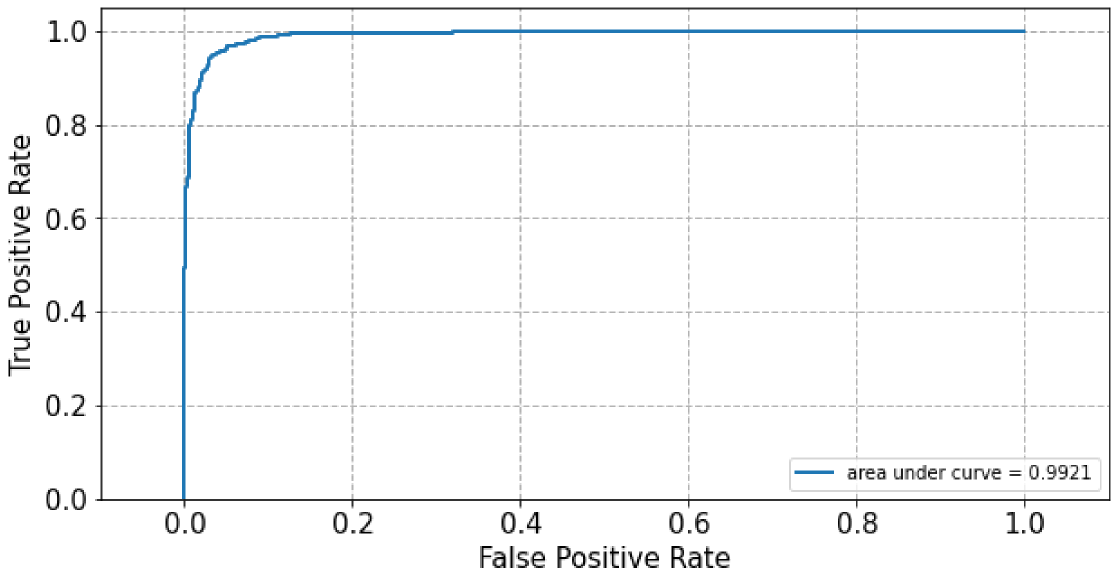
[8]:
```
#Q4
from sklearn.svm import LinearSVC
from sklearn import metrics

#Start by changing to binary classification
```

```
train_targets_bin = train_dataset.target.copy()
for i in range(len(train_targets_bin)):
    if train_dataset.target[i] in [0,1,2,3]:
        train_targets_bin[i] = 0
    else:
        train_targets_bin[i] = 1

test_targets_bin = test_dataset.target.copy()
for i in range(len(test_targets_bin)):
    if test_dataset.target[i] in [0,1,2,3]:
        test_targets_bin[i] = 0
    else:
        test_targets_bin[i] = 1

#Train hard and soft margin SVCs - using LSI data per instructions - then get␣
 ↪stats
soft_margin_SVC = LinearSVC(C=0.0001,max_iter=90000).fit(X_train_lsi,␣
 ↪train_targets_bin)
soft_margin_prediction_func = soft_margin_SVC.decision_function(X_test_lsi)
soft_margin_prediction = soft_margin_SVC.predict(X_test_lsi)

hard_margin_SVC = LinearSVC(C=1000,max_iter=90000).fit(X_train_lsi,␣
 ↪train_targets_bin)
hard_margin_prediction_func = hard_margin_SVC.decision_function(X_test_lsi)
hard_margin_prediction = hard_margin_SVC.predict(X_test_lsi)

fpr_soft, tpr_soft, thresholds_soft = metrics.roc_curve(test_targets_bin,␣
 ↪soft_margin_prediction_func, pos_label=1)
fpr_hard, tpr_hard, thresholds_hard = metrics.roc_curve(test_targets_bin,␣
 ↪hard_margin_prediction_func, pos_label=1)

#I stole this helper function from the discussion notebook!
def plot_roc(fpr, tpr):
    fig, ax = plt.subplots()

    roc_auc = metrics.auc(fpr,tpr)

    ax.plot(fpr, tpr, lw=2, label= 'area under curve = %0.4f' % roc_auc)

    ax.grid(color='0.7', linestyle='--', linewidth=1)

    ax.set_xlim([-0.1, 1.1])
    ax.set_ylim([0.0, 1.05])
    ax.set_xlabel('False Positive Rate',fontsize=15)
    ax.set_ylabel('True Positive Rate',fontsize=15)

    ax.legend(loc="lower right")
```

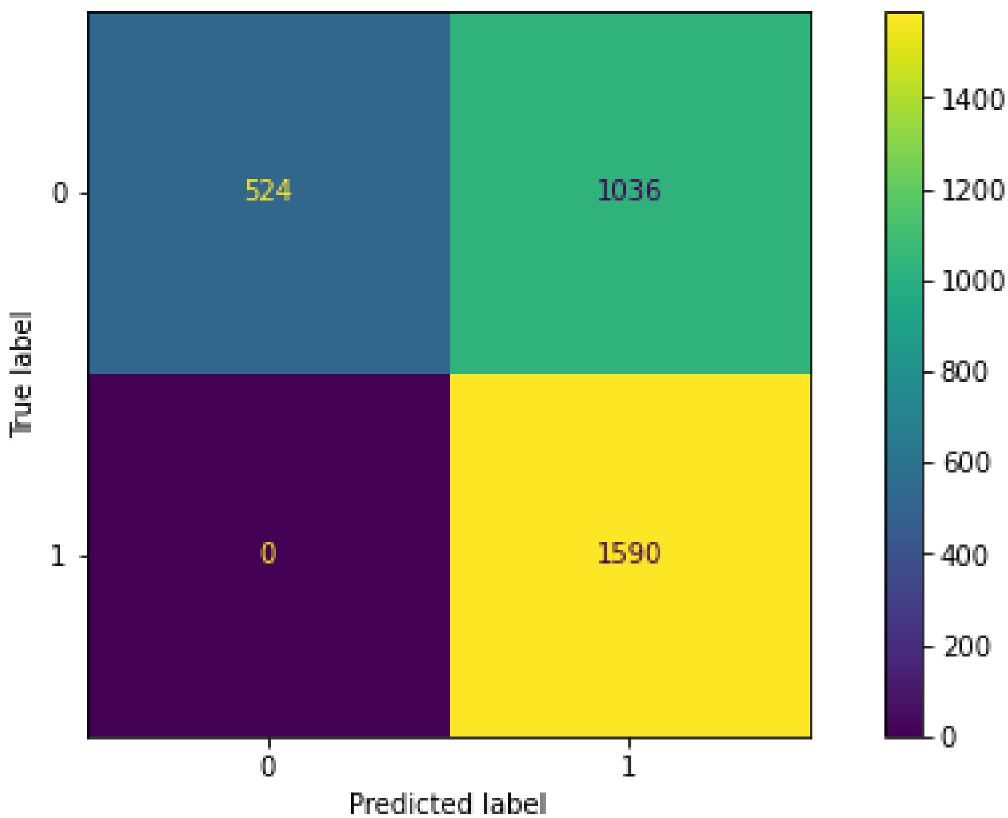
```
    for label in ax.get_xticklabels()+ax.get_yticklabels():
        label.set_fontsize(15)

#Show the stats for our model (part 1 of question)
plot_roc(fpr_soft,tpr_soft)
metrics.plot_confusion_matrix(soft_margin_SVC, X_test_lsi, test_targets_bin)
print("Soft margin prediction stats:")
print(metrics.
 ↪classification_report(test_targets_bin,soft_margin_prediction,digits=4))

plot_roc(fpr_hard,tpr_hard)
metrics.plot_confusion_matrix(hard_margin_SVC, X_test_lsi, test_targets_bin)
print("Hard margin prediction stats:")
print(metrics.
 ↪classification_report(test_targets_bin,hard_margin_prediction,digits=4))

#hard margin is much better than soft!

#Part 2!
def find_best_gamma(ks):
    best_k = -100
    best_acc = 0
    for i in ks:
        k_SVC = LinearSVC(C=10**i,max_iter=90000).fit(X_train_lsi,␣
 ↪train_targets_bin)
        k_prediction = k_SVC.predict(X_test_lsi)
        scores = metrics.confusion_matrix(test_targets_bin,k_prediction)
        acc = (scores[0][0]+ scores[1][1])/(scores[0][0]+ scores[0][1] +␣
 ↪scores[1][0]+ scores[1][1])
        if acc > best_acc:
            best_k = i
            best_acc = acc
    return best_k


k = find_best_gamma([-3,-2,-1,0,1,2,3])
print("my best gamma is 10^" + str(k))
#recalculate model w/best k and report stats
optimal_SVC = LinearSVC(C=10**k,max_iter=50000).fit(X_train_lsi,␣
 ↪train_targets_bin)
optimal_prediction_func = optimal_SVC.decision_function(X_test_lsi)
optimal_prediction = optimal_SVC.predict(X_test_lsi)
fpr_opt, tpr_opt, thresholds_opt = metrics.roc_curve(test_targets_bin,␣
 ↪optimal_prediction_func, pos_label=1)
plot_roc(fpr_opt,tpr_opt)
```

```
metrics.plot_confusion_matrix(optimal_SVC, X_test_lsi, test_targets_bin)
print("Optimal prediction stats:")
print(metrics.
 ↪classification_report(test_targets_bin,optimal_prediction,digits=4))
```

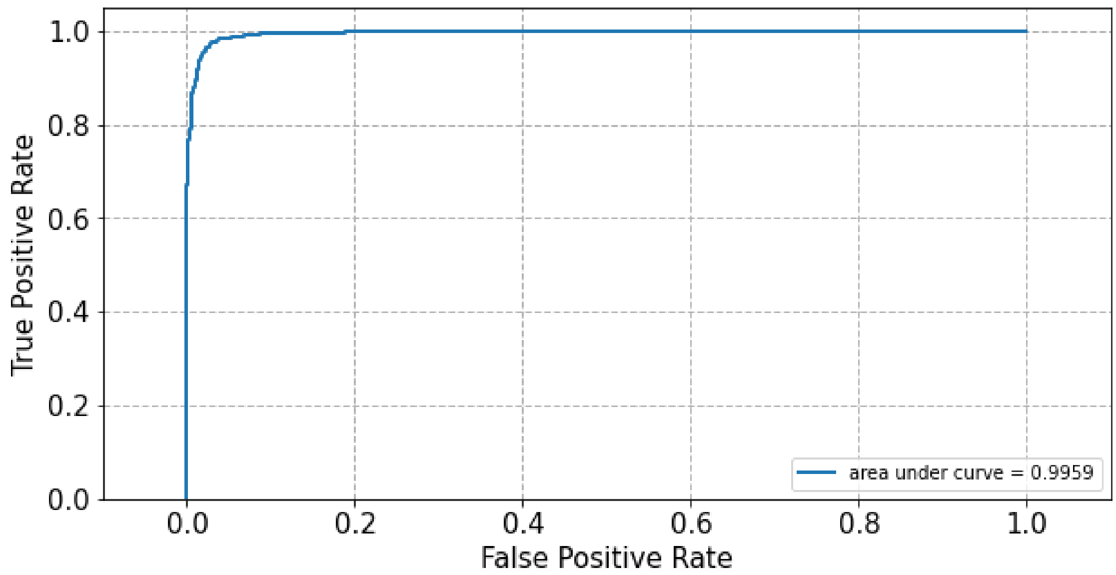
```
Soft margin prediction stats:
              precision    recall  f1-score   support

           0     1.0000    0.3359    0.5029      1560
           1     0.6055    1.0000    0.7543      1590

    accuracy                         0.6711      3150
   macro avg     0.8027    0.6679    0.6286      3150
weighted avg     0.8009    0.6711    0.6298      3150

Hard margin prediction stats:
              precision    recall  f1-score   support

           0     0.9792    0.9641    0.9716      1560
           1     0.9653    0.9799    0.9725      1590

    accuracy                         0.9721      3150
   macro avg     0.9722    0.9720    0.9721      3150
weighted avg     0.9722    0.9721    0.9721      3150

my best gamma is 10^0
Optimal prediction stats:
              precision    recall  f1-score   support

           0     0.9823    0.9622    0.9722      1560
           1     0.9636    0.9830    0.9732      1590

    accuracy                         0.9727      3150
   macro avg     0.9730    0.9726    0.9727      3150
weighted avg     0.9729    0.9727    0.9727      3150
```



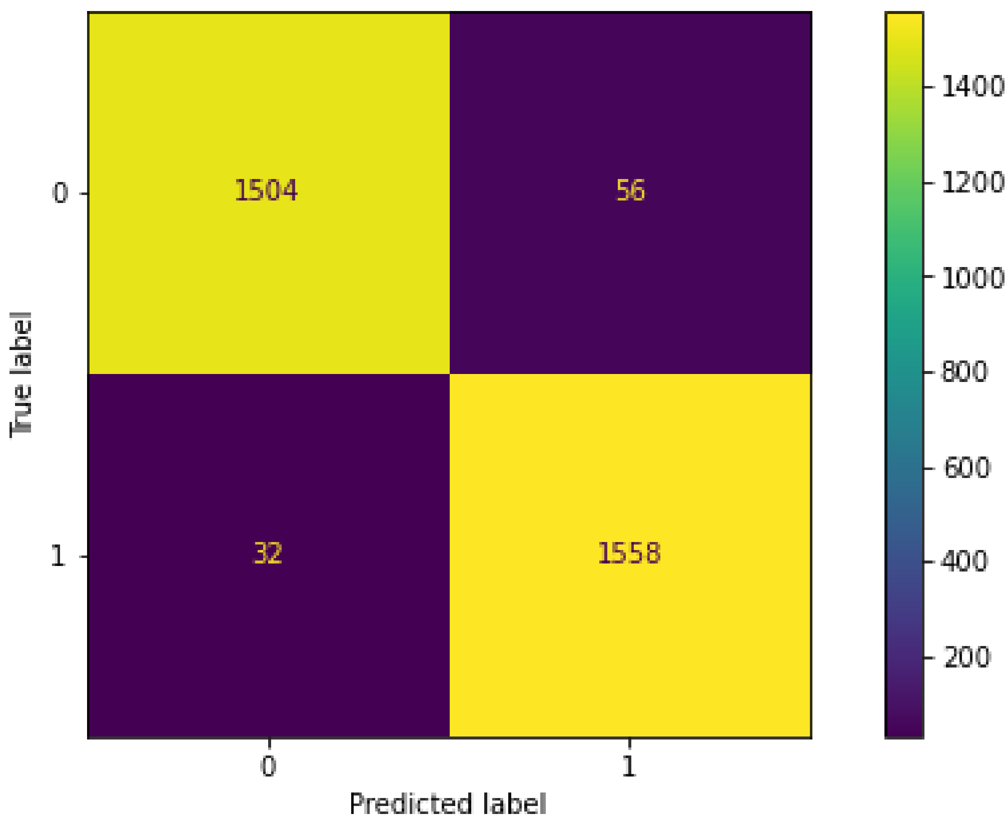
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
area under curve = 0.9921



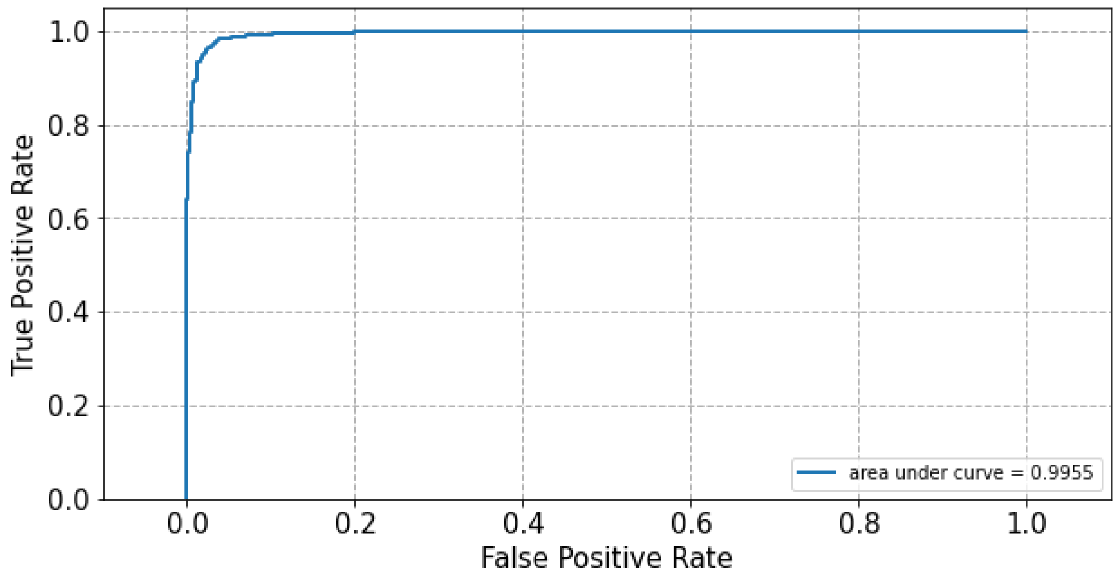
True label
0
1
524
1036
0
1590
0
1
Predicted label
1400
1200
1000
800
600
400
200
0

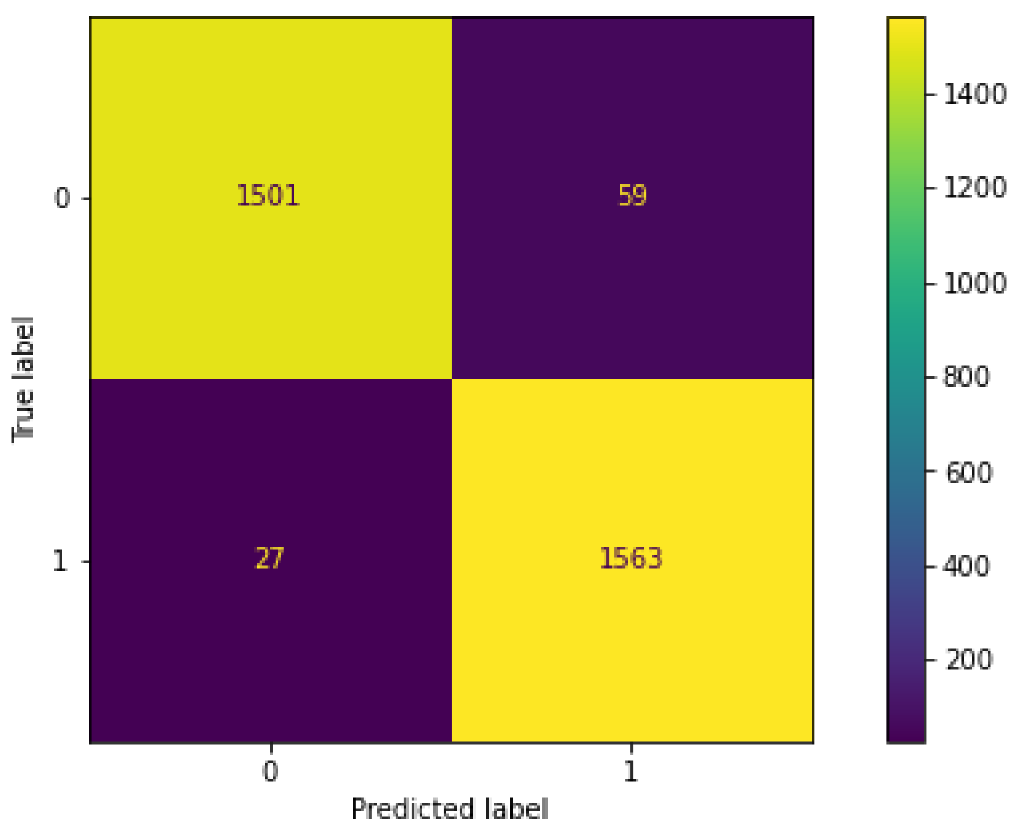


area under curve = 0.9959
True Positive Rate
False Positive Rate



1504
56
32
1558
True label
Predicted label


1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
area under curve = 0.9955


1501
59
27
1563
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200

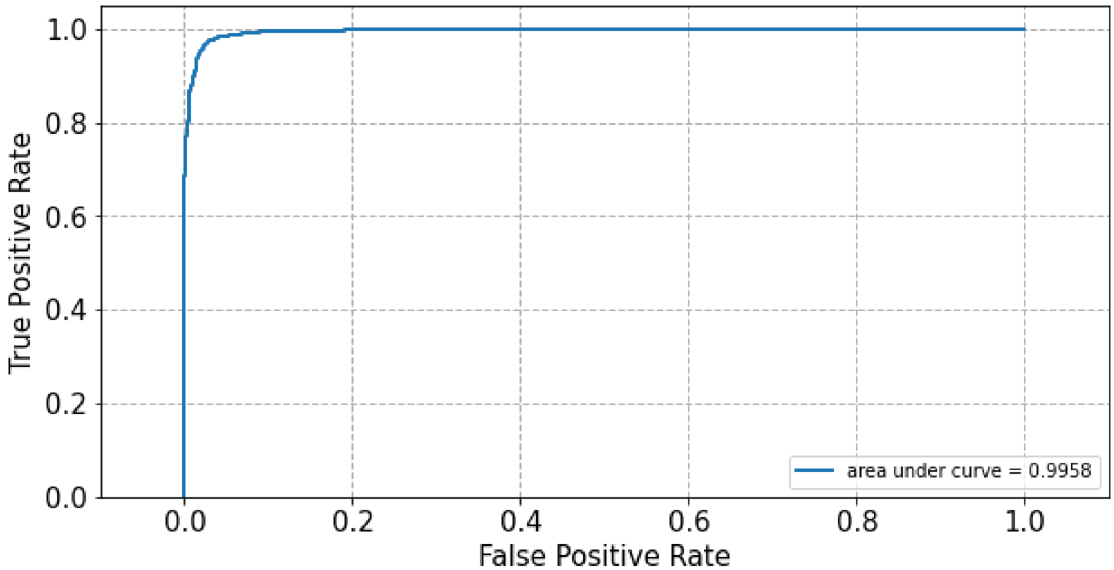
[11]:

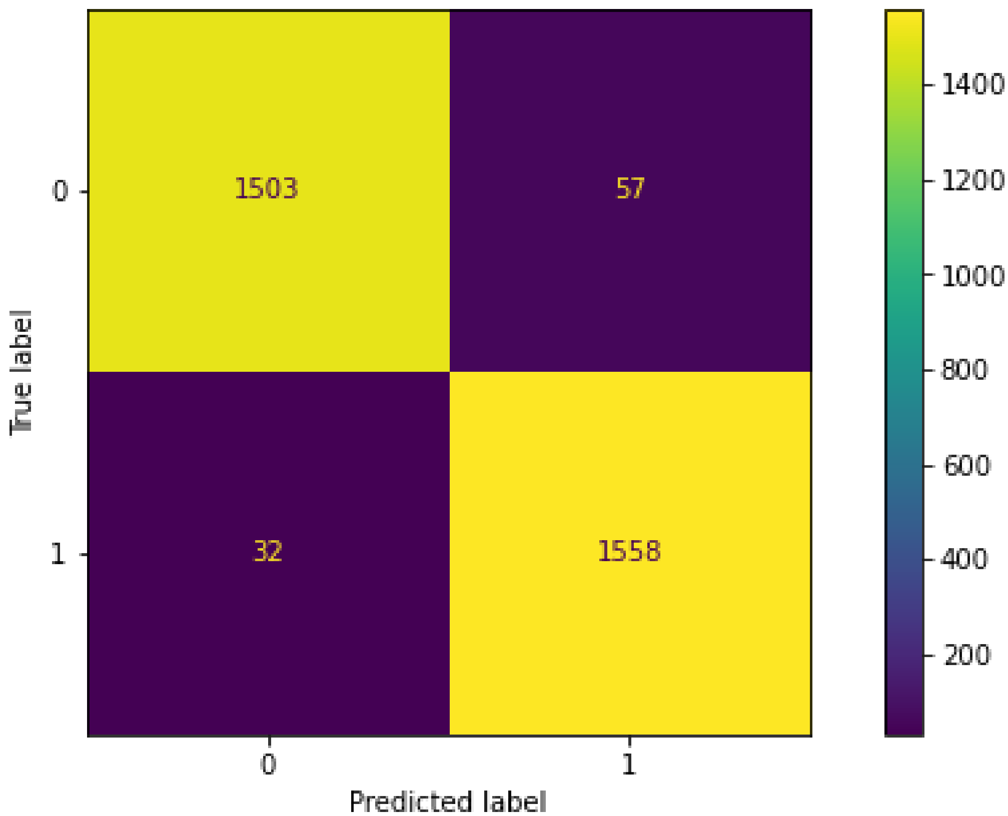
```
#Q5
from sklearn.linear_model import LogisticRegression
#Part 1
#C is "Inverse of regularization strength", so a really big value approximates␣
↪no regularization
log_class_no_reg = LogisticRegression(C=9999999999,solver='liblinear').
↪fit(X_train_lsi, train_targets_bin)
log_class_no_reg_prediction_func = log_class_no_reg.decision_function(X_test_lsi)
log_class_no_reg_prediction = log_class_no_reg.predict(X_test_lsi)

fpr, tpr, threshold = metrics.roc_curve(test_targets_bin,␣
↪log_class_no_reg_prediction_func, pos_label=1)

plot_roc(fpr,tpr)
metrics.plot_confusion_matrix(log_class_no_reg, X_test_lsi, test_targets_bin)
print("Non-regularized logistic classifier stats:")
print(metrics.
↪classification_report(test_targets_bin,log_class_no_reg_prediction,digits=4))

#Part 2a - finding best regularization parameter
def find_best_reg(ks, reg_type):
    best_k = -100
    best_acc = 0
    for i in ks:
        k_LR = LogisticRegression(C=10**i, penalty=reg_type,solver='liblinear').
↪fit(X_train_lsi, train_targets_bin)
        k_prediction = k_LR.predict(X_test_lsi)
        scores = metrics.confusion_matrix(test_targets_bin,k_prediction)
        acc = (scores[0][0]+ scores[1][1])/(scores[0][0]+ scores[0][1] +␣
↪scores[1][0]+ scores[1][1])
        if acc > best_acc:
            best_k = i
            best_acc = acc
    return best_k

k_l1 = find_best_reg([-3,-2,-1,0,1,2,3], "l1")
k_l2 = find_best_reg([-3,-2,-1,0,1,2,3], "l2")

#part 2b: getting stats for optimal l1 and l2 reg parameters (no-reg done in␣
↪part 1)

#l1 case
log_class_l1 = LogisticRegression(C=10**k_l1, penalty="l1",solver='liblinear').
↪fit(X_train_lsi, train_targets_bin)
log_class_l1_prediction_func = log_class_l1.decision_function(X_test_lsi)
log_class_l1_prediction = log_class_l1.predict(X_test_lsi)
```

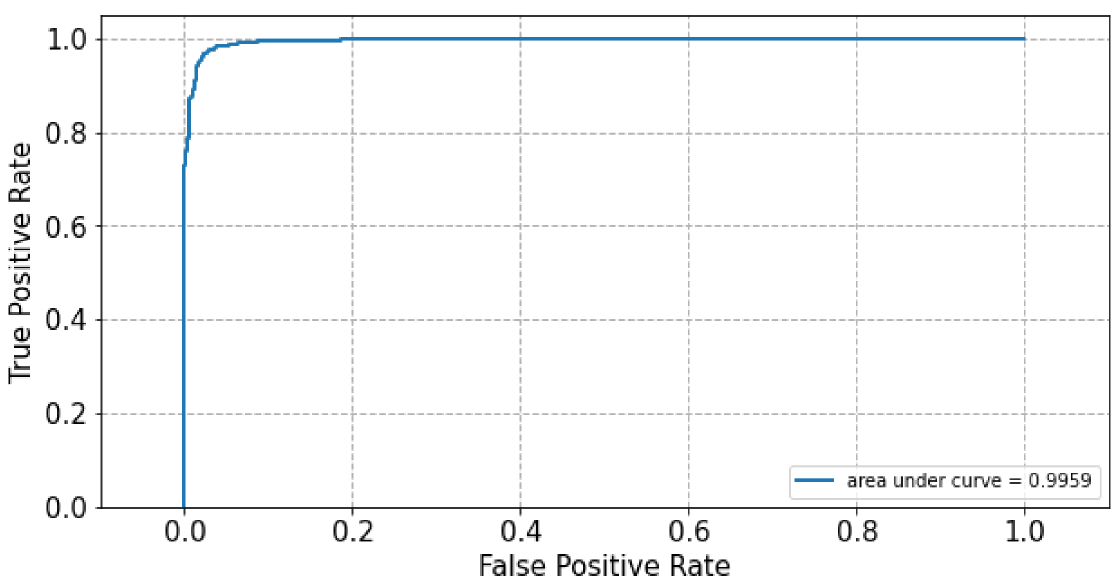
```
fpr_l1, tpr_l1, threshold_l1 = metrics.roc_curve(test_targets_bin,␣
 ↪log_class_l1_prediction_func, pos_label=1)

plot_roc(fpr_l1,tpr_l1)
metrics.plot_confusion_matrix(log_class_l1, X_test_lsi, test_targets_bin)
print("L1 logistic classifier stats with C = " + str(k_l1))
print(metrics.
 ↪classification_report(test_targets_bin,log_class_l1_prediction,digits=4))

#l2 case
log_class_l2 = LogisticRegression(C=10**k_l2, penalty="l2",solver='liblinear').
 ↪fit(X_train_lsi, train_targets_bin)
log_class_l2_prediction_func = log_class_l2.decision_function(X_test_lsi)
log_class_l2_prediction = log_class_l2.predict(X_test_lsi)

fpr_l2, tpr_l2, threshold_l2 = metrics.roc_curve(test_targets_bin,␣
 ↪log_class_l2_prediction_func, pos_label=1)

plot_roc(fpr_l2,tpr_l2)
metrics.plot_confusion_matrix(log_class_l2, X_test_lsi, test_targets_bin)
print("L2 logistic classifier stats with C = " + str(k_l2))
print(metrics.
 ↪classification_report(test_targets_bin,log_class_l2_prediction,digits=4))
```

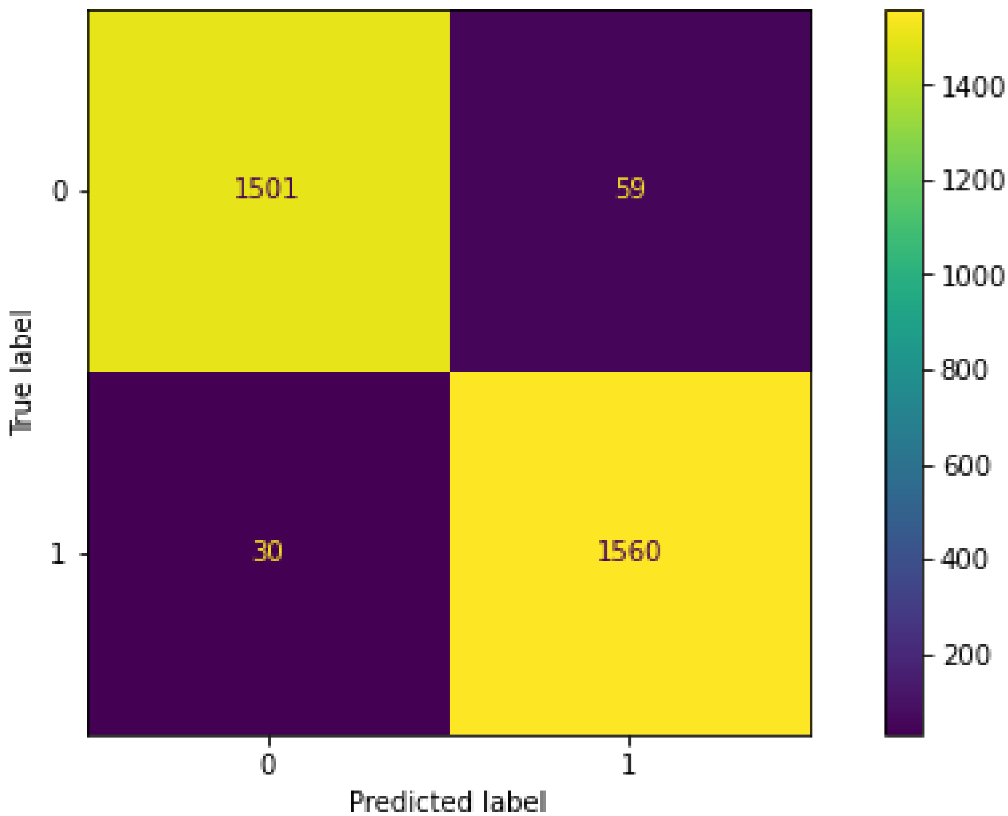
```
Non-regularized logistic classifier stats:
              precision    recall  f1-score   support

           0     0.9792    0.9635    0.9712      1560
           1     0.9647    0.9799    0.9722      1590

    accuracy                         0.9717      3150
   macro avg     0.9719    0.9717    0.9717      3150
weighted avg     0.9719    0.9717    0.9717      3150

L1 logistic classifier stats with C = 1
              precision    recall  f1-score   support

           0     0.9804    0.9622    0.9712      1560
           1     0.9636    0.9811    0.9723      1590

    accuracy                         0.9717      3150
   macro avg     0.9720    0.9717    0.9717      3150
weighted avg     0.9719    0.9717    0.9717      3150

L2 logistic classifier stats with C = 1
              precision    recall  f1-score   support
```

0	0.9823	0.9603	0.9712	1560
1	0.9618	0.9830	0.9723	1590
accuracy			0.9717	3150
macro avg	0.9721	0.9716	0.9717	3150
weighted avg	0.9720	0.9717	0.9717	3150

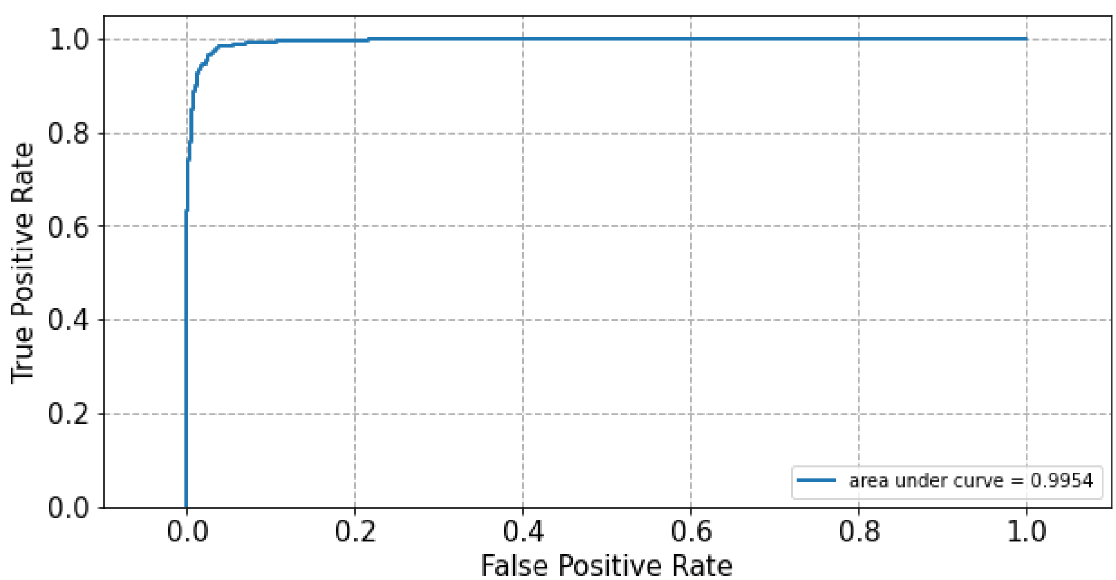



1503
57
32
1558
0
1
True label
0
1
Predicted label
1400
1200
1000
800
600
400
200

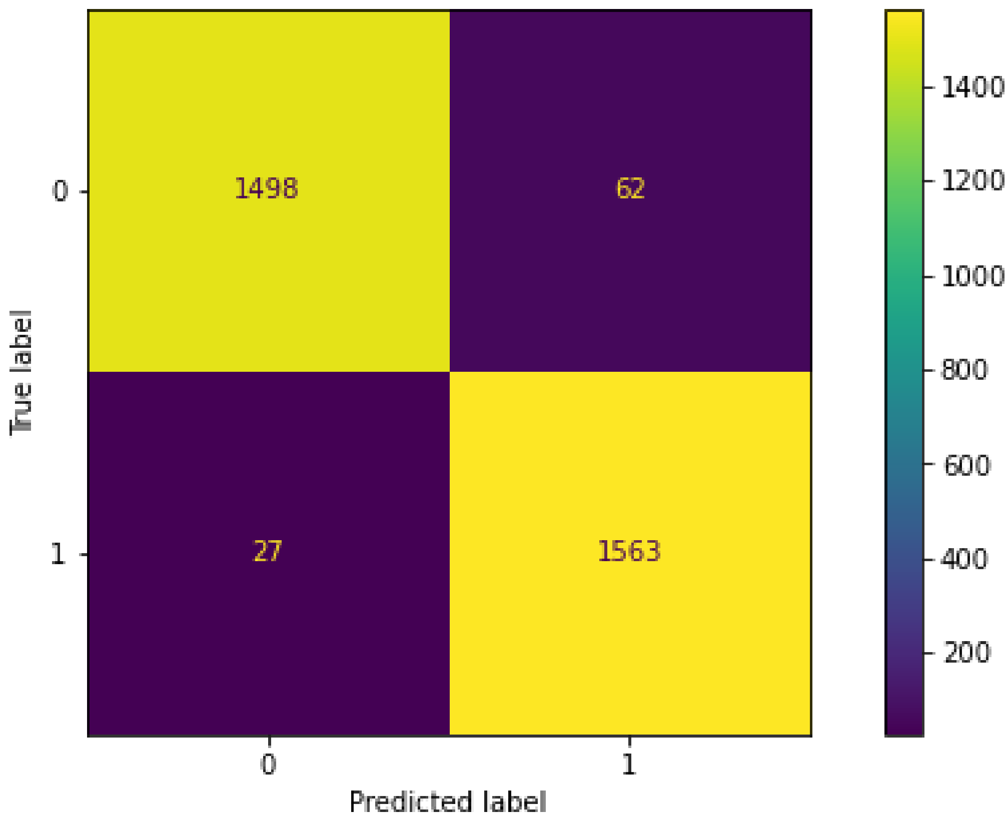

1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
area under curve = 0.9959



1501
59
30
1560
0
1
True label
Predicted label
1400
1200
1000
800
600
400
200



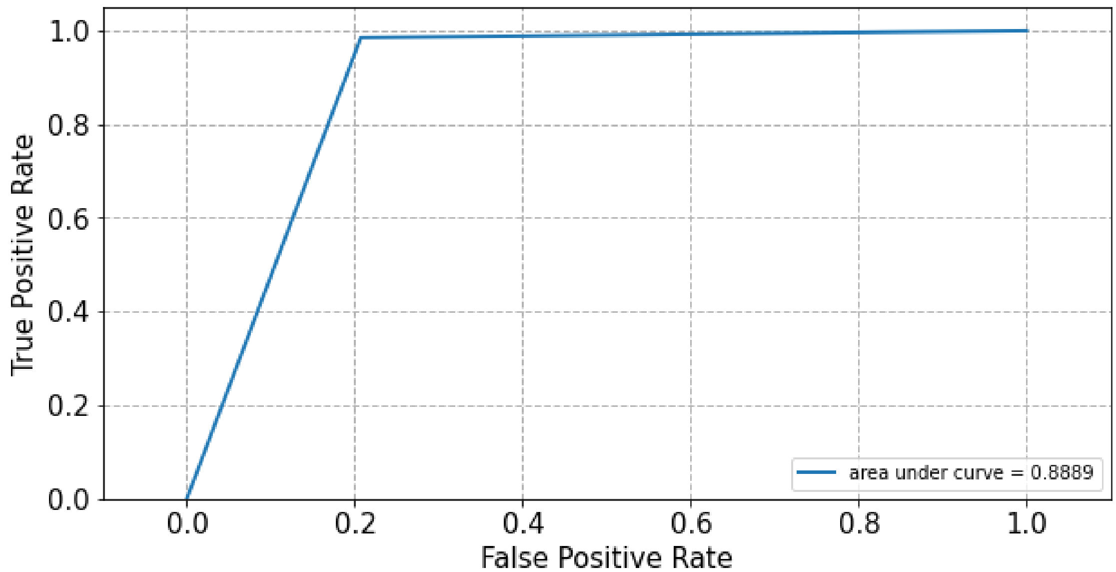
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
area under curve = 0.9954

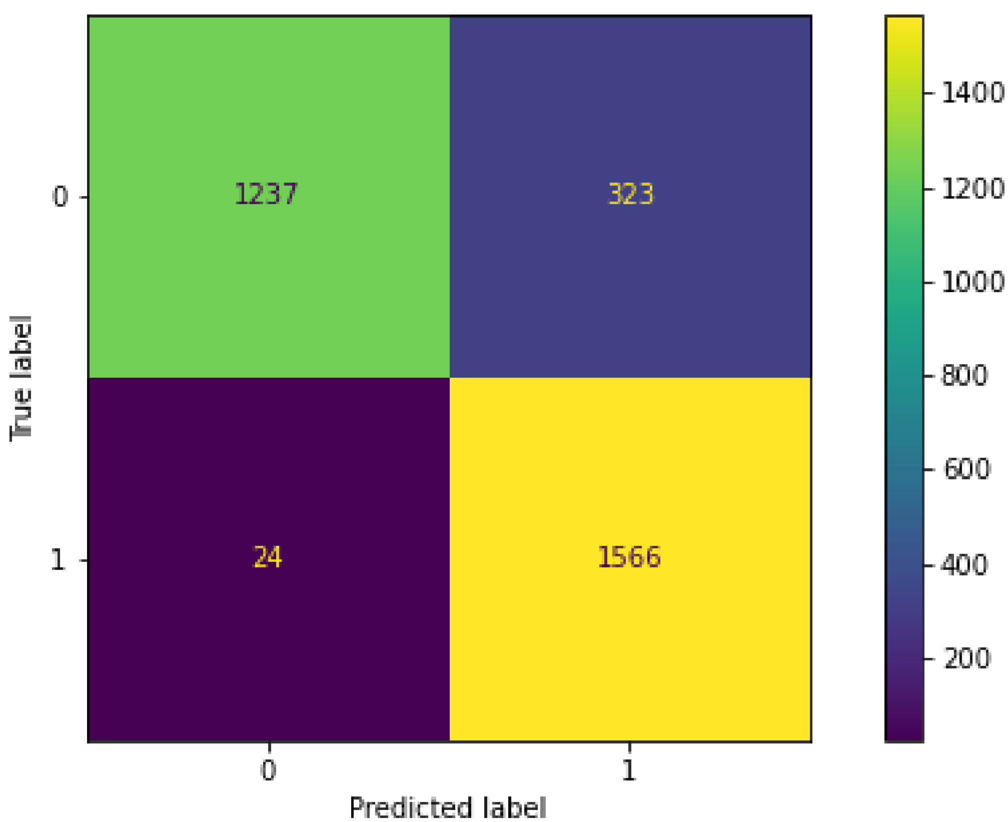


[12]:
```
#Q6
from sklearn.naive_bayes import GaussianNB

gauss_NB = GaussianNB().fit(X_train_lsi, train_targets_bin)
gauss_NB_prediction = gauss_NB.predict(X_test_lsi)

fpr_NB, tpr_NB, threshold_NB = metrics.roc_curve(test_targets_bin,␣
 ↪gauss_NB_prediction, pos_label=1)

plot_roc(fpr_NB,tpr_NB)
metrics.plot_confusion_matrix(gauss_NB, X_test_lsi, test_targets_bin)
print("Naive Bayes classification stats:")
print(metrics.classification_report(test_targets_bin,gauss_NB_prediction))
```

```
Naive Bayes classification stats:
              precision    recall  f1-score   support

           0       0.98      0.79      0.88      1560
           1       0.83      0.98      0.90      1590

    accuracy                           0.89      3150
```

macro avg	0.90	0.89	0.89	3150
weighted avg	0.90	0.89	0.89	3150





[13]:
```
#Q7
#Can we do two grid searches, one for each set of data, then pick the best of␣
↪the two for our best parameters?
#Are we allowed to remove headers/footers when loading in the data? Instead of␣
↪using the code given to us
#What is test score rank? From table or from evaluating on the test set? For the␣
↪former, do we take the best performance between the two datasets?
# How will it be graded? What if my best combination is different that your␣
↪default one?

from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
# used to cache results
from tempfile import mkdtemp
from shutil import rmtree
import pandas as pd

import joblib

def stem_rmv_punc_nolem(doc):
    return (word for word in analyzer(doc) if word not in combined_stopwords and␣
↪not word.isdigit())

class EstimatorSelectionHelper:

    def __init__(self, models, params):
        if not set(models.keys()).issubset(set(params.keys())):
            missing_params = list(set(models.keys()) - set(params.keys()))
            raise ValueError("Some estimators are missing parameters: %s" %␣
↪missing_params)
        self.models = models
        self.params = params
        self.keys = models.keys()
        self.grid_searches = {}

    def fit(self, X, y, cv=3, n_jobs=3, verbose=1, scoring=None, refit=False):
        for key in self.keys:
            print("Running GridSearchCV for %s." % key)
            model = self.models[key]
            params = self.params[key]
            gs = GridSearchCV(model, params, cv=cv, n_jobs=n_jobs,
                              verbose=verbose, scoring=scoring, refit=refit,
                              return_train_score=True)
            gs.fit(X,y)
```

```
            self.grid_searches[key] = gs

    def score_summary(self, sort_by='mean_score'):
        def row(key, scores, params):
            d = {
                 'estimator': key,
                 'min_score': min(scores),
                 'max_score': max(scores),
                 'mean_score': np.mean(scores),
                 'std_score': np.std(scores),
            }
            return pd.Series({**params,**d})

        rows = []
        for k in self.grid_searches:
            print(k)
            params = self.grid_searches[k].cv_results_['params']
            scores = []
            for i in range(self.grid_searches[k].cv):
                key = "split{}_test_score".format(i)
                r = self.grid_searches[k].cv_results_[key]
                scores.append(r.reshape(len(params),1))

            all_scores = np.hstack(scores)
            for p, s in zip(params,all_scores):
                rows.append((row(k, s, p)))

        df = pd.concat(rows, axis=1).T.sort_values([sort_by], ascending=False)

        columns = ['estimator', 'min_score', 'mean_score', 'max_score',␣
 ↪'std_score']
        columns = columns + [c for c in df.columns if c not in columns]

        return df[columns]

#Get train and test set with no headers/footers, and turn classification to␣
 ↪binary
train_dataset_nohf = fetch_20newsgroups(subset = 'train', categories =␣
 ↪categories, shuffle = True, random_state = 42, remove=('headers','footers'))
test_dataset_nohf = fetch_20newsgroups(subset = 'test', categories = categories,␣
 ↪shuffle = True, random_state = 42, remove=('headers','footers'))

train_targets_nohf_bin = train_dataset_nohf.target.copy()
for i in range(len(train_targets_nohf_bin)):
    if train_dataset_nohf.target[i] in [0,1,2,3]:
        train_targets_nohf_bin[i] = 0
    else:
```

```
        train_targets_nohf_bin[i] = 1

test_targets_nohf_bin = test_dataset_nohf.target.copy()
for i in range(len(test_targets_nohf_bin)):
    if test_dataset_nohf.target[i] in [0,1,2,3]:
        test_targets_nohf_bin[i] = 0
    else:
        test_targets_nohf_bin[i] = 1

#init pipeline
cachedir = mkdtemp()
memory = joblib.Memory(cachedir=cachedir, verbose=10)
pipeline = Pipeline([
    ('vect', None),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', None),
    ('clf', None),
],
memory=memory
)

#define pipeline params
param_grid = [
    {
        'vect': [CountVectorizer(analyzer=stem_rmv_punc, stop_words='english'),␣
 ↪CountVectorizer(analyzer=stem_rmv_punc_nolem, stop_words='english')],
        'reduce_dim': [NMF(n_components=50, init='random', random_state=42,␣
 ↪max_iter=1000), TruncatedSVD(n_components=50, random_state=42)],
        'clf': [LogisticRegression(C=10, penalty="l2", solver='liblinear',␣
 ↪max_iter=10000), LogisticRegression(C=10, penalty="l1", solver='liblinear',␣
 ↪max_iter=10000), LinearSVC(C=1, max_iter=100000),  GaussianNB()],
        'vect__min_df':[3,5],
    },
]

models_in = {
    'Pipeline': pipeline
}

params_in = {
    'Pipeline': param_grid
}

#Run grid search
grid_search = EstimatorSelectionHelper(models_in, params_in)
grid_search.fit(train_dataset.data, train_targets_bin, n_jobs=1, cv=5,␣
 ↪scoring='accuracy')
```

```
grid_search.score_summary(sort_by='mean_score')


#grid = GridSearchCV(pipeline, cv=5, n_jobs=1, param_grid=param_grid,␣
 ↪scoring='accuracy')
#grid.fit(train_dataset.data, train_targets_bin)
```

```
<ipython-input-13-c4bfbd7c95e5>:93: DeprecationWarning: The 'cachedir' parameter
has been deprecated in version 0.12 and will be removed in version 0.14.
You provided "cachedir='C:\\Users\\Zoe\\AppData\\Local\\Temp\\tmplqaajv4i'", use
"location='C:\\Users\\Zoe\\AppData\\Local\\Temp\\tmplqaajv4i'" instead.
  memory = joblib.Memory(cachedir=cachedir, verbose=10)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
```

```
Running GridSearchCV for Pipeline.
Fitting 5 folds for each of 32 candidates, totalling 160 fits
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-13-c4bfbd7c95e5> in <module>
    121 #Run grid search
    122 grid_search = EstimatorSelectionHelper(models_in, params_in)
--> 123 grid_search.fit(train_dataset.data, train_targets_bin, n_jobs=1, cv=5,␣
 ↪scoring='accuracy')
    124 grid_search.score_summary(sort_by='mean_score')
    125

<ipython-input-13-c4bfbd7c95e5> in fit(self, X, y, cv, n_jobs, verbose, scoring,␣
 ↪refit)
     36                                   verbose=verbose, scoring=scoring,␣
 ↪refit=refit,
     37                                   return_train_score=True)
---> 38             gs.fit(X,y)
     39             self.grid_searches[key] = gs
     40

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args,␣
 ↪**kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X,␣
 ↪y, groups, **fit_params)
    734                 return results
    735
```

```
--> 736             self._run_search(evaluate_candidates)
    737
    738         # For multi-metric evaluation, store the best_index_, best_params_␣
 ↪and

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in␣
 ↪_run_search(self, evaluate_candidates)
   1186     def _run_search(self, evaluate_candidates):
   1187         """Search all candidates in param_grid"""
-> 1188         evaluate_candidates(ParameterGrid(self.param_grid))
   1189
   1190

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in␣
 ↪evaluate_candidates(candidate_params)
    706                               n_splits, n_candidates, n_candidates *␣
 ↪n_splits))
    707
--> 708                 out =␣
 ↪parallel(delayed(_fit_and_score)(clone(base_estimator),
    709                                                          X, y,
    710                                                          train=train,␣
 ↪test=test,

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1046             # remaining jobs.
   1047             self._iterating = False
-> 1048             if self.dispatch_one_batch(iterator):
   1049                 self._iterating = self._original_iterator is not None
   1050

~\anaconda3\lib\site-packages\joblib\parallel.py in dispatch_one_batch(self,␣
 ↪iterator)
    864                 return False
    865             else:
--> 866                 self._dispatch(tasks)
    867                 return True
    868

~\anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self, batch)
    782         with self._lock:
    783             job_idx = len(self._jobs)
--> 784             job = self._backend.apply_async(batch, callback=cb)
    785             # A job can complete so quickly than its callback is
    786             # called before we get here, causing self._jobs to
```

```
~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in apply_async(self,␣
↪func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573 
    574     def get(self):

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)

    263                     for func, args, kwargs in self.items]
    264 

~\anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)

    263                     for func, args, kwargs in self.items]
    264 

~\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in␣
↪_fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters,␣
↪fit_params, return_train_score, return_parameters, return_n_test_samples,␣
↪return_times, return_estimator, error_score)
    529             estimator.fit(X_train, **fit_params)
    530         else:
--> 531             estimator.fit(X_train, y_train, **fit_params)
    532 
    533     except Exception as e:

~\anaconda3\lib\site-packages\sklearn\pipeline.py in fit(self, X, y, **fit_params)
    328         """
    329         fit_params_steps = self._check_fit_params(**fit_params)
--> 330         Xt = self._fit(X, y, **fit_params_steps)
    331         with _print_elapsed_time('Pipeline',
    332                                  self._log_message(len(self.steps) - 1)):
```

```
~\anaconda3\lib\site-packages\sklearn\pipeline.py in _fit(self, X, y,␣
↪**fit_params_steps)
    290                 cloned_transformer = clone(transformer)
    291             # Fit or load from cache the current transformer
--> 292             X, fitted_transformer = fit_transform_one_cached(
    293                 cloned_transformer, X, y, None,
    294                 message_clsname='Pipeline',

~\anaconda3\lib\site-packages\joblib\memory.py in __call__(self, *args, **kwargs)
    589
    590     def __call__(self, *args, **kwargs):
--> 591         return self._cached_call(args, kwargs)[0]
    592
    593     def __getstate__(self):

~\anaconda3\lib\site-packages\joblib\memory.py in _cached_call(self, args, kwargs,␣
↪shelving)
    532
    533         if must_call:
--> 534             out, metadata = self.call(*args, **kwargs)
    535             if self.mmap_mode is not None:
    536                 # Memmap the output at the first call to be consistent with

~\anaconda3\lib\site-packages\joblib\memory.py in call(self, *args, **kwargs)
    758         func_id, args_id = self._get_output_identifiers(*args, **kwargs)
    759         if self._verbose > 0:
--> 760             print(format_call(self.func, args, kwargs))
    761         output = self.func(*args, **kwargs)
    762         self.store_backend.dump_item(

~\anaconda3\lib\site-packages\joblib\func_inspect.py in format_call(func, args,␣
↪kwargs, object_name)
    338         call with the given arguments.
    339     """
--> 340     path, signature = format_signature(func, *args, **kwargs)
    341     msg = '%s\n[%s] Calling %s...\n%s' % (80 * '_', object_name,
    342                                           path, signature)

~\anaconda3\lib\site-packages\joblib\func_inspect.py in format_signature(func,␣
↪*args, **kwargs)
    322     previous_length = 0
    323     for arg in args:
--> 324         formatted_arg = _format_arg(arg)
    325         if previous_length > 80:
    326             formatted_arg = '\n%s' % formatted_arg

~\anaconda3\lib\site-packages\joblib\func_inspect.py in _format_arg(arg)
```

```
    304
    305 def _format_arg(arg):
--> 306     formatted_arg = pformat(arg, indent=2)
    307     if len(formatted_arg) > 1500:
    308         formatted_arg = '%s...' % formatted_arg[:700]

~\anaconda3\lib\site-packages\joblib\logger.py in pformat(obj, indent, depth)
     52     else:
     53         print_options = None
---> 54     out = pprint.pformat(obj, depth=depth, indent=indent)
     55     if print_options:
     56         np.set_printoptions(**print_options)

~\anaconda3\lib\pprint.py in pformat(object, indent, width, depth, compact, sort_dicts)
     56             compact=False, sort_dicts=True):
     57     """Format a Python object into a pretty-printed representation."""
---> 58     return PrettyPrinter(indent=indent, width=width, depth=depth,
     59                          compact=compact, sort_dicts=sort_dicts).pformat(object)
     60

~\anaconda3\lib\pprint.py in pformat(self, object)
    151     def pformat(self, object):
    152         sio = _StringIO()
--> 153         self._format(object, sio, 0, 0, {}, 0)
    154         return sio.getvalue()
    155

~\anaconda3\lib\pprint.py in _format(self, object, stream, indent, allowance, context, level)
    174             if p is not None:
    175                 context[objid] = 1
--> 176                 p(self, object, stream, indent, allowance, context, level + 1)
    177                 del context[objid]
    178                 return

~\anaconda3\lib\pprint.py in _pprint_list(self, object, stream, indent, allowance, context, level)
    219     def _pprint_list(self, object, stream, indent, allowance, context, level):
    220         stream.write('[')
--> 221         self._format_items(object, stream, indent, allowance + 1,
    222                            context, level)
    223         stream.write(']')
```

```
~\anaconda3\lib\pprint.py in _format_items(self, items, stream, indent, allowance,␣
↪context, level)
    397                 write(delim)
    398                 delim = delimnl
--> 399                 self._format(ent, stream, indent,

    400                              allowance if last else 1,
    401                              context, level)

~\anaconda3\lib\pprint.py in _format(self, object, stream, indent, allowance,␣
↪context, level)
    174             if p is not None:
    175                 context[objid] = 1
--> 176                 p(self, object, stream, indent, allowance, context, level␣
↪+ 1)
    177                 del context[objid]
    178                 return

~\anaconda3\lib\pprint.py in _pprint_str(self, object, stream, indent, allowance,␣
↪context, level)
    298         for i, rep in enumerate(chunks):
    299             if i > 0:
--> 300                 write('\n' + ' '*indent)
    301             write(rep)
    302         if level == 1:

KeyboardInterrupt:
```

[14]:

```
grid_search.fit(train_dataset_nohf.data, train_targets_nohf_bin, n_jobs=1, cv=5,␣
↪scoring='accuracy')
grid_search.score_summary(sort_by='mean_score')
rmtree(cachedir)
```

```
Running GridSearchCV for Pipeline.
Fitting 5 folds for each of 32 candidates, totalling 160 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.

________________________________________________________________________________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer=<function stem_rmv_punc at
0x000001B849C74940>,
                min_df=3, stop_words='english'),
[ 'Hank Greenberg was probably the greatest ever.  He was also subject to a\n'
  'lot of heckling from bigots on the opposing teams and in the stands, but\n'
  'it never seemed to affect his performance negatively.',
  'On March 21, 1993 Roger Maynard wrote (in reply to an article by Graham\n'
  'Hudson):\n'
```

```
  '\n'
  '>> will still have the Jennings Trophy at the end of the year.  Potvin is '
  'very\n'
  '>> good, and I do believe that he will be a star, but I want to see him\n'
  '>> perform in the playoffs under pressure.\n'
  '\n'
  '>You don\'t think he is performing "under pressure" now?  The major\n'
  '>differences  between playoff hockey and normal hockey is 1. play-\n'
  '>ing ever...,
array([1, ..., 0], dtype=int64), None, message_clsname='Pipeline', message=None)
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
~\anaconda3\lib\genericpath.py in isfile(path)
     29     try:
---> 30         st = os.stat(path)
     31     except (OSError, ValueError):

FileNotFoundError: [WinError 2] The system cannot find the file specified: 'C:
 ↪\\Users\\Zoe/nltk_data'

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-14-0de4e484a0eb> in <module>
----> 1 grid_search.fit(train_dataset_nohf.data, train_targets_nohf_bin, n_jobs=1,␣
 ↪cv=5, scoring='accuracy')
      2 grid_search.score_summary(sort_by='mean_score')
      3 rmtree(cachedir)

<ipython-input-13-c4bfbd7c95e5> in fit(self, X, y, cv, n_jobs, verbose, scoring,␣
 ↪refit)
     36                                   verbose=verbose, scoring=scoring,␣
 ↪refit=refit,
     37                                   return_train_score=True)
---> 38             gs.fit(X,y)
     39             self.grid_searches[key] = gs
     40

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in inner_f(*args,␣
 ↪**kwargs)
     70                               FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74
```

```
~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in fit(self, X,␣
↪y, groups, **fit_params)
    734                 return results
    735 
--> 736             self._run_search(evaluate_candidates)
    737 
    738         # For multi-metric evaluation, store the best_index_, best_params_␣
↪and

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in␣
↪_run_search(self, evaluate_candidates)
   1186     def _run_search(self, evaluate_candidates):
   1187         """Search all candidates in param_grid"""
-> 1188         evaluate_candidates(ParameterGrid(self.param_grid))
   1189 
   1190 

~\anaconda3\lib\site-packages\sklearn\model_selection\_search.py in␣
↪evaluate_candidates(candidate_params)
    706                               n_splits, n_candidates, n_candidates *␣
↪n_splits))
    707 
--> 708                 out =␣
↪parallel(delayed(_fit_and_score)(clone(base_estimator),
    709                                                        X, y,
    710                                                        train=train,␣
↪test=test,

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self, iterable)
   1046             # remaining jobs.
   1047             self._iterating = False
-> 1048             if self.dispatch_one_batch(iterator):
   1049                 self._iterating = self._original_iterator is not None
   1050 

~\anaconda3\lib\site-packages\joblib\parallel.py in dispatch_one_batch(self,␣
↪iterator)
    864                 return False
    865             else:
--> 866                 self._dispatch(tasks)
    867                 return True
    868 

~\anaconda3\lib\site-packages\joblib\parallel.py in _dispatch(self, batch)
    782         with self._lock:
    783             job_idx = len(self._jobs)
--> 784             job = self._backend.apply_async(batch, callback=cb)
```

```
    785             # A job can complete so quickly than its callback is
    786             # called before we get here, causing self._jobs to

~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in apply_async(self, func, callback)
    206     def apply_async(self, func, callback=None):
    207         """Schedule a func to be run"""
--> 208         result = ImmediateResult(func)
    209         if callback:
    210             callback(result)

~\anaconda3\lib\site-packages\joblib\_parallel_backends.py in __init__(self, batch)
    570         # Don't delay the application, to avoid keeping the input
    571         # arguments in memory
--> 572         self.results = batch()
    573
    574     def get(self):

~\anaconda3\lib\site-packages\joblib\parallel.py in __call__(self)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264

~\anaconda3\lib\site-packages\joblib\parallel.py in <listcomp>(.0)
    260         # change the default number of processes to -1
    261         with parallel_backend(self._backend, n_jobs=self._n_jobs):
--> 262             return [func(*args, **kwargs)
    263                     for func, args, kwargs in self.items]
    264

~\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py in _fit_and_score(estimator, X, y, scorer, train, test, verbose, parameters, fit_params, return_train_score, return_parameters, return_n_test_samples, return_times, return_estimator, error_score)
    529             estimator.fit(X_train, **fit_params)
    530         else:
--> 531             estimator.fit(X_train, y_train, **fit_params)
    532
    533     except Exception as e:

~\anaconda3\lib\site-packages\sklearn\pipeline.py in fit(self, X, y, **fit_params)
    328         """
    329         fit_params_steps = self._check_fit_params(**fit_params)
--> 330         Xt = self._fit(X, y, **fit_params_steps)
    331         with _print_elapsed_time('Pipeline',
```

```
    332                                  self._log_message(len(self.steps) - 1)):

~\anaconda3\lib\site-packages\sklearn\pipeline.py in _fit(self, X, y, **fit_params_steps)
    290                 cloned_transformer = clone(transformer)
    291             # Fit or load from cache the current transformer
--> 292             X, fitted_transformer = fit_transform_one_cached(

    293                 cloned_transformer, X, y, None,
    294                 message_clsname='Pipeline',

~\anaconda3\lib\site-packages\joblib\memory.py in __call__(self, *args, **kwargs)
    589
    590     def __call__(self, *args, **kwargs):
--> 591         return self._cached_call(args, kwargs)[0]
    592
    593     def __getstate__(self):

~\anaconda3\lib\site-packages\joblib\memory.py in _cached_call(self, args, kwargs, shelving)
    532
    533         if must_call:
--> 534             out, metadata = self.call(*args, **kwargs)
    535             if self.mmap_mode is not None:
    536                 # Memmap the output at the first call to be consistent with

~\anaconda3\lib\site-packages\joblib\memory.py in call(self, *args, **kwargs)
    759         if self._verbose > 0:
    760             print(format_call(self.func, args, kwargs))
--> 761         output = self.func(*args, **kwargs)
    762         self.store_backend.dump_item(
    763             [func_id, args_id], output, verbose=self._verbose)

~\anaconda3\lib\site-packages\sklearn\pipeline.py in _fit_transform_one(transformer, X, y, weight, message_clsname, message, **fit_params)
    738     with _print_elapsed_time(message_clsname, message):
    739         if hasattr(transformer, 'fit_transform'):
--> 740             res = transformer.fit_transform(X, y, **fit_params)
    741         else:
    742             res = transformer.fit(X, y, **fit_params).transform(X)

~\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in fit_transform(self, raw_documents, y)
   1196         max_features = self.max_features
   1197
-> 1198         vocabulary, X = self._count_vocab(raw_documents,

   1199                                           self.fixed_vocabulary_)
```

```
   1200

~\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in _count_vocab(self, raw_documents, fixed_vocab)
   1108         for doc in raw_documents:
   1109             feature_counter = {}
-> 1110             for feature in analyze(doc):
   1111                 try:
   1112                     feature_idx = vocabulary[feature]

~\anaconda3\lib\site-packages\sklearn\feature_extraction\text.py in _analyze(doc, analyzer, tokenizer, ngrams, preprocessor, decoder, stop_words)
     99         doc = decoder(doc)
    100     if analyzer is not None:
--> 101         doc = analyzer(doc)
    102     else:
    103         if preprocessor is not None:

<ipython-input-3-40bbdb0933d3> in stem_rmv_punc(doc)
     39 #Updated analyzer to avoid counting digits and punctuation
     40 def stem_rmv_punc(doc):
---> 41     return (word for word in lemmatize_sent(analyzer(doc)) if word not in combined_stopwords and not word.isdigit())
     42
     43 count_vect = CountVectorizer(min_df=3,analyzer=stem_rmv_punc, stop_words='english')

<ipython-input-3-40bbdb0933d3> in lemmatize_sent(text)
     35
     36 def lemmatize_sent(text):
---> 37     return [lemur.lemmatize(word.lower(), pos=penn2morphy(tag)) for word, tag in pos_tag(text)]
     38
     39 #Updated analyzer to avoid counting digits and punctuation

~\anaconda3\lib\site-packages\nltk\tag\__init__.py in pos_tag(tokens, tagset, lang)
    158     :rtype: list(tuple(str, str))
    159     """
--> 160     tagger = _get_tagger(lang)
    161     return _pos_tag(tokens, tagset, tagger, lang)
    162

~\anaconda3\lib\site-packages\nltk\tag\__init__.py in _get_tagger(lang)
    104         tagger.load(ap_russian_model_loc)
    105     else:
--> 106         tagger = PerceptronTagger()
    107     return tagger
    108
```

```
~\anaconda3\lib\site-packages\nltk\tag\perceptron.py in __init__(self, load)
    166         if load:
    167             AP_MODEL_LOC = "file:" + str(
--> 168                 find("taggers/averaged_perceptron_tagger/" + PICKLE)
    169             )
    170             self.load(AP_MODEL_LOC)

~\anaconda3\lib\site-packages\nltk\data.py in find(resource_name, paths)
    522     for path_ in paths:
    523         # Is the path item a zipfile?
--> 524         if path_ and (os.path.isfile(path_) and path_.endswith(".zip")):
    525             try:
    526                 return ZipFilePathPointer(path_, resource_name)

~\anaconda3\lib\genericpath.py in isfile(path)
     28     """Test whether a path is a regular file"""
     29     try:
---> 30         st = os.stat(path)
     31     except (OSError, ValueError):
     32         return False

KeyboardInterrupt:
```

[15]:

```
#Q7 get test values

#Models with best params (min_df = 3 by default the best)
model_l2 = LogisticRegression(C=10, penalty="l2", solver='liblinear',␣
 ↪max_iter=10000)
model_l1 = LogisticRegression(C=10, penalty="l1", solver='liblinear',␣
 ↪max_iter=10000)
model_svc = LinearSVC(C=1, max_iter=100000)
model_gaus = GaussianNB()

count_vect_5 = CountVectorizer(min_df=5,analyzer=stem_rmv_punc,␣
 ↪stop_words='english')

#Header/footer included performance using best params
model_l1.fit(X_train_lsi,train_targets_bin)
best_predict = model_l1.predict(X_test_lsi)
print(metrics.classification_report(test_targets_bin,best_predict,digits=4))


#No header/footer performance using best params
X_train_counts_nohf = count_vect.fit_transform(train_dataset_nohf.data)
X_train_tfidf_nohf = tfidf_transformer.fit_transform(X_train_counts_nohf)
```

```
X_test_counts_nohf = count_vect.transform(test_dataset_nohf.data)
X_test_tfidf_nohf = tfidf_transformer.transform(X_test_counts_nohf)

X_train_lsi_nohf = trunc_svd.fit_transform(X_train_tfidf_nohf)
X_test_lsi_nohf = trunc_svd.transform(X_test_tfidf_nohf)

model_l1.fit(X_train_lsi_nohf,train_targets_nohf_bin)
best_predict_nohf = model_l1.predict(X_test_lsi_nohf)
print(metrics.
 ↪classification_report(test_targets_nohf_bin,best_predict_nohf,digits=4))
```

```
              precision    recall  f1-score   support

           0     0.9810    0.9622    0.9715      1560
           1     0.9636    0.9818    0.9726      1590

    accuracy                         0.9721      3150
   macro avg     0.9723    0.9720    0.9721      3150
weighted avg     0.9722    0.9721    0.9721      3150

              precision    recall  f1-score   support

           0     0.9752    0.9577    0.9664      1560
           1     0.9592    0.9761    0.9676      1590

    accuracy                         0.9670      3150
   macro avg     0.9672    0.9669    0.9670      3150
weighted avg     0.9671    0.9670    0.9670      3150
```

[4]:
```
#Q9

embeddings_dict = {}
dimension_of_glove = 300
glove_file = "D:/glove.6B."+ str(dimension_of_glove) + "d.txt"

with open(glove_file, 'r', encoding='utf8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

[5]:
```
from sklearn.preprocessing import normalize
```

```
def glove_preprocess(input_data, input_target):
    first_vect = 0
    doc_num = 0

    #Keep track of which docs have no embeddings so that we remove those indices␣
 ↪from the target values
    bad_doc_index_list = []

    for doc in input_data:
        found_subj = 0
        found_key = 0
        final_str = ""

        #Take subject and keywords lines, and also remove the fwd/re.
        for line in doc.splitlines():
            if (not(found_subj and found_key)):
                if ("Subject:" in line and not found_subj):
                    final_str += " " + line.replace("Subject:", "").replace("Re:
 ↪","").replace("Fwd:","")
                    found_subj = 1
                elif("subject:" in line and not found_subj):
                    final_str += " " + line.replace("subject:", "").replace("Re:
 ↪","").replace("Fwd:","")
                    found_subj = 1
                elif("Keywords:" in line and not found_key):
                    final_str += " " + line.replace("Keywords:", "").replace("Re:
 ↪","").replace("Fwd:","")
                    found_key = 1
                elif("keywords:" in line and not found_key):
                    final_str += " " + line.replace("keywords:", "").replace("Re:
 ↪","").replace("Fwd:","")
                    found_key = 1
        temp = np.zeros((1,dimension_of_glove))

        count = 0

        #Lemmatize each word, then normalize and add to temp
        for elem in stem_rmv_punc(final_str):
            if (elem in embeddings_dict.keys()):
                count += 1
                temp += normalize(embeddings_dict[elem].reshape(-1,1), axis=0).T

        #Only add vector to training matrix if embeddings were found
        if (count != 0):
            #Take divide by the count to average out the word vectors for the doc
            temp /= count
            if (first_vect == 0):
```

```
                glove_data = temp
                first_vect = 1
            else:
                glove_data = np.concatenate((glove_data, temp), axis=0)
        else:
            bad_doc_index_list.append(doc_num)

        doc_num += 1


    glove_targets = input_target.copy()
    glove_targets = np.delete(glove_targets, bad_doc_index_list, axis=0)
    return glove_data, glove_targets
```

[16]:
```
glove_train, glove_train_targets = glove_preprocess(train_dataset.
 ↪data,train_targets_bin)
glove_test, glove_test_targets = glove_preprocess(test_dataset.
 ↪data,test_targets_bin)

model_l1.fit(glove_train, glove_train_targets)
glove_predict = model_l1.predict(glove_test)
print(metrics.classification_report(glove_test_targets,glove_predict,digits=4))
```

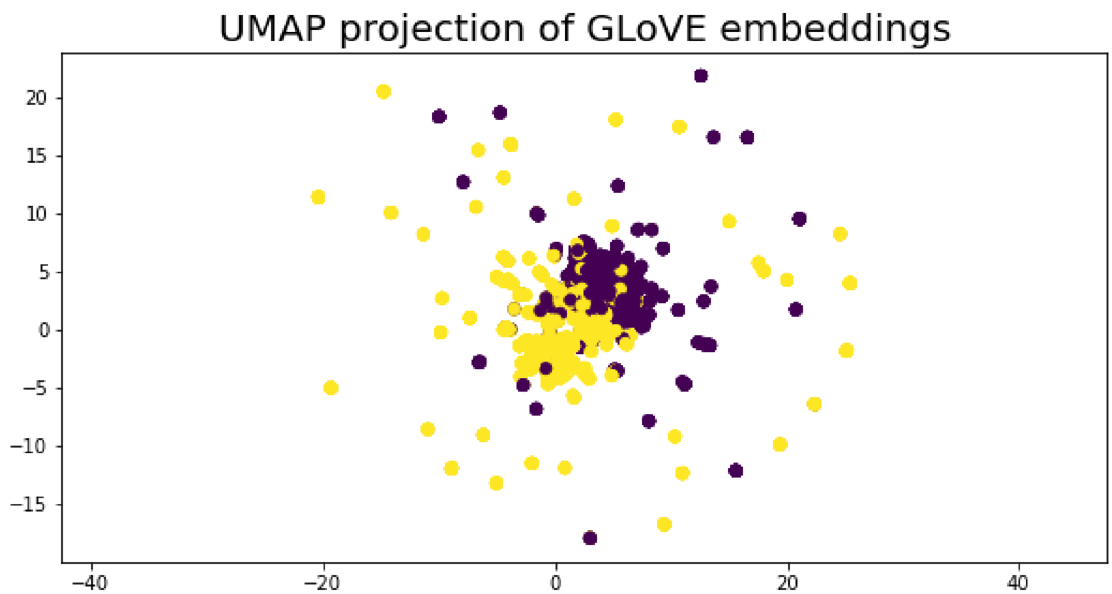
```
              precision    recall  f1-score   support

           0     0.9311    0.8960    0.9132      1539
           1     0.9023    0.9354    0.9185      1579

    accuracy                         0.9160      3118
   macro avg     0.9167    0.9157    0.9159      3118
weighted avg     0.9165    0.9160    0.9159      3118
```

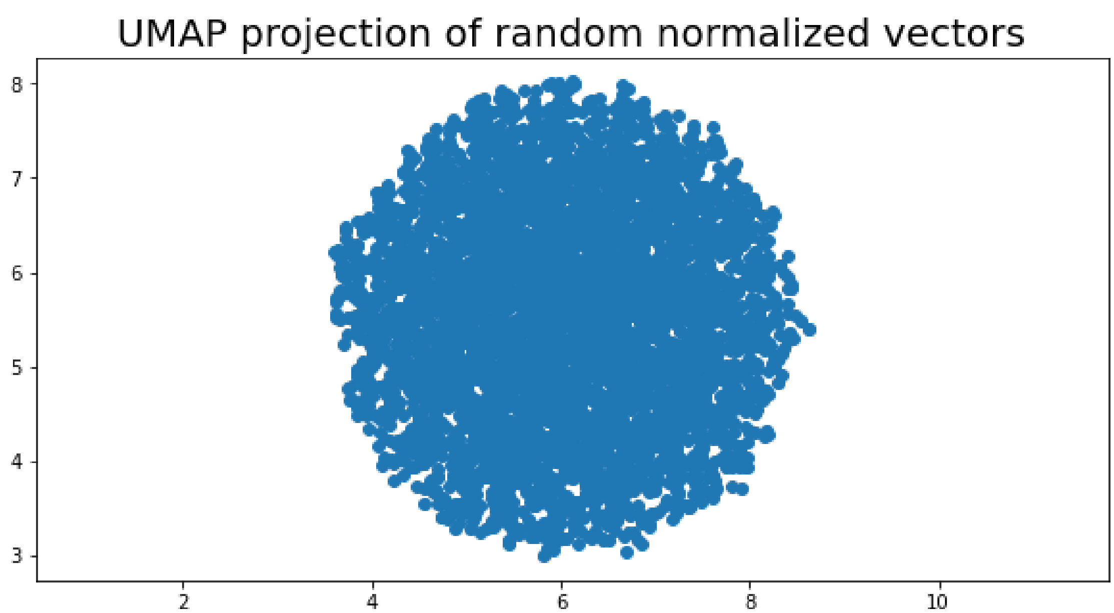
[ ]:
```
#Q10
from sklearn.metrics import accuracy_score

test_accuracies = []
dims = [50, 100, 200, 300]
for dim in dims:

    embeddings_dict = {}
    dimension_of_glove = dim
    glove_file = "glove.6B."+ str(dimension_of_glove) + "d.txt"

    with open(glove_file, 'r', encoding='utf8') as f:
        for line in f:
            values = line.split()
```

```
            word = values[0]
            vector = np.asarray(values[1:], "float32")
            embeddings_dict[word] = vector

    glove_train, glove_train_targets = glove_preprocess(train_dataset.
↪data,train_targets_bin)
    glove_test, glove_test_targets = glove_preprocess(test_dataset.
↪data,test_targets_bin)
    model_l1.fit(glove_train, glove_train_targets)
    glove_predict = model_l1.predict(glove_test)
    test_accuracies.append(accuracy_score(glove_test_targets,glove_predict))

plt.figure()
plt.scatter(dims, test_accuracies)
plt.title("GLoVE Embedding Dimension vs. Test Accuracy")
plt.xlabel("Dimension")
plt.ylabel("Test Accuracy")
plt.show()
```

[17]:
```
#Q11
import umap
reducer = umap.UMAP()
embedding = reducer.fit_transform(glove_train)

plt.scatter(embedding[:, 0], embedding[:, 1], c=glove_train_targets)
plt.gca().set_aspect('equal', 'datalim')
plt.title('UMAP projection of GLoVE embeddings', fontsize=20)
plt.show()

#random vectors for comparison
random_vecs = []
for i in range(len(glove_train)):
    vec = np.random.rand(1,300)
    norm_vec = normalize(vec)[0]
    random_vecs.append(norm_vec)

rand_reducer = umap.UMAP()
rand_embedding = reducer.fit_transform(random_vecs)

plt.scatter(rand_embedding[:, 0], rand_embedding[:, 1])
plt.gca().set_aspect('equal', 'datalim')
plt.title('UMAP projection of random normalized vectors', fontsize=20)
plt.show()
```

```
C:\Users\Zoe\anaconda3\lib\site-packages\umap\__init__.py:9: UserWarning:
Tensorflow not installed; ParametricUMAP will be unavailable
  warn("Tensorflow not installed; ParametricUMAP will be unavailable")
```





[ ]:
```
#Q12

from sklearn.svm import SVC

# Set new multiple classes
categories = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
              'misc.forsale' ,'soc.religion.christian']
```

```
#Download dataset
train_dataset = fetch_20newsgroups(subset = 'train', categories = categories,␣
↪shuffle = True, random_state = 42) #should 42 be None?
test_dataset = fetch_20newsgroups(subset = 'test', categories = categories,␣
↪shuffle = True, random_state = 42)

# Feature extraction
X_train_counts = count_vect.fit_transform(train_dataset.data)
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)

X_test_counts = count_vect.transform(test_dataset.data)
X_test_tfidf = tfidf_transformer.transform(X_test_counts)


# Dimensionality reduction
X_train_lsi = trunc_svd.fit_transform(X_train_tfidf)
X_test_lsi = trunc_svd.transform(X_test_tfidf)


# Naive Bayes
gauss_NB = GaussianNB().fit(X_train_lsi, train_dataset.target)
gauss_NB_prediction = gauss_NB.predict(X_test_lsi)

metrics.plot_confusion_matrix(gauss_NB, X_test_lsi, test_dataset.target)
print("Naive Bayes classification stats:")
print(metrics.classification_report(test_dataset.target,gauss_NB_prediction))


# One vs. One SVM
ovo_SVC = SVC(C=10**k,max_iter=50000,decision_function_shape='ovo').
↪fit(X_train_lsi, train_dataset.target)
ovo_prediction = ovo_SVC.predict(X_test_lsi)

metrics.plot_confusion_matrix(ovo_SVC, X_test_lsi, test_dataset.target)
print("One Vs. One SVM prediction stats:")
print(metrics.classification_report(test_dataset.target,ovo_prediction,digits=4))


# One vs. The Rest SVM
## LinearSVC defaults to ovr decision function shape
ovr_SVC = LinearSVC(C=10**k,max_iter=50000).fit(X_train_lsi, train_dataset.
↪target)
ovr_prediction = ovr_SVC.predict(X_test_lsi)

metrics.plot_confusion_matrix(ovr_SVC, X_test_lsi, test_dataset.target)
print("One Vs. Rest SVM prediction stats:")
```

```
print(metrics.classification_report(test_dataset.target,ovr_prediction,digits=4))
```