

ECE ENGR 219 - Project 1
Classification Analysis on Textual Data

Boyuan He (UID 004791432)
Ganesha Durbha (UID 405291327)

January 20, 2021

Table of Contents

1	20 Newsgroups data set (Question 1)	2
2	Feature Extraction (Question 2)	2
3	Dimensionality Reduction (Question 3)	2
4	Classification	3
4.1	Support Vector Machines (Question 4)	3
4.2	Logistic Regression (Question 5)	5
4.3	Naive Bayes (Question 6)	7
4.4	Grid Search of Parameters (Question 7)	8
5	Word Embeddings (Question 8)	9
5.1	(Question 8.a)	9
5.2	(Question 8.b)	9
5.3	(Question 8.c)	9
5.4	(Question 8.d)	10
6	Classification using GLoVE features (Question 9)	10
6.1	(Question 9.a)	10
6.2	(Question 9.b)	10
6.3	(Question 10)	11
7	Visualization of clusters (Question 11)	12
8	Multiclass Classification (Question 12)	13
9	Appendix	15

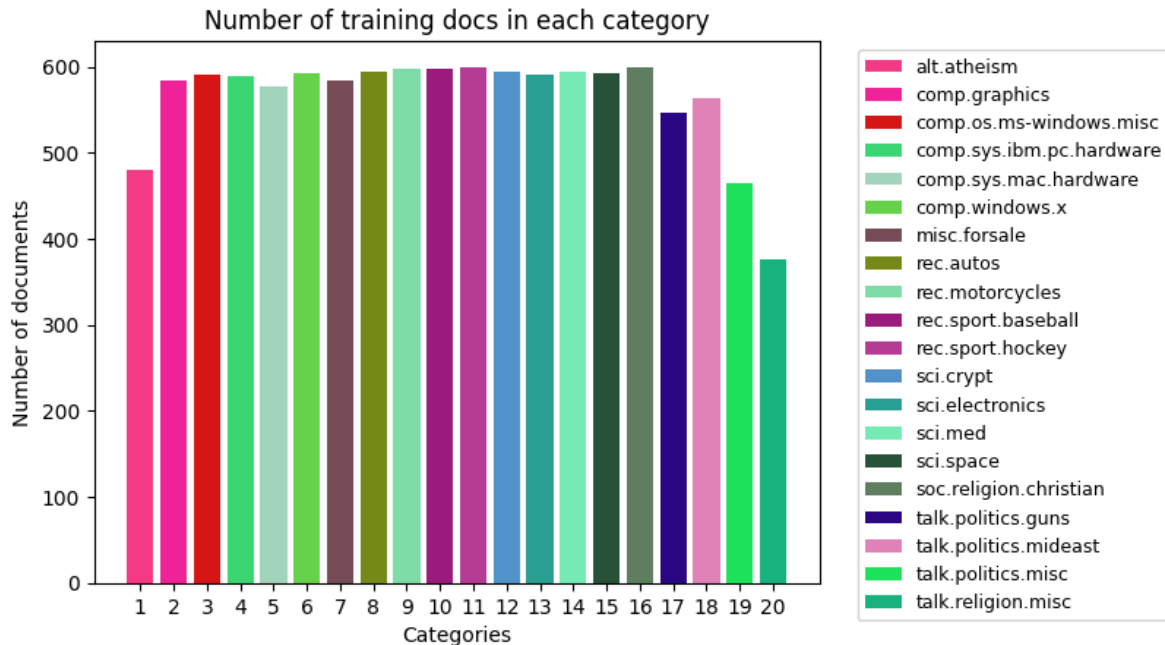
List of Figures

1	Distribution of training examples across various categories	2
2	ROC curve and confusion matrix of hard and soft SVM	4
3	ROC curve and confusion matrix of the best SVM	5
4	ROC curve and confusion matrix of Logistic Regression	6
5	ROC curve and confusion matrix of Naive Bayes	8
6	Evaluation on using GLoVE embeddings using SVM	11
7	Effect of number of components on accuracy	12
8	UMPA representation of each document	12
9	UMPA representation of random vectors	13
10	Confusion matrix of multiclass classification	13

1. 20 Newsgroups data set (Question 1)

The number of training documents for each of the 20 categories of the *20 Newsgroups* dataset are approximately evenly distributed. This is necessary to ensure that the model sees the entire input space during training.

Figure 1: Distribution of training examples across various categories



2. Feature Extraction (Question 2)

To classify the documents, it is necessary to pre-process data and extract useful information from each document. The common words in the English language appear in most of the documents and do not help us to discriminate between the categories. A collection of such common words are available in the scikit-learn library as stopwords, which were removed from all documents. Lemmatization aims to convert all words into their root and also considers the parts of speech. This step is necessary to map representations of similar words in a family to the base form. On removing stopwords, numeric data, performing lemmatization, and considering words that appeared in at least 3 documents, we get the size of the TF - IDF matrices of the train and test subsets as

$$\text{Size of TF-IDF}_{\text{train}} = (4732, 16319)$$

$$\text{Size of TF-IDF}_{\text{test}} = (3150, 11243)$$

3. Dimensionality Reduction (Question 3)

After the feature extraction step, we need to reduce the dimension of the TF-IDF matrix in order to improve the performance of learning algorithms. There are two choices for dimension reduction – LSI and NMF.

Latent Semantic Indexing (LSI) representation uses singular value decomposition (SVD) to retrieve information from high dimension matrix; Non-negative matrix factorization (NMF) tries to approximate the high dimensional matrix with a low dimensional one using non-negative weights of certain learnt topic vectors. Here, we tried both approaches.

For both NMF and LSI, we map each document to 50 dimensional vector, and get a resulting vector of shape 4732×50 . We also calculated the error margin defined by $\|X - WH\|_{\mathcal{F}}^2$ for NMF and $\|X - U_k \Sigma_k V_k^T\|_{\mathcal{F}}^2$ for LSI. The LSI error result is 4106.9935 and the NMF error result is 4143.1639 .

$$\begin{aligned}\|X - WH\|_{\mathcal{F}}^2 &= 4143.1639 \\ \|X - U_k \Sigma_k V_k^T\|_{\mathcal{F}}^2 &= 4106.9935\end{aligned}$$

A possible reason for the slightly better capture by the LSI representation is that the SVD method does not impose non-negative weighing condition on the linear combination of certain bases vectors. It can therefore better capture the matrix. However, the NMF method does come close because it is approximating a TF-IDF matrix whose meaning can be interpreted as positive combination of some "topical" vectors.

4. Classification

In this section, we trained and tested multiple classification algorithms including Support Vector Machines, Logistic Regression, and Naive Bayes. In the end, we performed grid search the classification algorithms and related parameters, and report the ones that perform the best. One thing to note here is that although we have 8 categories of data, we classify them into two groups: Computer Technology and Recreational Activity. Specifically, we modified the data label to map the categories of comp.graphics, comp.os.ms-windows.misc, comp.sys.ibm.pc.hardware and comp.sys.mac.hardware to Computer Technology and the categories of rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey to Recreational Activity.

4.1. Support Vector Machines (Question 4)

We use the the dimension-reduced data from LSI to trained two models of linear SVMs, one with trade off parameter γ set to 1000 (hard margin) and one set to 0.0001 (soft margin). And for each classifier, we used our testing data to plot receiver operating characteristic (ROC) curve and calculate the accuracy, recall, precision and F-1 score. Figure 2 show the ROC curve and confusion matrix of hard and soft margin SVM; table 1 shows the accuracy (normalized), recall, precision and F-1 score of both SVM classifiers.

Figure 2: ROC curve and confusion matrix of hard and soft SVM

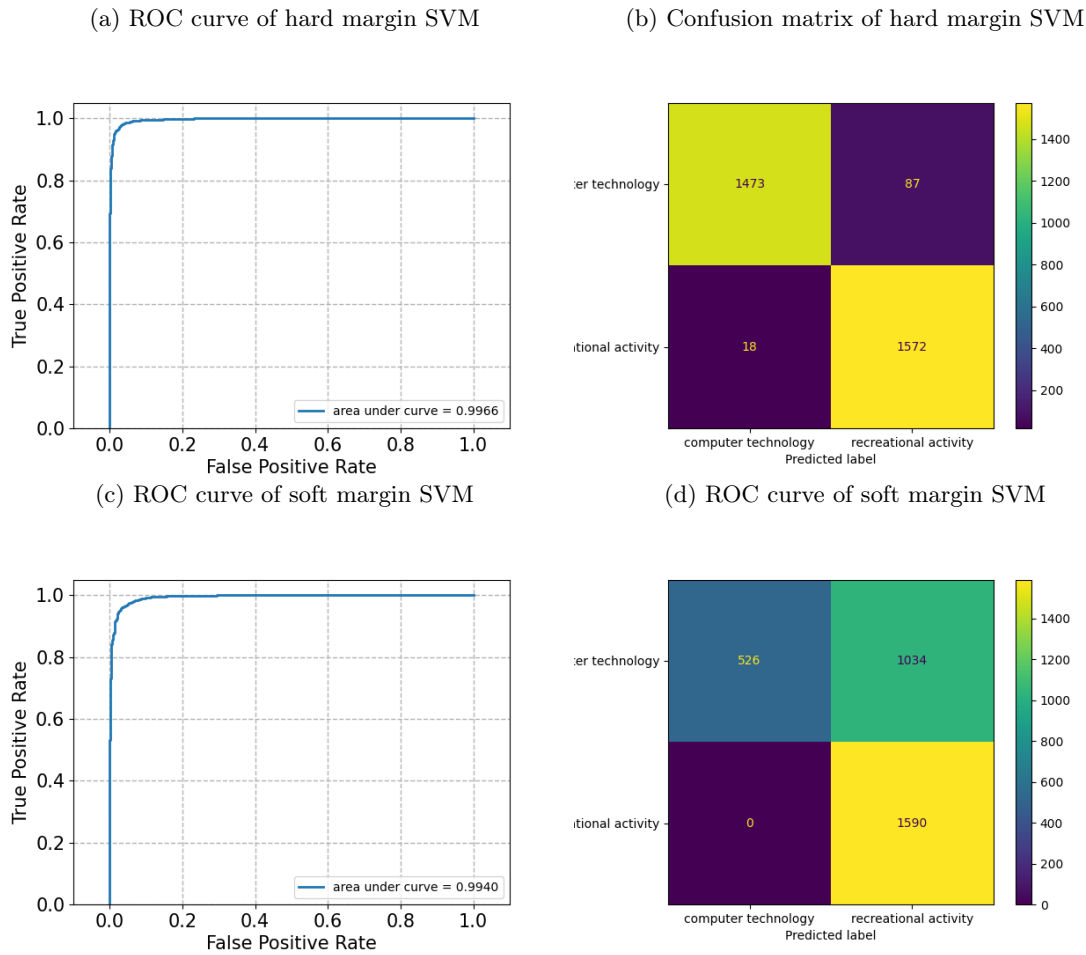


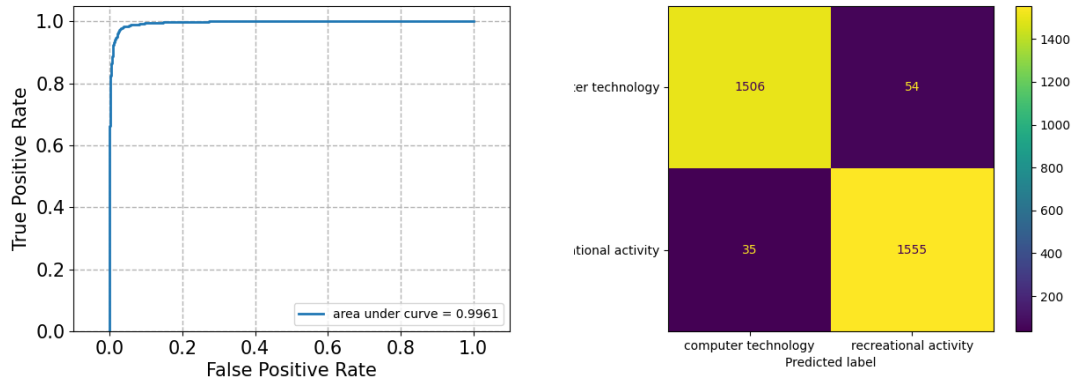
Table 1: accuracy, recall, precision and F-1 score of both SVM classifiers

	accuracy	recall	precision	F-1 score
hard SVM	0.9667	0.9887	0.9476	0.9677
soft SVM	0.6717	1.0000	0.6059	0.7546

The hard margin SVM classifier performs better because in the case of the soft margin classifier the model could not learn the correct decision boundary due to the slack provided to training data points close to the boundary. This is reflected in the poor accuracy, precision, and F-1 score of the soft margin SVM classifier. The high ROC curve looks good for the soft margin SVM. This does not conflict with the poor accuracy and precision metrics of the model. It is because the area under the ROC curve is a measure of how well the positive examples are correctly classified. A very high ROC could result from the model having a greater tendency to classify any data point as a positive class. Thus the positive examples are correctly classified (as shown by perfect recall score), but there is a large number of false positives too (since negative examples were also predicted to be positive) which rightly led to very poor accuracy, precision, and F-1 metrics.

To choose the best trade off parameter γ , we used 5-fold cross validation to test for γ ranging between 1000 and 0.001. The best γ value was found to be 1. The ROC curve and confusion matrix of SVM with $\gamma = 1$ are shown in figures 3. Table 2 shows its accuracy, recall, precision and F-1 score. The SVM with $\gamma = 1$ out performs soft SVM by a great extent, especially considering the accuracy, precision and F1-score. The hard margin SVM is quite effective and has performance very similar to the best case SVM with $\gamma = 1$.

Figure 3: ROC curve and confusion matrix of the best SVM

(a) ROC curve of SVM with $\gamma = 1$ (b) Confusion matrix of SVM with $\gamma = 1$ Table 2: accuracy (normalized), recall, precision and F-1 score of SVM with $\gamma = 1$

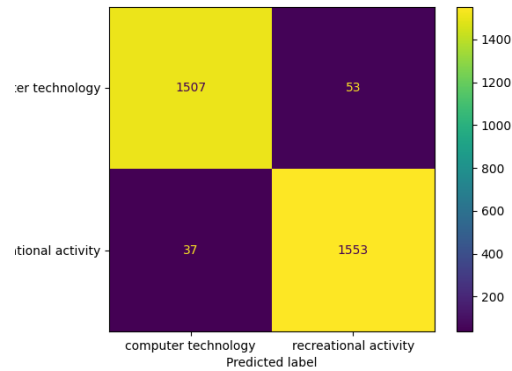
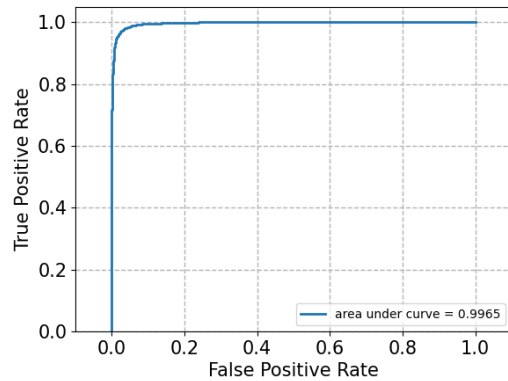
	accuracy	recall	precision	F-1 score
SVM $\gamma = 1$	0.9717	0.9780	0.9664	0.9721

4.2. Logistic Regression (Question 5)

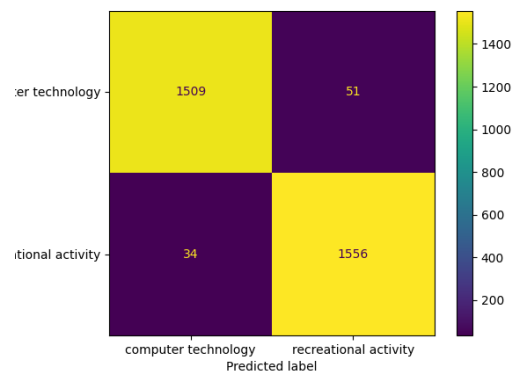
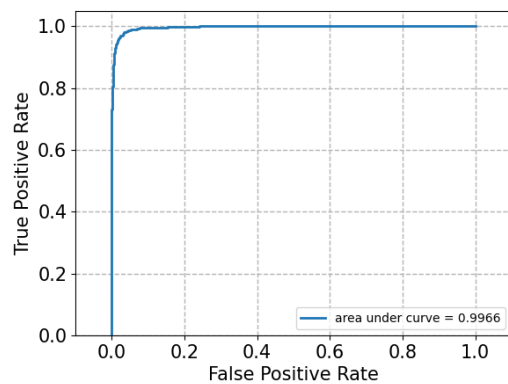
A logistic classifier was trained without regularization, with L1 regularization, and with L2 regularization. A 5-fold cross-validation was performed to find the best regularization strength in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$ for both L1 and L2 regularization. The results are shown in the next page. Figure 4 shows the ROC curve and confusion matrix with different normalizations, table 3 shows the accuracy, recall, precision and F-1 score of logistic regressions, and table 4 shows the best regularization strength we used.

Figure 4: ROC curve and confusion matrix of Logistic Regression

(a) ROC curve of logistic classifier without regularization (b) Confusion matrix of logistic classifier without regularization



(c) ROC curve of logistic classifier with l1 regularization (d) Confusion matrix of logistic classifier with l1 regularization



(e) ROC curve of logistic classifier with l2 regularization (f) Confusion matrix of logistic classifier with l2 regularization

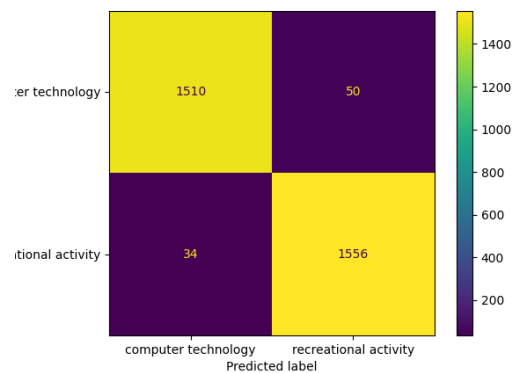
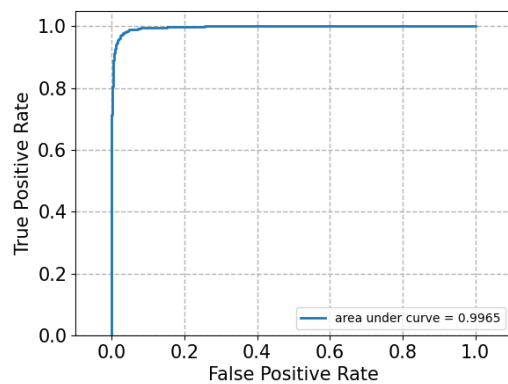


Table 3: accuracy (normalized), recall, precision and F-1 score of logistic classifier

	accuracy	recall	precision	F-1 score
logistic classifier without regularization	0.9714	0.9767	0.9670	0.9718
logistic classifier with l1 regularization	0.9730	0.9786	0.9683	0.9734
logistic classifier with l2 regularization	0.9733	0.9786	0.9689	0.9737

Table 4: Best regularization strength (k)

L1	best k = 0.1
L2	best k = 0.01

Thus the methods, ranked in order of performance from the highest to the lowest are - L2 regularization, L1 regularization, no regularization. The reasons are explored below.

Regularization is a shrinkage method whose main purpose is to prevent overfitting of the training data by shrinking the size of the coefficients of features that are not important, thereby improving the test classification. L1 regularization aims to reduce the 1-norm of the coefficient vector to be learnt, while L2 regularization aims to reduce the 2-norm of the coefficient vector to be learnt. Due to this definition, on increasing the regularization strength, L1 regularization would shift all coefficient estimates by the same amount towards zero, and the sufficiently small coefficients are shrunk all the way to zero. However, in an L2 regularization, coefficient estimates are proportionally shrunk towards zero but are not entirely eliminated.

The higher performance of L2 regularization in our classifier shows that all features in the training set are important in this classification task and the ability of L1 regularization to eliminate lesser weighted features led to some loss of information which impacted the classifier's performance. L2 regularization should be used when we desire to use information from all input features, however weighing them by their importance. L2 regularization can be used if we believe that there are features that do not have useful information and could be entirely eliminated.

Logistic regression and linear SVM - both determine a linear decision boundary to classify data points. The difference is in the cost considered to determine the boundary. Logistic regression is based on probabilistic representation and the boundary is learnt in such a way that all data points tend to be the farthest they could from the boundary. However, SVM is based on geometric representation and intends to maximize the distance (the margin) of only the closest data points (called support vectors) from the decision boundary. It is not much affected by data points far from the margin. Thus, logistic regression is more sensitive to outliers while SVMs are good for making generalizations.

4.3. Naive Bayes (Question 6)

We also used naive Bayes classifier for the classification problem. Specifically, we chose the GaussianNB to run the training and testing. Figure 5 shows the ROC curve and confusion matrix. Table 5 shows its accuracy, recall, precision and F-1 score.

Figure 5: ROC curve and confusion matrix of Naive Bayes

(a) ROC curve of Naive Bayes

(b) Confusion matrix of Naive Bayes

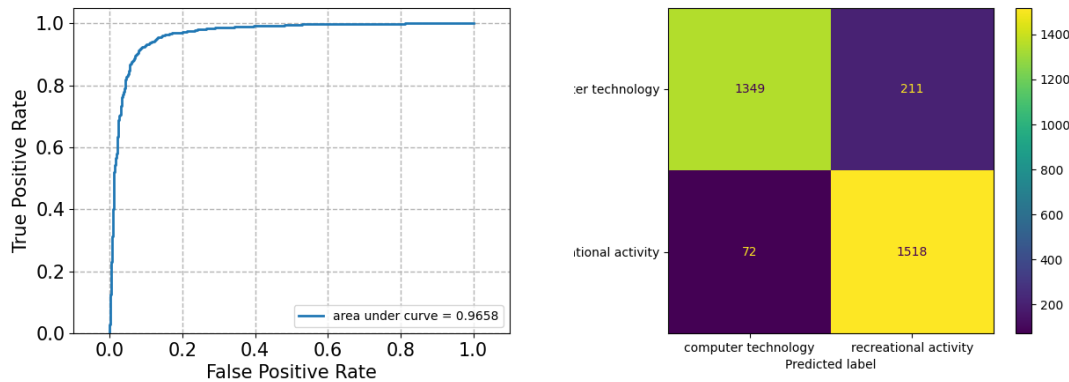


Table 5: accuracy (normalized), recall, precision and F-1 score of Naive Bayes

	accuracy	recall	precision	F-1 score
Naive Bayes	0.9101	0.9547	0.8779	0.9147

4.4. Grid Search of Parameters (Question 7)

To choose the best performing algorithm and parameters in all steps (data loading, feature extraction, dimension reduction, classification), we run grid search with parameters listed in table 6 below, which can also be found in project manual.

Table 6: all options considered for grid search

Procedure	Options
Loading Data	remove “headers” and “footers” vs not
Feature Extraction	min df = 3 vs 5; use lemmatization vs not
Dimensionality Reduction	LSI vs NMF
Classifier	SVM vs L1 vs L2 logistic regression vs GaussianNB

There are 64 combinations and the detailed grid search result can be found in grid_serch.csv. Here we only show the top 5 performing combinations with most related parameters in table 7.

Table 7: Top combinations from grid search

header footer	classifier	C	dimension reduction	min_df	penalty	test score	rank
yes	LogisticRegression()	10	TruncatedSVD()	5	l1	0.976542592	1
yes	LogisticRegression()	10	TruncatedSVD()	3	l1	0.976120429	2
yes	LogisticRegression()	100	TruncatedSVD()	3	l2	0.976120206	3
yes	LogisticRegression()	100	TruncatedSVD()	5	l2	0.97611976	4
yes	LogisticRegression()	10	TruncatedSVD()	5	l1	0.975275656	5

Note* : C is the inverse of regularization strength.

As we can see, the best combination is logistic regression classifier, regularization strength $k = 1/10$, using LSI representation for dimension reduction, minimum document frequency set to 5, with L1 penalty, and including the header and footer information. But generally speaking, we find that the classifier has a big influence on the accuracy: Logistic regression classifiers take the top 7 spots and the worst one is still at position 41, SVM is in the middle, and can beat only few logistic regression classifiers. Naive Bayes

performs the worst, and even the best Naive Bayes is at position 45. The dimension reduction method is also very important: LSI would outperform NMF in most cases with logistic regression and SVM, but NMF is actually better for Naive Bayes.

5. Word Embeddings (Question 8)

5.1. (Question 8.a)

The embeddings are trained on the ratio of co-occurrence probabilities because it provides a clear distinction between two target words. As noticed in the GloVe paper, the ratio of co-occurrence probabilities $\frac{P_{ik}}{P_{jk}}$ of two target words i, j with a context word k , gives how well the word k would fit in the context of word i . If the ratio is much higher than 1, then k is specific to word i . Similarly, if the ratio $\frac{P_{ik}}{P_{jk}}$ is much less than 1, then k is specific to word j .

The ratio also serves well to cancel out noisy information by returning a value around 1 when the context word k is irrelevant or is related to both i and j .

5.2. (Question 8.b)

The same vector would be returned when queried for the GloVe embedding of the word *running*. Ideally, the returned vector contains information related to both styles of usage of the word *running*. This is because, in the training stage, each word is trained considering the context window. If samples in the training set had similar usages of the word *running*, all such contexts of the word shall be learnt by the model.

5.3. (Question 8.c)

It is expected that $\|\text{GloVe}(\text{"queen"}) - \text{GloVe}(\text{"king"}) - \text{GloVe}(\text{"wife"}) + \text{GloVe}(\text{"husband"})\|_2 = 0$. The vector space is generated in a manner that the dimensions are meaningful with respect to the language rules. Using such vector representations, the model is expected to give *queen - king = wife - husband*, which makes *queen - king - wife + husband = 0*.

This would make $\|\text{GloVe}(\text{"queen"}) - \text{GloVe}(\text{"king"}) - \text{GloVe}(\text{"wife"}) + \text{GloVe}(\text{"husband"})\|_2 = 0$

The actual values on using GloVe embeddings are:

$$\|\text{GloVe}(\text{"queen"}) - \text{GloVe}(\text{"king"}) - \text{GloVe}(\text{"wife"}) + \text{GloVe}(\text{"husband"})\|_2 = 6.165$$

$$\|\text{GloVe}(\text{"queen"}) - \text{GloVe}(\text{"king"})\|_2 = 5.966$$

$$\|\text{GloVe}(\text{"wife"}) - \text{GloVe}(\text{"husband"})\|_2 = 3.152$$

While the analogy $\|\text{GloVe}(\text{"queen"}) - \text{GloVe}(\text{"king"}) - \text{GloVe}(\text{"wife"}) + \text{GloVe}(\text{"husband"})\|_2$ had a 2 - norm 6.165, replacing one of the words in the expression with an irrelevant word like *mango* resulted in higher 2 - norm values between 9.2 and 10.6. For practical implementation, the GloVe paper used the dot product of word embeddings in calculating and minimizing training cost. The experiments on word analogy used cosine similarity to select the closest vector. The same exercise would provide more assurance than comparing the 2-norm of the vectors. The cosine similarity between the vectors *queen - king* and *wife - husband* was found to be 0.200. If any one word in the analogy is replaced by an irrelevant word like *mango*,

we get cosine similarities 0.062, 0.010, 0.013 which are less than 0.200. Training accuracy constraints and limitations of the hypothesis are among other reasons for not getting the expected output.

5.4. (Question 8.d)

Lemmatizing would be preferred over stemming. Since the GloVe embeddings hold contextual information, words like *fly* and *flying* are represented by different vectors. Therefore, stemming is not helpful as it could reduce both words to *fly*, losing the information in the *flying* vector. At times, words would be trimmed to the extent that they do not have meaning, and hence cannot be mapped into a GloVe embedding.

6. Classification using GloVe features (Question 9)

6.1. (Question 9.a)

Stop words were removed to avoid words which repeat often. Then, each document string was split into lines. Words from the *Subject*, and *Keywords* lines were collected. To consider information from these words, the mean of their GloVe vector representations was taken and normalized into a unit vector. This normalized mean vector represented the document during training. Considering documents which had words in *Subject* and *Keywords* lines that could be mapped into GloVe embeddings, there were 4476 training documents and 3020 testing documents.

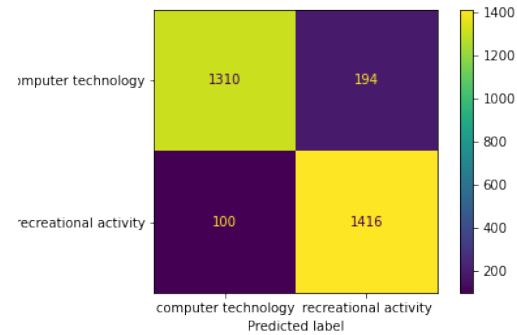
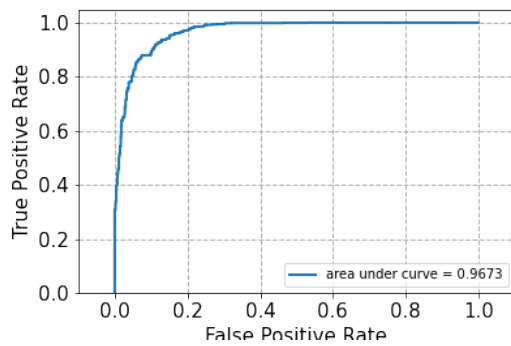
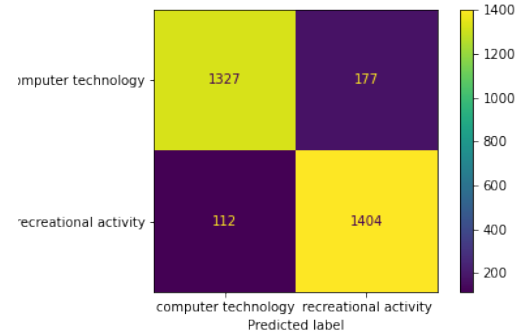
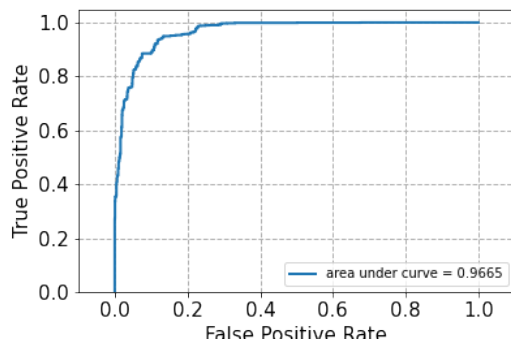
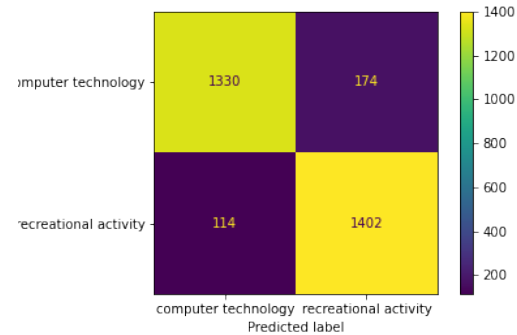
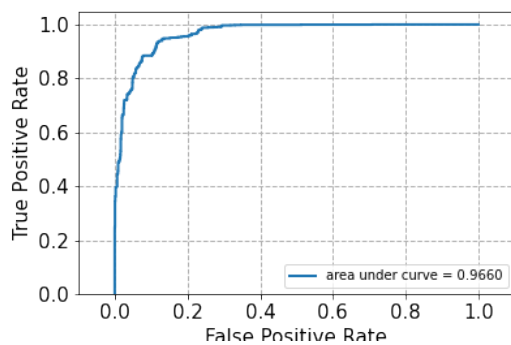
6.2. (Question 9.b)

Since the GloVe embeddings are 300- D vectors, the data matrix for our classification has a size (4476, 300). Dimensionality was reduced to 100 components using the SVD method. The training matrix thus had a size (4476, 100). Support vector machines were used for the classification with $\gamma = 0.1$. The evaluation scores are:

Table 8: accuracy, recall, precision and F-1 score on training with GloVe embeddings of documents

Using GloVe	accuracy	recall	precision	F-1 score
SVM $\gamma = 0.1$	0.90265	0.93404	0.8795	0.90595
SVM $\gamma = 1$	0.9043	0.92612	0.88805	0.90668
SVM $\gamma = 10$	0.90464	0.9248	0.88959	0.90686

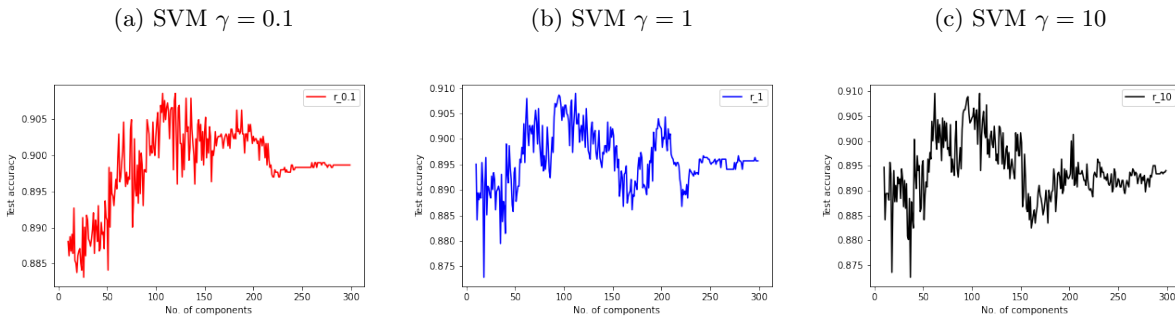
Figure 6: Evaluation on using GLoVe embeddings using SVM

(a) roc curve SVM $\gamma = 0.1$ (b) confusion matrix SVM $\gamma = 0.1$ (c) roc curve SVM $\gamma = 1$ (d) confusion matrix SVM $\gamma = 1$ (e) roc curve SVM $\gamma = 10$ (f) confusion matrix SVM $\gamma = 10$ 

6.3. (Question 10)

To study the effect of the number of components on the test accuracy, the number of components were varied from 10 to 300 in integer steps. The resulting test accuracy was plotted for γ values taking 0.1, 1, 10.

Figure 7: Effect of number of components on accuracy



The trend observed is that at low number of components the classification accuracy is poor. This might be due to the fact that sufficient information was lost due to dimensionality reduction. As the number of components increases, the classification accuracy increases and reaches a maximum between $n_{comp} = 100$ and $n_{comp} = 125$. At higher dimensions, the accuracy decreases. This might be attributed to the fact that more components result in capturing the noise in training data by a larger extent.

7. Visualization of clusters (Question 11)

The reasonable metrics obtained justify the model learnt using GloVe embeddings. Each document was represented by the normalized mean of GloVe embeddings of its keywords and subject. The UMPA library learns about this manifold and generates a reduced two dimensional representation of each document. In the figure below, we observe two clusters of red (clustered slightly to the left) and violet (clustered slightly to the right), showing documents of the two categories. However when we try to represent random 300-D vectors, they are rightly scattered all over the plot as they do not have any inherent clustering.

Figure 8: UMPA representation of each document

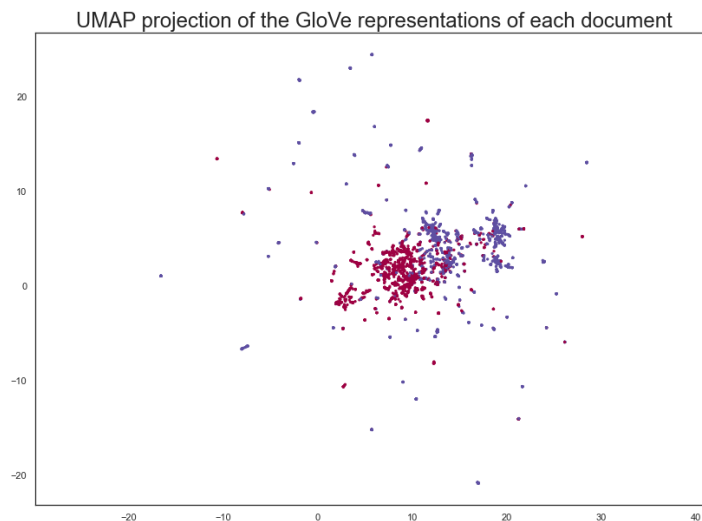
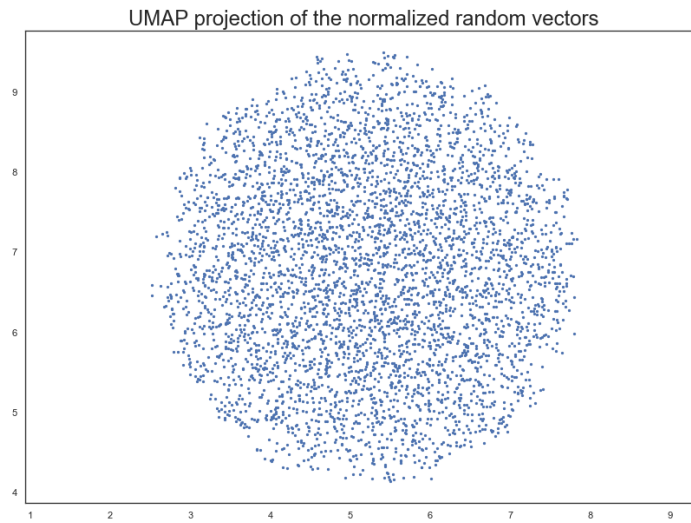


Figure 9: UMPA representation of random vectors

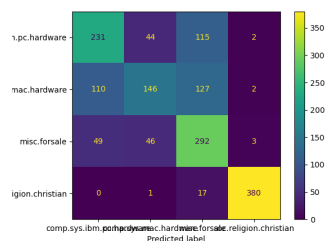


8. Multiclass Classification (Question 12)

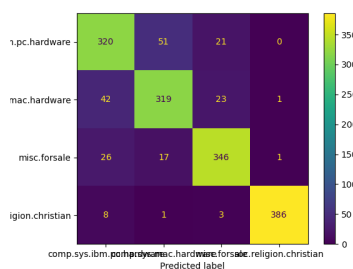
In the end, we try to perform multi-class classification. Specifically, we choose data from comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, misc.forsale and soc.religion.christian categories; used Naive Bayes classification and multiclass SVM classification (one vs one and one vs rest). The confusion matrix of each classification method is shown in the graph 10, and the accuracy, recall, precision and F-1 score are shown in table 9 below

Figure 10: Confusion matrix of multiclass classification

(a) Naive Bayes classification



(b) Multiclass SVM classification (one vs one)



(c) Multiclass SVM classification (one vs res)

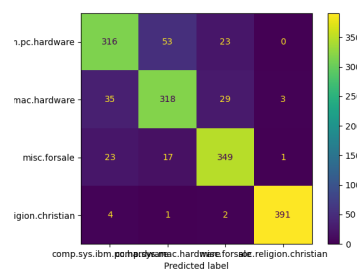


Table 9: accuracy, recall, precision and F-1 score of the classifiers

	accuracy	recall	precision	F-1 score
Naive Bayes	0.6703	0.6703	0.6703	0.6703
SVM (one vs one)	0.8760	0.8760	0.8760	0.8760
SVM (one vs rest)	0.8780	0.8780	0.8780	0.8780

9. Appendix

Below are the code for entire project1. Due to decisions made during development, code is divided into three groups/files. The first group contains code for questions 1,2 and 3; the second group contains code for questions 4, 5, 6, 7 and 12; the third group contains code for questions 8, 9, 10, 11.

```

1 import nltk
2 import random
3 from sklearn.datasets import fetch_20newsgroups
4 import numpy as np
5 from matplotlib import pyplot as plt
6 from nltk import pos_tag
7 from sklearn.metrics import auc
8 from sklearn.pipeline import Pipeline
9 from sklearn.svm import LinearSVC
10 from sklearn.decomposition import NMF
11 from sklearn.decomposition import TruncatedSVD
12 from sklearn.feature_extraction.text import TfidfTransformer
13 from sklearn.feature_extraction.text import CountVectorizer
14 from sklearn.metrics import roc_curve
15
16 ## Question 1
17 twenty_train = fetch_20newsgroups(
18     subset='train', shuffle=True, random_state=42)
19
20 # doc_number = list(map(lambda category: len(fetch_20newsgroups(
21 #     subset='train', categories=[category]).data), twenty_train.target_names))
22 ind, counts = np.unique(twenty_train.target, return_counts=True)
23
24 bar_colors = []
25 for i in range(len(counts)):
26     rgb = np.random.rand(3,)
27     bar_colors.append(rgb)
28
29 p1 = plt.bar(ind, counts, width=0.8, color=bar_colors)
30 plt.ylabel('Number of documents')
31 plt.xlabel('Categories')
32 plt.title('Number of training docs in each category')
33 plt.xticks(ind, np.arange(1, len(counts)+1))
34 plt.legend(p1, twenty_train.target_names, bbox_to_anchor=(
35     1.04, 1), loc="upper left", prop={'size': 9})
36 plt.savefig("q1.G.png", bbox_inches="tight")
37
38 # Question 2
39 np.random.seed(42)
40 random.seed(42)
41
42 categories = ['comp.graphics', 'comp.os.ms-windows.misc',
43              'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
44              'rec.autos', 'rec.motorcycles',
45              'rec.sport.baseball', 'rec.sport.hockey']
46
47 target_map = {
48     0: 0,
49     1: 0,
50     2: 0,
51     3: 0,
52     4: 1,
53     5: 1,
54     6: 1,
55     7: 1,
56 }
57 twenty_train = fetch_20newsgroups(
58     subset='train', categories=categories, shuffle=True, random_state=None)
59
60 twenty_test = fetch_20newsgroups(
61     subset='test', categories=categories, shuffle=True, random_state=None)
62
63
64 nltk.download('wordnet')
65 nltk.download('punkt')
66 nltk.download('averaged_perceptron_tagger')
67 wnl = nltk.wordnet.WordNetLemmatizer()
68
69
70 def penn2morphy(penntag):
71     """ Converts Penn Treebank tags to WordNet. """
72     morphy_tag = {'NN': 'n', 'JJ': 'a',
73                  'VB': 'v', 'RB': 'r'}
74     try:
75         return morphy_tag[penntag[:2]]
76     except:

```



```

77         return 'n'
78
79
80 def lemmatize_sent(list_word):
81     # Text input is list of strings, returns array of lowercased strings(words).
82     return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
83             for word, tag in pos_tag(list_word)]
84
85
86 # overwrite analyzer with callable function:
87 vectorizer1 = CountVectorizer(stop_words='english')
88 stop_words = vectorizer1.get_stop_words()
89 # print(stop_words)
90 print("Number of stop words are:")
91 print(len(stop_words))
92
93 analyzer = CountVectorizer().build_analyzer()
94
95
96 def stem_rmv_punc(doc):
97     return [word for word in lemmatize_sent(analyzer(doc)) if word not in stop_words and not word.isdigit()]
98     # Stopwords and checking for digits must be done here itself.
99     # Because, when a callable is passed to analyzer in CoutVectorizer, it extracts features from THE RAW, UNPROCESSED INPUT.
100
101
102 count_vect = CountVectorizer(min_df=3, analyzer=stem_rmv_punc)
103 # G - I do not think stop_words = 'english' is necessary here because of the comments mentioned in stem_rmv_punc
104
105 X_train_counts = count_vect.fit_transform(twenty_train.data)
106 print("Shape of X_train_counts is")
107 print(X_train_counts.shape)
108
109 tfidf_transformer = TfidfTransformer()
110 X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
111 print("Shape of X_train_tfidf is")
112 print(X_train_tfidf.shape)
113
114 # Question 3
115
116 # LSI
117
118 svd = TruncatedSVD(n_components=50)
119 X_train_reduced = svd.fit_transform(X_train_tfidf)
120 print(X_train_reduced.shape)
121 svd.components_
122 print(np.sum(np.array(X_train_tfidf - X_train_reduced.dot(svd.components_))**2))
123 # NMF
124
125 model = NMF(n_components=50, init='random', random_state=0)
126 W_train = model.fit_transform(X_train_tfidf)
127
128 print(W_train.shape)
129
130 # calculate ||X - WHk||^2
131 H = model.components_
132 H.shape
133 NMF_result = np.sum(np.array(X_train_tfidf - W_train.dot(H))**2)
134 print(NMF_result)

```

Listing 1: Code for Question 1, 2, 3

```

1 from sklearn.multiclass import OneVsOneClassifier
2 from sklearn.multiclass import OneVsRestClassifier
3 import pandas as pd
4 from sklearn.model_selection import GridSearchCV
5 from joblib import Memory
6 from shutil import rmtree
7 from tempfile import mkdtemp
8 from sklearn.model_selection import cross_val_score
9 from sklearn.metrics import plot_confusion_matrix
10 from sklearn.base import BaseEstimator, TransformerMixin
11 from sklearn.naive_bayes import GaussianNB
12 import nltk
13 import random
14 from sklearn.datasets import fetch_20newsgroups
15 import numpy as np
16 from matplotlib import pyplot as plt
17 from nltk import pos_tag
18 from sklearn.metrics import auc
19 from sklearn.pipeline import Pipeline
20 from sklearn.svm import LinearSVC
21 from sklearn.decomposition import NMF
22 from sklearn.decomposition import TruncatedSVD
23 from sklearn.feature_extraction.text import TfidfTransformer
24 from sklearn.feature_extraction.text import CountVectorizer

```

```

25 from sklearn.metrics import roc_curve
26 from sklearn.linear_model import LogisticRegression
27 from statistics import mean
28 from sklearn import metrics
29
30 np.random.seed(42)
31 random.seed(42)
32
33 SCORE_TEMPLATE = """
34 {}
35     accuracy (normalized): {}
36     recall: {}
37     precision: {}
38     f1 {}
39
40 """
41
42 TARGET_MAP = {
43     0: 0,
44     1: 0,
45     2: 0,
46     3: 0,
47     4: 1,
48     5: 1,
49     6: 1,
50     7: 1,
51 }
52
53 result_file = open("results.txt", "w")
54
55 categories = ['comp.graphics', 'comp.os.ms-windows.misc',
56              'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
57              'rec.autos', 'rec.motorcycles',
58              'rec.sport.baseball', 'rec.sport.hockey']
59
60
61 # load data and convert to 2 category
62 twenty_train = fetch_20newsgroups(
63     subset='train', categories=categories, shuffle=True, random_state=None)
64 twenty_train.target = np.array(
65     list(map(lambda label: TARGET_MAP[label], twenty_train.target)))
66
67 twenty_test = fetch_20newsgroups(
68     subset='test', categories=categories, shuffle=True, random_state=None)
69 twenty_test.target = np.array(
70     list(map(lambda label: TARGET_MAP[label], twenty_test.target)))
71
72
73 # feature extraction
74 nltk.download('wordnet')
75 nltk.download('punkt')
76 nltk.download('averaged_perceptron_tagger')
77 wnl = nltk.wordnet.WordNetLemmatizer()
78
79 # configure stop words
80 stop_words = CountVectorizer(stop_words='english').get_stop_words()
81
82
83 def penn2morphy(pennntag):
84     # Converts Penn Treebank tags to WordNet
85     morphy_tag = {'NN': 'n', 'JJ': 'a',
86                  'VB': 'v', 'RB': 'r'}
87     if len(pennntag) > 1 and pennntag[:2] in morphy_tag:
88         return morphy_tag[pennntag[:2]]
89     else:
90         return "n"
91
92
93 def lemmatize_sent(list_word):
94     # Text input is list of strings, returns array of lowercased strings(words).
95     return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
96             for word, tag in pos_tag(list_word)]
97
98
99 def stem_rmv_punc(doc):
100     return [word for word in lemmatize_sent(CountVectorizer().build_analyzer()(doc)) if word not in stop_words and not word.isdigit()]
101
102
103 def plot_roc(fpr, tpr):
104     # roc curve plot
105     fig, ax = plt.subplots()
106
107     roc_auc = auc(fpr, tpr)
108
109     ax.plot(fpr, tpr, lw=2, label='area under curve = %0.4f' % roc_auc)
110

```

```

111 ax.grid(color='0.7', linestyle='--', linewidth=1)
112
113 ax.set_xlim([-0.1, 1.1])
114 ax.set_ylim([0.0, 1.05])
115 ax.set_xlabel('False Positive Rate', fontsize=15)
116 ax.set_ylabel('True Positive Rate', fontsize=15)
117
118 ax.legend(loc="lower right")
119
120 for label in ax.get_xticklabels()+ax.get_yticklabels():
121     label.set_fontsize(15)
122
123 fig.savefig(figure_name)
124
125
126 def fit_predict_and_plot_roc(figure_name, pipe, train_data, train_label, test_data, test_label):
127     # fit, predict and plot roc
128     pipe.fit(train_data, train_label)
129
130     if hasattr(pipe, 'decision_function'):
131         prob_score = pipe.decision_function(test_data)
132         fpr, tpr, _ = roc_curve(test_label, prob_score)
133     else:
134         prob_score = pipe.predict_proba(test_data)
135         fpr, tpr, _ = roc_curve(test_label, prob_score[:, 1])
136
137     plot_roc(figure_name, fpr, tpr)
138
139
140 # preprocessing
141 LSI_preprocessing_pipeline = Pipeline([
142     ('vect', CountVectorizer(min_df=3, analyzer=stem_rmv_punc)),
143     ('tfidf', TfidfTransformer()),
144     ('reduce_dim', TruncatedSVD(n_components=50)),
145 ])
146
147 reduced_train = LSI_preprocessing_pipeline.fit_transform(twenty_train.data)
148 reduced_test = LSI_preprocessing_pipeline.transform(twenty_test.data)
149
150
151 def calculate_socre(clf, title, average="binary"):
152     pred = clf.predict(reduced_test)
153     accuracy_score = metrics.accuracy_score(
154         twenty_test.target, pred, normalize=True)
155     recall_score = metrics.recall_score(
156         twenty_test.target, pred, average=average)
157     precision_score = metrics.precision_score(
158         twenty_test.target, pred, average=average)
159     f1_score = metrics.f1_score(
160         twenty_test.target, pred, average=average)
161
162     result_file.write(SCORE_TEMPLATE.format(
163         title, accuracy_score, recall_score, precision_score, f1_score))
164
165
166 # Q4
167 result_file.write("===== Question 4 =====\n")
168
169 # r = 1000
170 svm1000_pipeline = Pipeline([
171     ('clf', LinearSVC(C=1000)),
172 ])
173
174 # roc curve
175 fit_predict_and_plot_roc('q4_roc_r1000.png', svm1000_pipeline, reduced_train,
176     twenty_train.target, reduced_test, twenty_test.target)
177
178 # confusion matrix
179 plot_confusion_matrix(svm1000_pipeline['clf'], reduced_test, twenty_test.target, display_labels=[
180     "computer technology", "recreational activity"])
181 plt.savefig('q4_confusion_matrix_r1000.png')
182
183 calculate_socre(svm1000_pipeline['clf'], "linear SVMs with C = 1000")
184
185 # r = 0.0001
186 svm00001_pipeline = Pipeline([
187     ('clf', LinearSVC(C=0.0001)),
188 ])
189
190 # roc curve
191 fit_predict_and_plot_roc('q4_roc_r0001.png', svm00001_pipeline, reduced_train,
192     twenty_train.target, reduced_test, twenty_test.target)
193
194 # confusion matrix
195 plot_confusion_matrix(svm00001_pipeline['clf'], reduced_test, twenty_test.target, display_labels=[
196     "computer technology", "recreational activity"])

```

```

197
198 plt.savefig('q4_confusion_matrix_r0001.png')
199
200 calculate_socre(svm00001_pipeline["clf"], "linear SVMs with C = 0.0001")
201
202
203 # cross validation
204 tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
205 for c in tradeoff:
206     clf = LinearSVC(C=c)
207     fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
208     tradeoff[c] = mean(fold_score)
209
210 # pipeline for the best tradeoff value
211 svm_C = max(tradeoff, key=tradeoff.get)
212 svm_best_pipeline = Pipeline([
213     ('clf', LinearSVC(C=svm_C)),
214 ])
215
216 # roc curve
217 fit_predict_and_plot_roc('q4_roc_best.png', svm_best_pipeline, reduced_train,
218                          twenty_train.target, reduced_test, twenty_test.target)
219
220 # confusion matrix
221 plot_confusion_matrix(svm_best_pipeline["clf"], reduced_test, twenty_test.target, display_labels=[
222     "computer technology", "recreational activity"])
223 plt.savefig('q4_confusion_matrix_best.png')
224
225 calculate_socre(svm_best_pipeline["clf"],
226                "linear SVMs with best C = {}".format(svm_C))
227
228 # Q5
229 result_file.write("===== Question 5 =====\n")
230
231 # Note: the instruction "you may need to come up with some way to approximate this if you use sklearn.linear model.LogisticRegression"
232 # suggest using L2 and very large C to cancel the effect of regularization. But it seem to be outdated, now there is an None option.
233
234 # Logistic regression
235 logistic_pipeline = Pipeline([
236     ('clf', LogisticRegression(penalty="none")),
237 ])
238
239 # roc curve
240 fit_predict_and_plot_roc('q5_roc_none.png', logistic_pipeline, reduced_train,
241                          twenty_train.target, reduced_test, twenty_test.target)
242
243 # confusion matrix
244 plot_confusion_matrix(logistic_pipeline["clf"], reduced_test, twenty_test.target, display_labels=[
245     "computer technology", "recreational activity"])
246 plt.savefig('q5_confusion_matrix_none.png')
247
248 calculate_socre(logistic_pipeline["clf"], "logistic classifier")
249
250 # cross validation on L1
251 tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
252 for c in tradeoff:
253     clf = LogisticRegression(C=c, penalty="l1", solver="liblinear")
254     fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
255     tradeoff[c] = mean(fold_score)
256
257 # pipeline for the best L1
258 log1_C = max(tradeoff, key=tradeoff.get)
259 log_bestl1_pipeline = Pipeline([
260     ('clf', LogisticRegression(C=log1_C, penalty="l1", solver="liblinear")),
261 ])
262
263 # roc curve
264 fit_predict_and_plot_roc('q5_roc_bestL1.png', log_bestl1_pipeline, reduced_train,
265                          twenty_train.target, reduced_test, twenty_test.target)
266
267 # confusion matrix
268 plot_confusion_matrix(log_bestl1_pipeline["clf"], reduced_test, twenty_test.target, display_labels=[
269     "computer technology", "recreational activity"])
270 plt.savefig('q5_confusion_matrix_bestL1.png')
271
272 calculate_socre(
273     log_bestl1_pipeline["clf"], "logistic classifier with L1, best C = {}".format(log1_C))
274
275 # cross validation on L2
276 tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
277 for c in tradeoff:
278     clf = LogisticRegression(C=c, penalty="l2")
279     fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
280     tradeoff[c] = mean(fold_score)
281
282 # pipeline for the best L2
283 log2_C = max(tradeoff, key=tradeoff.get)
284 log_bestl2_pipeline = Pipeline([

```

```

283 ('clf', LogisticRegression(C=log2_C, penalty="l2")),
284 ])
285
286 # roc curve
287 fit_predict_and_plot_roc('q5_roc_bestL2.png', log_bestl2_pipeline, reduced_train,
288                          twenty_train.target, reduced_test, twenty_test.target)
289
290 # confusion matrix
291 plot_confusion_matrix(log_bestl2_pipeline["clf"], reduced_test, twenty_test.target, display_labels=[
292     "computer technology", "recreational activity"])
293 plt.savefig('q5_confusion_matrix_bestL2.png')
294
295 calculate_socre(
296     log_bestl2_pipeline["clf"], "logistic classifier with L2, best    = {}".format(log2_C))
297
298
299 # Q6
300 result_file.write("===== Question 6 =====\n")
301
302 # Naive Bayes
303 bayes_pipeline = Pipeline([
304     ('clf', GaussianNB()),
305 ])
306
307 # roc curve
308 fit_predict_and_plot_roc('q6_roc.png', bayes_pipeline, reduced_train,
309                          twenty_train.target, reduced_test, twenty_test.target)
310
311 # confusion matrix
312 plot_confusion_matrix(bayes_pipeline["clf"], reduced_test, twenty_test.target, display_labels=[
313     "computer technology", "recreational activity"])
314 plt.savefig('q6_confusion_matrix.png')
315
316 calculate_socre(bayes_pipeline["clf"], "naive bayes classifier")
317
318
319 # Q7
320
321 def stem_rmv_punc_nonlemmatize(doc):
322     return (word for word in CountVectorizer().build_analyzer()(doc) if word not in stop_words and not word.isdigit())
323
324
325 # load removed header footer data and convert to 2 category
326 twenty_train_nohf = fetch_20newsgroups(
327     subset='train', categories=categories, shuffle=True, random_state=None, remove=('headers', 'footers'))
328 twenty_train_nohf.target = np.array(
329     list(map(lambda label: TARGET_MAP[label], twenty_train_nohf.target)))
330
331 cachedir = mkdtemp()
332 memory = Memory(cachedir=cachedir, verbose=10)
333
334 grid_pipeline = Pipeline([
335     ('vect', CountVectorizer()),
336     ('tfidf', TfidfTransformer()),
337     ('reduce_dim', TruncatedSVD(n_components=50)),
338     ('clf', GaussianNB()),
339 ],
340    memory=memory
341 )
342
343 param_grid = [
344     {
345         'vect__min_df': [3, 5],
346         'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
347         'reduce_dim': [TruncatedSVD(), NMF()],
348         'reduce_dim__n_components': [50],
349         'clf': [LinearSVC(C=svm_C)],
350         'clf__C': [svm_C]
351     },
352     {
353         'vect__min_df': [3, 5],
354         'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
355         'reduce_dim': [TruncatedSVD(), NMF()],
356         'reduce_dim__n_components': [50],
357         'clf': [LogisticRegression()],
358         'clf__C': [log1_C],
359         'clf__penalty': ["l1"],
360         'clf__solver': ["liblinear"]
361     },
362     {
363         'vect__min_df': [3, 5],
364         'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
365         'reduce_dim': [TruncatedSVD(), NMF()],
366         'reduce_dim__n_components': [50],
367         'clf': [LogisticRegression()],
368         'clf__C': [log2_C],

```

```

369         'clf__penalty': ["l2"],
370     },
371     {
372         'vect__min_df': [3, 5],
373         'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
374         'reduce_dim': [TruncatedSVD(), NMF()],
375         'reduce_dim__n_components': [50],
376         'clf': [GaussianNB()],
377     },
378 ]
379
380 grid = GridSearchCV(grid_pipeline, cv=5, n_jobs=1,
381                     param_grid=param_grid, scoring='accuracy')
382 grid.fit(twenty_train.data, twenty_train.target)
383 df = pd.DataFrame(grid.cv_results_)
384 df.to_csv('q7_header_footer.csv')
385
386 grid.fit(twenty_train_nohf.data, twenty_train_nohf.target)
387 df = pd.DataFrame(grid.cv_results_)
388 df.to_csv('q7_no_header_footer.csv')
389 rmtree(cachedir)
390
391 # Q12
392 result_file.write("===== Question 12 =====\n")
393
394 categories = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
395              'misc.forsale', 'soc.religion.christian']
396
397 twenty_train = fetch_20newsgroups(
398     subset='train', categories=categories, shuffle=True, random_state=None)
399
400 twenty_test = fetch_20newsgroups(
401     subset='test', categories=categories, shuffle=True, random_state=None)
402
403 reduced_train = LSI_preprocessing_pipeline.fit_transform(twenty_train.data)
404 reduced_test = LSI_preprocessing_pipeline.transform(twenty_test.data)
405
406 # Naive Bayes
407 clf = GaussianNB().fit(reduced_train, twenty_train.target)
408
409 # confusion matrix
410 plot_confusion_matrix(clf, reduced_test, twenty_test.target,
411                       display_labels=categories)
412 plt.savefig('q12_confusion_matrix_NB.png')
413
414 calculate_socre(clf, "naive bayes classifier", average='micro')
415
416 # multiclass SVM classification
417 # one vs one
418 clf = OneVsOneClassifier(LinearSVC()).fit(reduced_train, twenty_train.target)
419 plot_confusion_matrix(clf, reduced_test, twenty_test.target,
420                       display_labels=categories)
421 plt.savefig('q12_confusion_matrix_svmito1.png')
422
423 calculate_socre(
424     clf, "multiclass SVM classification with one VS one", average='micro')
425
426 # one vs rest
427 clf = OneVsRestClassifier(LinearSVC()).fit(reduced_train, twenty_train.target)
428 plot_confusion_matrix(clf, reduced_test, twenty_test.target,
429                       display_labels=categories)
430 plt.savefig('q12_confusion_matrix_svmitores.png')
431
432 calculate_socre(
433     clf, "multiclass SVM classification with one VS rest", average='micro')
434
435 result_file.close()

```

Listing 2: Code for Question 4, 5, 6, 7, 12

```

1  #!/usr/bin/env python
2  # coding: utf-8
3
4  # In[2]:
5
6
7  print((37.0))
8
9
10 # In[3]:
11
12
13 #twenty_train.data[0]
14 #twenty_train.target[0]

```

```

15 import numpy as np
16 embeddings_dict = {}
17 dimension_of_glove = 300
18 with open("C:/Users/sriva/Desktop/glove.6B/glove.6B.300d.txt", 'r', encoding="utf8") as f:
19     for line in f:
20         values = line.split()
21         word = values[0]
22         vector = np.asarray(values[1:], "float32")
23         embeddings_dict[word] = vector
24
25
26 # In[30]:
27
28
29 glove_keys = embeddings_dict.keys()
30 print(type(glove_keys))
31 print(len(glove_keys))
32
33
34 # In[31]:
35
36
37 print(type(embeddings_dict['computer']))
38
39
40 # In[32]:
41
42
43 from sklearn.model_selection import cross_val_score
44 from sklearn.metrics import plot_confusion_matrix
45 from sklearn.base import BaseEstimator, TransformerMixin
46 from sklearn.naive_bayes import GaussianNB
47 import nltk
48 import random
49 from sklearn.datasets import fetch_20newsgroups
50 import numpy as np
51 from matplotlib import pyplot as plt
52 from nltk import pos_tag
53 from sklearn.metrics import auc
54 from sklearn.pipeline import Pipeline
55 from sklearn.svm import LinearSVC
56 from sklearn.decomposition import NMF
57 from sklearn.decomposition import TruncatedSVD
58 from sklearn.feature_extraction.text import TfidfTransformer
59 from sklearn.feature_extraction.text import CountVectorizer
60 from sklearn.metrics import roc_curve
61 from sklearn.linear_model import LogisticRegression
62 from statistics import mean
63
64 np.random.seed(42)
65 random.seed(42)
66
67 TARGET_MAP = {
68     0: 0,
69     1: 0,
70     2: 0,
71     3: 0,
72     4: 1,
73     5: 1,
74     6: 1,
75     7: 1,
76 }
77 categories = ['comp.graphics', 'comp.os.ms-windows.misc',
78              'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
79              'rec.autos', 'rec.motorcycles',
80              'rec.sport.baseball', 'rec.sport.hockey']
81
82 # load data and convert to 2 category
83 twenty_train = fetch_20newsgroups(
84     subset='train', categories=categories, shuffle=True, random_state=None)
85 twenty_train.target = np.array(
86     list(map(lambda label: TARGET_MAP[label], twenty_train.target)))
87 twenty_test = fetch_20newsgroups(
88     subset='test', categories=categories, shuffle=True, random_state=None)
89 twenty_test.target = np.array(
90     list(map(lambda label: TARGET_MAP[label], twenty_test.target)))
91
92 # In[6]:
93
94
95 def plot_roc(figure_name, fpr, tpr):
96     # roc curve plot
97     fig, ax = plt.subplots()
98     roc_auc = auc(fpr, tpr)
99     ax.plot(fpr, tpr, lw=2, label='area under curve = %0.4f' % roc_auc)
100     ax.grid(color='0.7', linestyle='--', linewidth=1)

```

```

101 ax.set_xlim([-0.1, 1.1])
102 ax.set_ylim([0.0, 1.05])
103 ax.set_xlabel('False Positive Rate', fontsize=15)
104 ax.set_ylabel('True Positive Rate', fontsize=15)
105 ax.legend(loc="lower right")
106 for label in ax.get_xticklabels()+ax.get_yticklabels():
107     label.set_fontsize(15)
108 fig.savefig(figure_name)
109 def fit_predict_and_plot_roc(figure_name, pipe, train_data, train_label, test_data, test_label):
110     # fit, predict and plot roc
111     pipe.fit(train_data, train_label)
112     if hasattr(pipe, 'decision_function'):
113         prob_score = pipe.decision_function(test_data)
114         fpr, tpr, _ = roc_curve(test_label, prob_score)
115     else:
116         prob_score = pipe.predict_proba(test_data)
117         fpr, tpr, _ = roc_curve(test_label, prob_score[:, 1])
118     plot_roc(figure_name, fpr, tpr)
119
120
121 # In[33]:
122
123
124 # configure stop words
125 stop_words = CountVectorizer(stop_words='english').get_stop_words()
126 def rmv_stp_digits(doc):
127     return (word for word in CountVectorizer().build_analyzer()(doc) if word not in stop_words and not word.isdigit())
128
129
130 # In[36]:
131
132
133 def get_glove(sen):
134     #Gives the average of golve representation of words in subject and keyword lines.
135     glove_vec = np.zeros((300,))
136     sen = sen.lower()
137     count = 0
138     for item in sen.split("\n"):
139         ww = item.split()
140         if len(ww)>0 and ww[0] in ["subject:", "keywords:"]:
141             for i in ww:
142                 if i in glove_keys and i not in stop_words:
143                     #print(i)
144                     glove_vec = glove_vec + embeddings_dict[i]
145                     count = count + 1
146
147     if count == 0:
148         return (glove_vec)
149     else:
150         avg_vec = glove_vec/count
151         return (avg_vec/np.linalg.norm(avg_vec))
152
153
154 # In[37]:
155
156
157 Xtrain_glove = np.zeros((len(twenty_train.target),300))
158
159 for ind in range(len(twenty_train.target)):
160     Xtrain_glove[ind] = get_glove(twenty_train.data[ind])
161
162 print(Xtrain_glove.shape)
163
164 #Remove documents that gave 0 mean glove vectors
165 twenty_train.target = twenty_train.target[~np.all(Xtrain_glove == 0, axis = 1)]
166 Xtrain_glove = Xtrain_glove[~np.all(Xtrain_glove == 0, axis = 1)]
167 print(Xtrain_glove.shape)
168
169
170 # In[38]:
171
172
173 #Modifying test set data too:
174 Xtest_glove = np.zeros((len(twenty_test.target),300))
175
176 for ind in range(len(twenty_test.target)):
177     Xtest_glove[ind] = get_glove(twenty_test.data[ind])
178
179 print(Xtest_glove.shape)
180
181 #Remove documents that gave 0 mean glove vectors
182 twenty_test.target = twenty_test.target[~np.all(Xtest_glove == 0, axis = 1)]
183 Xtest_glove = Xtest_glove[~np.all(Xtest_glove == 0, axis = 1)]
184 print(Xtest_glove.shape)
185
186

```



```

187 # In[13]:
188
189
190 svd = TruncatedSVD(n_components=100, random_state=0)#100 components chosen
191 Xtrain_reduced = svd.fit_transform(Xtrain_glove)
192 print(Xtrain_reduced.shape)
193 Xtest_reduced = svd.transform(Xtest_glove)
194 print(Xtest_reduced.shape)
195
196
197 # In[14]:
198
199
200 def apply_svm(r_val, train_data, train_labels, test_data, test_labels):
201
202     svm_pipeline = Pipeline([
203         ('clf', LinearSVC(C=r_val)),
204     ])
205     # roc curve
206     fit_predict_and_plot_roc("q9_roc_r{}.png".format(r_val), svm_pipeline, train_data,
207                             train_labels, test_data, test_labels)
208     # confusion matrix
209     plot_confusion_matrix(svm_pipeline["clf"], test_data, test_labels, display_labels=[
210         "computer technology", "recreational activity"])
211     plt.savefig('q9_confusion_matrix_r{}.png'.format(r_val))
212
213
214 # In[15]:
215
216
217 apply_svm(r_val = 1000, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
218         target)
219
220 # In[16]:
221
222
223 apply_svm(r_val = 0.1, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target
224         )
225
226 # In[17]:
227
228
229 apply_svm(r_val = 100, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target
230         )
231 apply_svm(r_val = 10, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target)
232 apply_svm(r_val = 1, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target)
233
234 # In[18]:
235
236
237 apply_svm(r_val = 0.01, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
238         target)
239 apply_svm(r_val = 0.001, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
240         target)
241
242 # In[19]:
243
244
245 def apply_svm_ncomp(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):
246
247     svd = TruncatedSVD(n_components= n_comp, random_state=0)#100 components chosen
248     train_data = svd.fit_transform(Xtrain)
249     #print(Xtrain_reduced.shape)
250     test_data = svd.transform(Xtest)
251     #print(Xtest_reduced.shape)
252
253     svm_pipeline = Pipeline([
254         ('clf', LinearSVC(C=r_val)),
255     ])
256     # roc curve
257     fit_predict_and_plot_roc("q10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipeline, train_data,
258                             train_labels, test_data, test_labels)
259     # confusion matrix
260     plot_confusion_matrix(svm_pipeline["clf"], test_data, test_labels, display_labels=[
261         "computer technology", "recreational activity"])
262     plt.savefig('q10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))
263
264 # In[20]:
265
266
267

```

```

268 apply_svm_ncomp(n_comp = 50, r_val = 0.001, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
269
270
271 # In[21]:
272
273
274 apply_svm_ncomp(n_comp = 50, r_val = 0.01, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
275 apply_svm_ncomp(n_comp = 50, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test
    .target)
276 apply_svm_ncomp(n_comp = 50, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test.
    target)
277 apply_svm_ncomp(n_comp = 50, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test.
    target)
278 apply_svm_ncomp(n_comp = 50, r_val = 100, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test
    .target)
279 apply_svm_ncomp(n_comp = 50, r_val = 1000, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
280 #=====
281 apply_svm_ncomp(n_comp = 175, r_val = 0.001, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
282 apply_svm_ncomp(n_comp = 175, r_val = 0.01, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
283 apply_svm_ncomp(n_comp = 175, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
284 apply_svm_ncomp(n_comp = 175, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test.
    target)
285 apply_svm_ncomp(n_comp = 175, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test
    .target)
286 apply_svm_ncomp(n_comp = 175, r_val = 100, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
287 apply_svm_ncomp(n_comp = 175, r_val = 1000, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
288 #=====
289 apply_svm_ncomp(n_comp = 250, r_val = 0.001, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
290 apply_svm_ncomp(n_comp = 250, r_val = 0.01, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
291 apply_svm_ncomp(n_comp = 250, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
292 apply_svm_ncomp(n_comp = 250, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test.
    target)
293 apply_svm_ncomp(n_comp = 250, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test
    .target)
294 apply_svm_ncomp(n_comp = 250, r_val = 100, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
295 apply_svm_ncomp(n_comp = 250, r_val = 1000, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
296 #=====
297 apply_svm_ncomp(n_comp = 300, r_val = 0.001, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
298 apply_svm_ncomp(n_comp = 300, r_val = 0.01, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
299 apply_svm_ncomp(n_comp = 300, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
300 apply_svm_ncomp(n_comp = 300, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test.
    target)
301 apply_svm_ncomp(n_comp = 300, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels = twenty_test
    .target)
302 apply_svm_ncomp(n_comp = 300, r_val = 100, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
303 apply_svm_ncomp(n_comp = 300, r_val = 1000, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
    twenty_test.target)
304
305
306 # In[39]:
307
308
309 from sklearn import metrics
310 def svm_ncomp_score(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):
311
312     svd = TruncatedSVD(n_components=n_comp, random_state=0)#100 components chosen
313     train_data = svd.fit_transform(Xtrain)
314     #print(Xtrain_reduced.shape)
315     test_data = svd.transform(Xtest)
316     #print(Xtest_reduced.shape)
317
318     svm_pipeline = Pipeline([
319         ('clf', LinearSVC(C=r_val)),
320     ])
321
322     # roc curve
323     fit_predict_and_plot_roc("tq10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipeline, train_data,
324         train_labels, test_data, test_labels)
325     # confusion matrix

```

```

326 plot_confusion_matrix(svm_pipeline["clf"], test_data, test_labels, display_labels=[
327     "computer technology", "recreational activity"])
328 plt.savefig('tq10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))
329
330 pred_y = svm_pipeline["clf"].predict(test_data)
331 return([ metrics.accuracy_score(test_labels, pred_y, normalize = True), metrics.recall_score(test_labels, pred_y), metrics.precision_score(
332     test_labels, pred_y), metrics.f1_score(test_labels, pred_y)])
333
334 # In[23]:
335
336
337 svm_ncomp_score(n_comp = 100, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
338     twenty_test.target)
339
340 # In[40]:
341
342
343 from sklearn import metrics
344 def svm_score(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):
345
346     svd = TruncatedSVD(n_components= n_comp, random_state=0)#100 components chosen
347     train_data = svd.fit_transform(Xtrain)
348     #print(Xtrain_reduced.shape)
349     test_data = svd.transform(Xtest)
350     #print(Xtest_reduced.shape)
351
352     #svm_pipeline = Pipeline([
353         #('clf', LinearSVC(C=r_val)),
354     #])
355
356     clf = LinearSVC(C=r_val).fit(train_data, train_labels)
357
358     # roc curve
359     #fit_predict_and_plot_roc("tq10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipeline, train_data,
360         #train_labels, test_data, test_labels)
361
362     # confusion matrix
363     #plot_confusion_matrix(svm_pipeline["clf"], test_data, test_labels, display_labels=[
364         #"computer technology", "recreational activity"])
365     #plt.savefig('tq10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))
366
367     pred_y = clf.predict(test_data)
368     acc_score = metrics.accuracy_score(test_labels, pred_y, normalize = True)
369     #return([ metrics.accuracy_score(test_labels, pred_y, normalize = True), metrics.recall_score(test_labels, pred_y), metrics.precision_score(
370         test_labels, pred_y), metrics.f1_score(test_labels, pred_y)])
371     return acc_score
372
373 # In[25]:
374
375
376 ncomp_list = np.arange(10,300,10)
377 acc_list = []
378 for num in ncomp_list:
379     acc_list.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
380         test_labels = twenty_test.target))
381
382 # In[26]:
383
384
385 plt.plot(ncomp_list, acc_list)#Test accuracy for r_val = 0. 1
386 plt.xlabel('No. of components')
387 plt.ylabel('Test accuracy')
388 plt.savefig('q11.png')
389
390
391 # In[46]:
392
393
394 ncomp_list = np.arange(10,300,5)
395 acc_listr01 = []
396 acc_listr1 = []
397 acc_listr10 = []
398 for num in ncomp_list:
399     acc_listr01.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
400         test_labels = twenty_test.target))
401     acc_listr1.append(svm_score(n_comp = num, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
402         test_labels = twenty_test.target))
403     acc_listr10.append(svm_score(n_comp = num, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
404         test_labels = twenty_test.target))
405 #plt.plot(ncomp_list, acc_list)#Test accuracy for r_val = 0. 1

```

```

405 #plt.xlabel('No. of components')
406 #plt.ylabel('Test accuracy')
407 #plt.savefig('q11_r{}.png'.format(r_val))
408
409
410 # In[47]:
411
412
413 plt.plot(ncomp_list, acc_listr01, 'r-',label = 'r_0.1')
414 plt.plot(ncomp_list, acc_listr1, 'b-',label = 'r_1')
415 plt.plot(ncomp_list, acc_listr10, 'k-',label = 'r_10')
416 plt.legend()
417 plt.xlabel('No. of components')
418 plt.ylabel('Test accuracy')
419 plt.savefig('q11_allr.png')
420
421
422 # In[48]:
423
424
425 ncomp_list = np.arange(10,300,1)
426 acc_listr01a = []
427 acc_listria = []
428 acc_listr10a = []
429 for num in ncomp_list:
430     acc_listr01a.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
431                                 test_labels = twenty_test.target))
432     acc_listria.append(svm_score(n_comp = num, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
433                                test_labels = twenty_test.target))
434     acc_listr10a.append(svm_score(n_comp = num, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
435                                 test_labels = twenty_test.target))
436
437 # In[53]:
438
439
440 fig = plt.figure()
441 plt.plot(ncomp_list, acc_listr01a, 'r-',label = 'r_0.1')
442 plt.legend()
443 plt.xlabel('No. of components')
444 plt.ylabel('Test accuracy')
445 plt.savefig('q10_r_0.1.png')
446 fig = plt.figure()
447 plt.plot(ncomp_list, acc_listria, 'b-',label = 'r_1')
448 plt.legend()
449 plt.xlabel('No. of components')
450 plt.ylabel('Test accuracy')
451 plt.savefig('q10_r_1.png')
452 fig = plt.figure()
453 plt.plot(ncomp_list, acc_listr10a, 'k-',label = 'r_10')
454 plt.legend()
455 plt.xlabel('No. of components')
456 plt.ylabel('Test accuracy')
457 plt.savefig('q10_r_10.png')
458
459
460 # In[27]:
461
462
463 #12
464 import numpy as np
465 from sklearn.datasets import load_digits
466 from sklearn.model_selection import train_test_split
467 from sklearn.preprocessing import StandardScaler
468 import matplotlib.pyplot as plt
469 import seaborn as sns
470 import pandas as pd
471 get_ipython().run_line_magic('matplotlib', 'inline')
472
473 sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})
474
475
476 # In[28]:
477
478
479 import umap
480 reducer = umap.UMAP()
481 scaled_data = StandardScaler().fit_transform(Xtrain_glove)
482 print(scaled_data.shape)
483 embedding = reducer.fit_transform(scaled_data)
484 print(embedding.shape)
485
486
487 # In[29]:

```

```

488
489
490 plt.scatter(embedding[:, 0], embedding[:, 1], c=twenty_train.target, cmap='Spectral', s=5)
491 plt.gca().set_aspect('equal', 'datalim')
492 #plt.colorbar(boundaries=[0,1])
493 plt.title('UMAP projection of the GloVe representations of each document', fontsize=24);
494 plt.savefig('q12_data.png')
495
496
497 # In[30]:
498
499
500 reducer = umap.UMAP()
501 embedding = reducer.fit_transform(Xtrain_glove)
502 print(embedding.shape)
503 plt.scatter(embedding[:, 0], embedding[:, 1], c=twenty_train.target, cmap='Spectral', s=5)
504 plt.gca().set_aspect('equal', 'datalim')
505 #plt.colorbar(boundaries=[0,1])
506 plt.title('UMAP projection of the GloVe representations of each document', fontsize=24);
507 plt.savefig('q12_data_noscaling.png')
508
509
510 # In[31]:
511
512
513 from random import gauss
514
515 def make_rand_vector(dims):
516     vec = [gauss(0, 1) for i in range(dims)]
517     mag = sum(x**2 for x in vec) ** .5
518     return [x/mag for x in vec]
519
520
521 # In[32]:
522
523
524 random_vects = np.zeros((4476, 300))
525 for i in range(4476):
526     random_vects[i] = make_rand_vector(300)
527
528
529 # In[33]:
530
531
532 reducer = umap.UMAP()
533 scaled_data = StandardScaler().fit_transform(random_vects)
534 print(scaled_data.shape)
535 embedding = reducer.fit_transform(scaled_data)
536 print(embedding.shape)
537 plt.scatter(embedding[:, 0], embedding[:, 1], cmap='Spectral', s=5)
538 plt.gca().set_aspect('equal', 'datalim')
539 #plt.colorbar(boundaries=[0,1])
540 plt.title('UMAP projection of the normalized random vectors', fontsize=24);
541 plt.savefig('q12_data_random.png')
542
543
544 # In[34]:
545
546
547 reducer = umap.UMAP()
548 embedding = reducer.fit_transform(random_vects)
549 print(embedding.shape)
550 plt.scatter(embedding[:, 0], embedding[:, 1], cmap='Spectral', s=5)
551 plt.gca().set_aspect('equal', 'datalim')
552 #plt.colorbar(boundaries=[0,1])
553 plt.title('UMAP projection of the normalized random vectors', fontsize=24);
554 plt.savefig('q12_data_random_Noscaling.png')
555
556 # In[162]:
557
558
559 aa = embeddings_dict["queen"] - embeddings_dict["king"] - embeddings_dict["wife"] + embeddings_dict["husband"]
560 print(np.linalg.norm(aa))
561
562
563 # In[164]:
564
565
566 aa1 = embeddings_dict["queen"] - embeddings_dict["king"]
567 aa2 = embeddings_dict["wife"] - embeddings_dict["husband"]
568 print(np.linalg.norm(aa1))
569 print(np.linalg.norm(aa2))
570
571
572 # In[4]:
573

```

```

574
575 aa = embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['husband']
576 bb = embeddings_dict['queen'] - embeddings_dict['king']
577 cc = embeddings_dict['wife'] - embeddings_dict['husband']
578 dd = [np.linalg.norm(embeddings_dict['queen']), np.linalg.norm(embeddings_dict['king']), np.linalg.norm(embeddings_dict['wife']), np.linalg.norm(
    embeddings_dict['husband'])]
579 print(np.linalg.norm(aa))
580 print(np.linalg.norm(bb))
581 print(np.linalg.norm(cc))
582 print(dd)
583
584
585 # In[14]:
586
587
588 from sklearn.metrics.pairwise import cosine_similarity
589 print(cosine_similarity([bb],[cc]))
590
591
592 # In[7]:
593
594
595 print(embeddings_dict['queen'][1:5])
596 print(embeddings_dict['king'][1:5])
597 print((embeddings_dict['queen']-embeddings_dict['king'])[1:5])
598
599
600 # In[18]:
601
602 print("****")
603 print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['mango']))
604 print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['mango'] + embeddings_dict['husband']))
605 print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['mango'] - embeddings_dict['wife'] + embeddings_dict['husband']))
606 print(np.linalg.norm(embeddings_dict['mango'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['husband']))
607
608 # In[10]:
609
610
611 ee = embeddings_dict['wife'] - embeddings_dict['mango']
612 ff = embeddings_dict['king'] - embeddings_dict['mango']
613 gg = embeddings_dict['queen'] - embeddings_dict['mango']
614 hh = embeddings_dict['husband'] - embeddings_dict['mango']
615 print(np.linalg.norm(ee))
616 print(np.linalg.norm(ff))
617 print(np.linalg.norm(gg))
618 print(np.linalg.norm(hh))
619
620
621 # In[17]:
622
623
624 ee = embeddings_dict['wife'] - embeddings_dict['mango']
625 ff = embeddings_dict['queen'] - embeddings_dict['king']
626 print(cosine_similarity([ee],[ff]))
627 ee = embeddings_dict['mango'] - embeddings_dict['husband']
628 ff = embeddings_dict['queen'] - embeddings_dict['king']
629 print(cosine_similarity([ee],[ff]))
630 ee = embeddings_dict['wife'] - embeddings_dict['husband']
631 ff = embeddings_dict['mango'] - embeddings_dict['king']
632 print(cosine_similarity([ee],[ff]))
633 ee = embeddings_dict['wife'] - embeddings_dict['husband']
634 ff = embeddings_dict['queen'] - embeddings_dict['mango']
635 print(cosine_similarity([ee],[ff]))

```

Listing 3: Code for Question 8, 9, 11