ECE ENGR 219 - Project 1
Classification Analysis on Textual Data

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

January 20, 2021

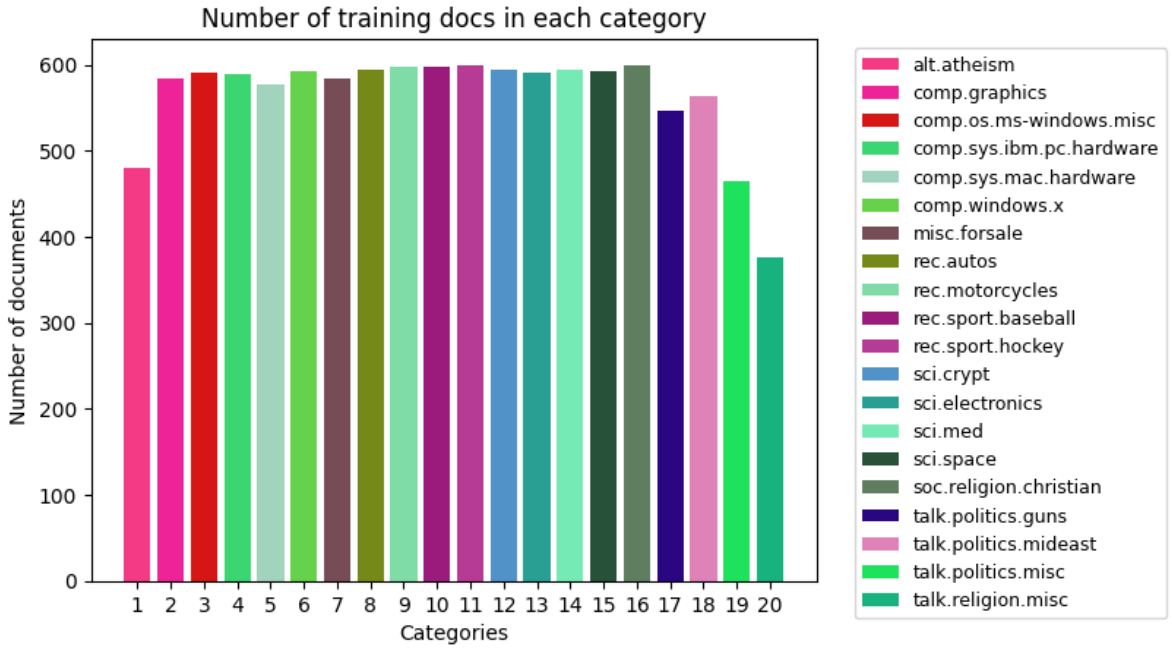
Table of Contents

List of Figures

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

$$\text{Size of TF-IDF}_{train} = (4732, 16319)$$
$$\text{Size of TF-IDF}_{test} = (3150, 11243)$$

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 $\left\|X - U_k \Sigma_k V_k^T\right\|_{\mathcal{F}}^2$ for LSI. The LSI error result is 4106.9935 and the NMF error result is 4143.1639 .

$$\|X - WH\|_{\mathcal{F}}^2 = 4143.1639$$
$$\left\|X - U_k \Sigma_k V_k^T\right\|_{\mathcal{F}}^2 = 4106.9935$$

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

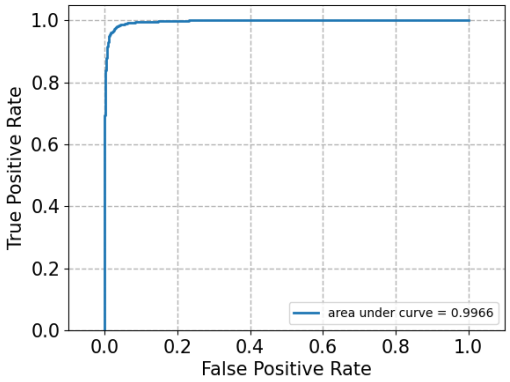
In this section, we trained and tested multiple classification algorithms including Support Vector Machines, Logistic Regression, and Naive Bayes. In the end, we performed grid search the classification algorithms and related parameters, and report the ones that perform the best. One thing to note here 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.

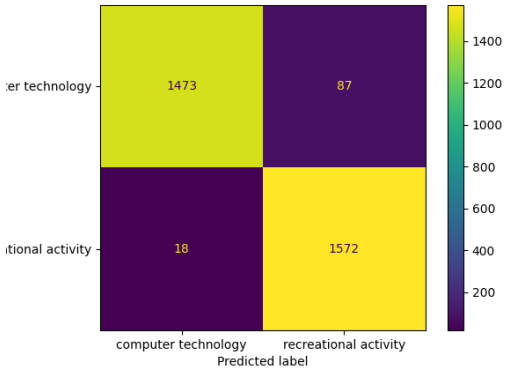
4.1. Support Vector Machines (Question 4)

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.

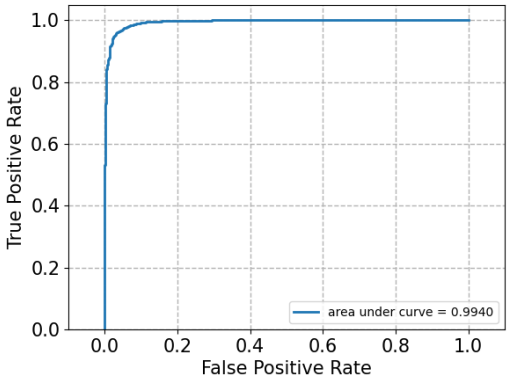
Figure 2: ROC curve and confusion matrix of hard and soft SVM

(a) ROC curve of hard margin SVM (b) Confusion matrix of hard margin SVM





(c) ROC curve of soft margin SVM (d) ROC curve of soft margin SVM



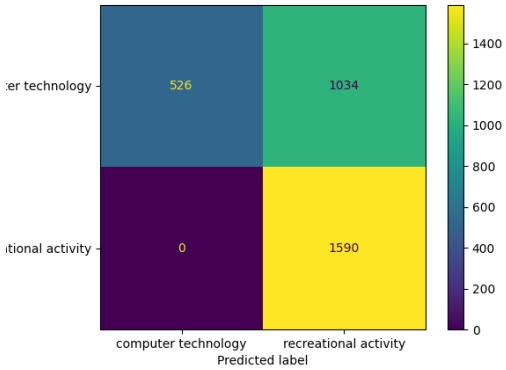


Table 1: accuracy, recall, precision and F-1 score of both SVM classifiers

	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 $\gamma = 1$

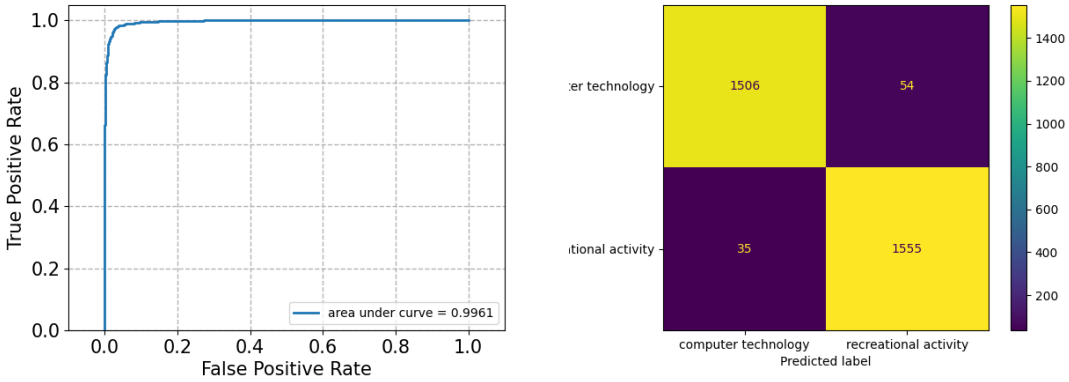


Table 2: accuracy (normalized), recall, precision and F-1 score of SVM with $\gamma = 1$

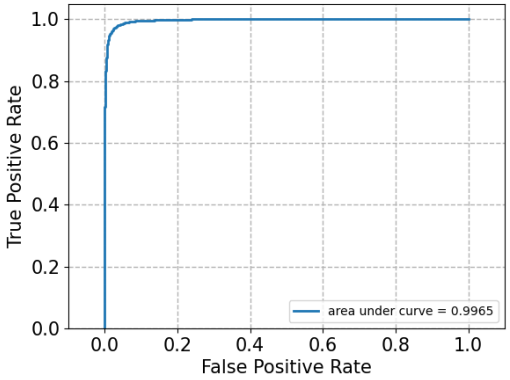
	accuracy	**recall**	**precision**	**F-1 score**
SVM $\gamma = 1$	**0.9717**	**0.9780**	**0.9664**	**0.9721**

4.2. Logistic Regression (Question 5)

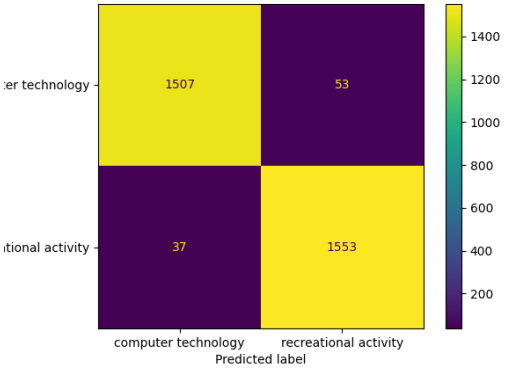
A logistic classifier was trained without regularization, with L1 regularization, and with L2 regularization. A 5-fold cross-validation was performed to find the best regularization strength in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$ for both L1 and L2 regularization. The results are show in the next page. Figure 4 shows the ROC curve and confusion matrix with different normalization's, 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

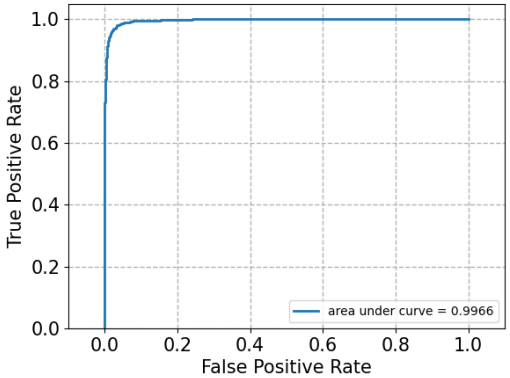
(a) ROC curve of logistic classifier without regularization



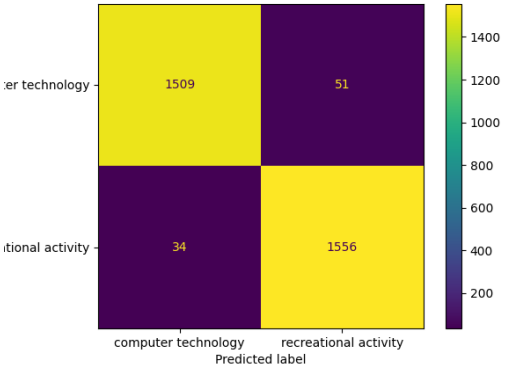
(b) Confusion matrix of logistic classifier without regularization



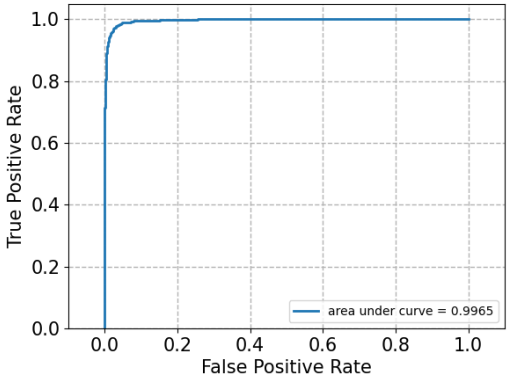
(c) ROC curve of logistic classifier with l1 regularization



(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

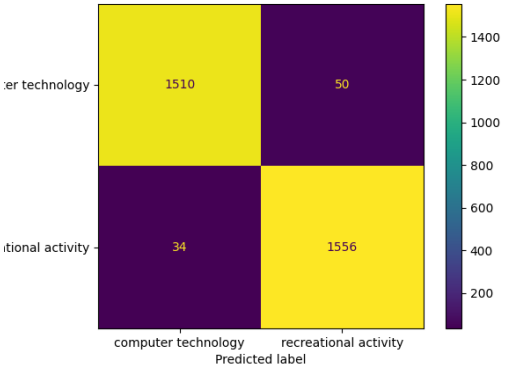


Table 3: accuracy (normalized), recall, precision and F-1 score of logistic classifier

	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 shrunken all the way to zero. However, in an L2 regularization, coefficient estimates are proportionally shrunken 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. Figures 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

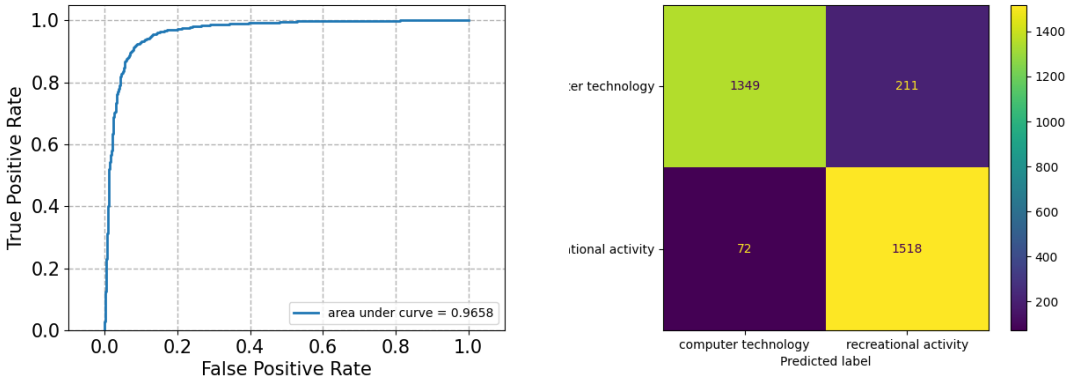


Table 5: accuracy (normalized), recall, precision and F-1 score of Naive Bayes

	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 seasrch

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 gird_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 out perform 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 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 $\|$GLoVE("queen") - GLoVE("king") - GLoVE("wife") + GLoVE("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 $\|$GLoVE("queen") - GLoVE("king") - GLoVE("wife") + GLoVE("husband")$\|_2 = 0$
The actual values on using GLoVE embeddings are:

$$\|\text{GLoVE ("queen") - GLoVE ("king") - GLoVE ("wife") + GLoVE("husband")}\|_2 = 6.165$$
$$\|\text{GLoVE("queen") - GLoVE("king")}\|_2 = 5.966$$
$$\|\text{GLoVE("wife) - GLoVE("husband")}\|_2 = 3.152$$

While the analogy $\|$GLoVE("queen") - GLoVE("king") - GLoVE("wife") + GLoVE("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 consine 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)

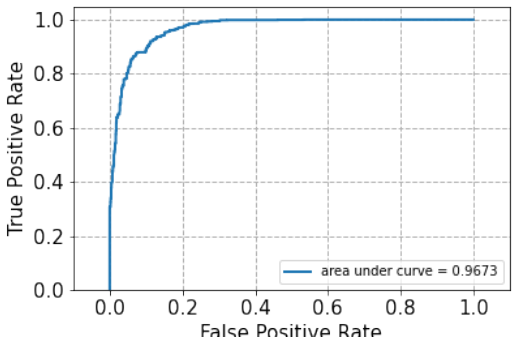
Since the GLoVE embeddings are 300-D vectors, the data matrix for our classification has a size $(4476, 300)$. Dimensionality was reduced to 100 components using the SVD method. The training matrix thus had a size $(4476, 100)$. Support vector machines were used for the classification with $\gamma =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

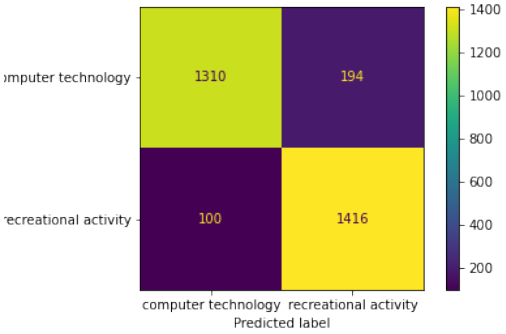
Table 8: accuracy, recall, precision and F-1 score on training with GLoVE embeddings of documents

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**

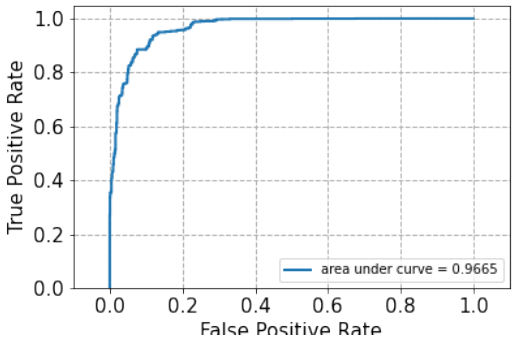
Figure 6: Evaluation on using GLoVE embeddings using SVM

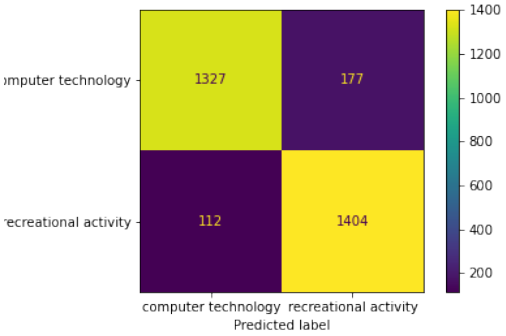
(a) roc curve SVM$\gamma = 0.1$ (b) confusion matrix SVM$\gamma = 0.1$



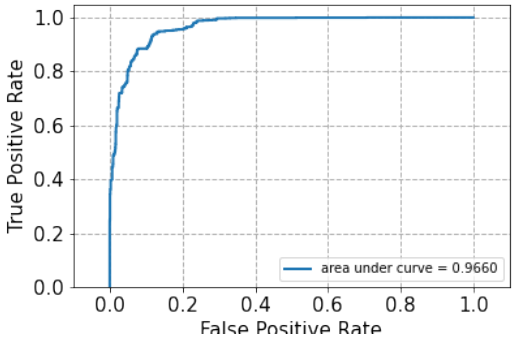


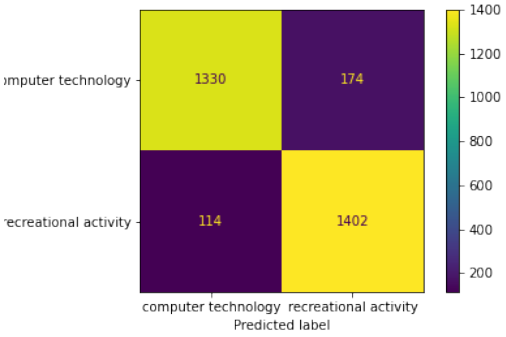
(c) roc curve SVM$\gamma = 1$ (d) confusion matrix SVM$\gamma = 1$





(e) roc curve SVM$\gamma = 10$ (f) confusion matrix SVM$\gamma = 10$



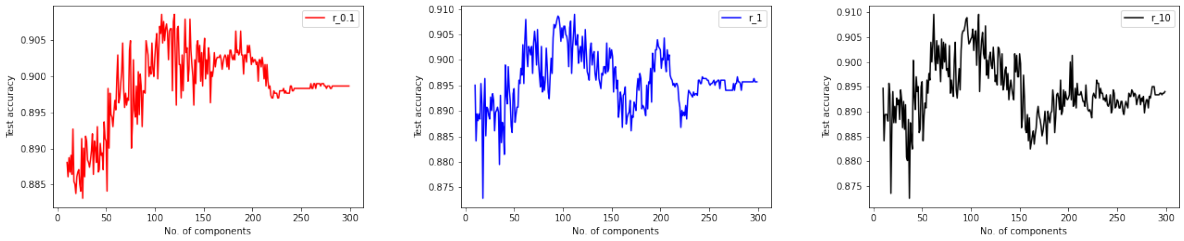


6.3. (Question 10)

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

(a) SVM $\gamma = 0.1$ (b) SVM $\gamma = 1$ (c) SVM $\gamma = 10$



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

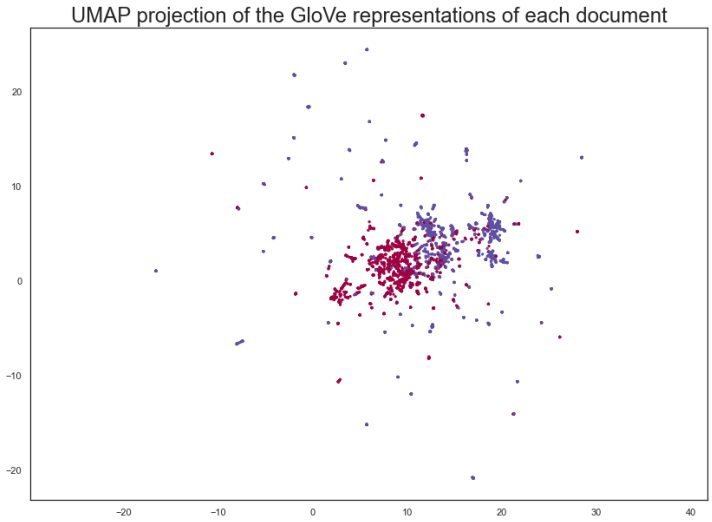
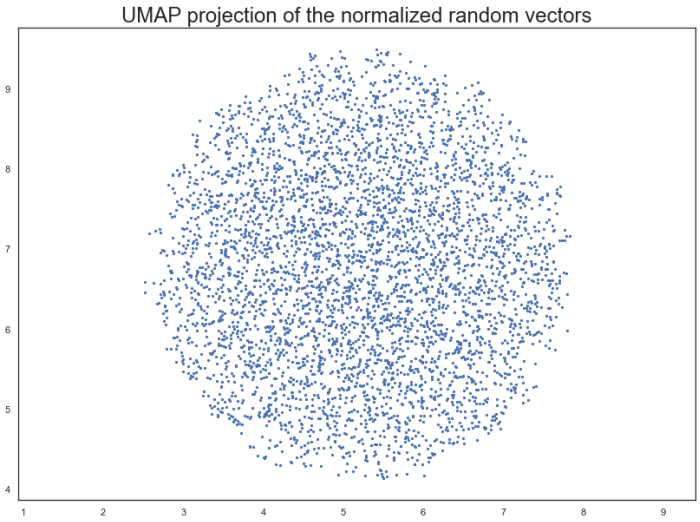


Figure 9: UMPA representation of random vectors

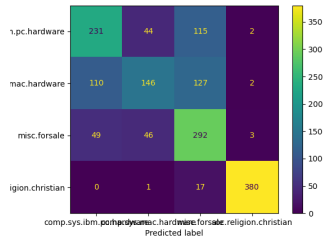


8. Multiclass Classification (Question 12)

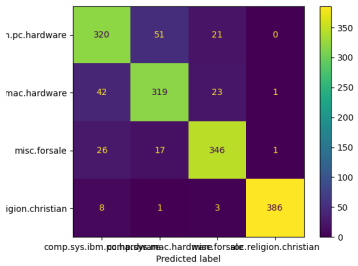
In the end, we try to perform multi-class classification. Specifically, we chooses data from comp.sys.ibm.pc.hardware, comp.sys.mac.hardware, misc.forsale and soc.religion.christian categories; used Naive Bayes classification and multiclass SVM classification (one vs one and one vs rest). The confusion 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)

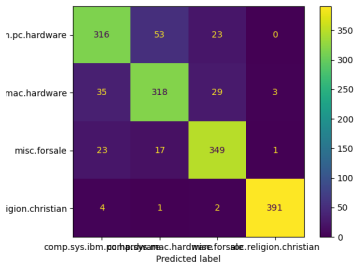


Table 9: accuracy, recall, precision and F-1 score of the classificers

	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.

```
import nltk
import random
from sklearn.datasets import fetch_20newsgroups
import numpy as np
from matplotlib import pyplot as plt
from nltk import pos_tag
from sklearn.metrics import auc
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.decomposition import NMF
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_curve

# # Question 1
twenty_train = fetch_20newsgroups(
    subset='train', shuffle=True, random_state=42)

# doc_number = list(map(lambda category: len(fetch_20newsgroups(
#    subset='train', categories=[category]).data), twenty_train.target_names))
ind, counts = np.unique(twenty_train.target, return_counts=True)

bar_colors = []
for i in range(len(counts)):
    rgb = np.random.rand(3,)
    bar_colors.append(rgb)

p1 = plt.bar(ind, counts, width=0.8, color=bar_colors)
plt.ylabel('Number of documents')
plt.xlabel('Categories')
plt.title('Number of training docs in each category')
plt.xticks(ind, np.arange(1, len(counts)+1))
plt.legend(p1, twenty_train.target_names, bbox_to_anchor=(
    1.04, 1), loc="upper left", prop={'size': 9})
plt.savefig("q1_G.png", bbox_inches="tight")

# Question 2
np.random.seed(42)
random.seed(42)

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']

target_map = {
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 1,
    5: 1,
    6: 1,
    7: 1,
}
twenty_train = fetch_20newsgroups(
    subset='train', categories=categories, shuffle=True, random_state=None)

twenty_test = fetch_20newsgroups(
    subset='test', categories=categories, shuffle=True, random_state=None)


nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
wnl = nltk.wordnet.WordNetLemmatizer()


def penn2morphy(penntag):
    """ Converts Penn Treebank tags to WordNet. """
    morphy_tag = {'NN': 'n', 'JJ': 'a',
                  'VB': 'v', 'RB': 'r'}
    try:
        return morphy_tag[penntag[:2]]
    except:
```

```
        return 'n'


def lemmatize_sent(list_word):
    # Text input is list of strings, returns array of lowercased strings(words).
    return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
            for word, tag in pos_tag(list_word)]


# overwrite analyzer with callable function:
vectorizer1 = CountVectorizer(stop_words='english')
stop_words = vectorizer1.get_stop_words()
# print(stop_words)
print("Number of stop words are:")
print(len(stop_words))

analyzer = CountVectorizer().build_analyzer()


def stem_rmv_punc(doc):
    return (word for word in lemmatize_sent(analyzer(doc)) if word not in stop_words and not word.isdigit())
    # Stopwords and checking for digits must be done here itself.
    # Because, when a callable is passed to analyzer in CoutVectorizer, it extracts features from THE RAW, UNPROCESSED INPUT.


count_vect = CountVectorizer(min_df=3, analyzer=stem_rmv_punc)
# G - I do not think stop_words = 'english' is necessary here because of the comments mentioned in stem_rmv_punc

X_train_counts = count_vect.fit_transform(twenty_train.data)
print("Shape of X_train_counts is")
print(X_train_counts.shape)

tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
print("Shape of X_train_tfidf is")
print(X_train_tfidf.shape)

# Question 3

# LSI

svd = TruncatedSVD(n_components=50)
X_train_reduced = svd.fit_transform(X_train_tfidf)
print(X_train_reduced.shape)
svd.components_
print(np.sum(np.array(X_train_tfidf - X_train_reduced.dot(svd.components_))**2))
# NMF

model = NMF(n_components=50, init='random', random_state=0)
W_train = model.fit_transform(X_train_tfidf)

print(W_train.shape)

# calculate ||X     WHk||^2
H = model.components_
H.shape
NMF_result = np.sum(np.array(X_train_tfidf - W_train.dot(H))**2)
print(NMF_result)
```

Listing 1: Code for Question 1, 2, 3

```
from sklearn.multiclass import OneVsOneClassifier
from sklearn.multiclass import OneVsRestClassifier
import pandas as pd
from sklearn.model_selection import GridSearchCV
from joblib import Memory
from shutil import rmtree
from tempfile import mkdtemp
from sklearn.model_selection import cross_val_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.naive_bayes import GaussianNB
import nltk
import random
from sklearn.datasets import fetch_20newsgroups
import numpy as np
from matplotlib import pyplot as plt
from nltk import pos_tag
from sklearn.metrics import auc
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.decomposition import NMF
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
```

```
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from statistics import mean
from sklearn import metrics

np.random.seed(42)
random.seed(42)

SCORE_TEMPLATE = """
{}
    accuracy (normalized): {}
    recall: {}
    precision: {}
     f 1  {}


"""

TARGET_MAP = {
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 1,
    5: 1,
    6: 1,
    7: 1,
}

result_file = open("results.txt", "w")

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']

# load data and convert to 2 category
twenty_train = fetch_20newsgroups(
    subset='train', categories=categories, shuffle=True, random_state=None)
twenty_train.target = np.array(
    list(map(lambda label: TARGET_MAP[label], twenty_train.target)))

twenty_test = fetch_20newsgroups(
    subset='test', categories=categories, shuffle=True, random_state=None)
twenty_test.target = np.array(
    list(map(lambda label: TARGET_MAP[label], twenty_test.target)))


# feature extraction
nltk.download('wordnet')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
wnl = nltk.wordnet.WordNetLemmatizer()

# configure stop words
stop_words = CountVectorizer(stop_words='english').get_stop_words()


def penn2morphy(penntag):
    # Converts Penn Treebank tags to WordNet
    morphy_tag = {'NN': 'n', 'JJ': 'a',
                  'VB': 'v', 'RB': 'r'}
    if len(penntag) > 1 and penntag[:2] in morphy_tag:
        return morphy_tag[penntag[:2]]
    else:
        return "n"


def lemmatize_sent(list_word):
    # Text input is list of strings, returns array of lowercased strings(words).
    return [wnl.lemmatize(word.lower(), pos=penn2morphy(tag))
            for word, tag in pos_tag(list_word)]


def stem_rmv_punc(doc):
    return (word for word in lemmatize_sent(CountVectorizer().build_analyzer()(doc)) if word not in stop_words and not word.isdigit())


def plot_roc(figure_name, fpr, tpr):
    # roc curve plot
    fig, ax = plt.subplots()

    roc_auc = 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)

    fig.savefig(figure_name)


def fit_predict_and_plot_roc(figure_name, pipe, train_data, train_label, test_data, test_label):
    # fit, predict and plot roc
    pipe.fit(train_data, train_label)

    if hasattr(pipe, 'decision_function'):
        prob_score = pipe.decision_function(test_data)
        fpr, tpr, _ = roc_curve(test_label, prob_score)
    else:
        prob_score = pipe.predict_proba(test_data)
        fpr, tpr, _ = roc_curve(test_label, prob_score[:, 1])

    plot_roc(figure_name, fpr, tpr)


# preprocessing
LSI_preprocessing_pipline = Pipeline([
    ('vect', CountVectorizer(min_df=3, analyzer=stem_rmv_punc)),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=50)),
])

reduced_train = LSI_preprocessing_pipline.fit_transform(twenty_train.data)
reduced_test = LSI_preprocessing_pipline.transform(twenty_test.data)


def calculate_socre(clf, title, average="binary"):
    pred = clf.predict(reduced_test)
    accuracy_score = metrics.accuracy_score(
        twenty_test.target, pred, normalize=True)
    recall_score = metrics.recall_score(
        twenty_test.target, pred, average=average)
    precision_score = metrics.precision_score(
        twenty_test.target, pred, average=average)
    f1_score = metrics.f1_score(
        twenty_test.target, pred, average=average)

    result_file.write(SCORE_TEMPLATE.format(
        title, accuracy_score, recall_score, precision_score, f1_score))


# Q4
result_file.write("============ Question 4 ============\n")

# r = 1000
svm1000_pipline = Pipeline([
    ('clf', LinearSVC(C=1000)),
])

# roc curve
fit_predict_and_plot_roc('q4_roc_r1000.png', svm1000_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(svm1000_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q4_confusion_matrix_r1000.png')

calculate_socre(svm1000_pipline["clf"], "linear SVMs with    = 1000")

# r = 0.0001
svm00001_pipline = Pipeline([
    ('clf', LinearSVC(C=0.0001)),
])

# roc curve
fit_predict_and_plot_roc('q4_roc_r0001.png', svm00001_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(svm00001_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
```

```

plt.savefig('q4_confusion_matrix_r0001.png')

calculate_socre(svm00001_pipline["clf"], "linear SVMs with    = 0.0001")


# cross validation
tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
for c in tradeoff:
    clf = LinearSVC(C=c)
    fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
    tradeoff[c] = mean(fold_score)

# pipeline for the best tradeoff value
svm_C = max(tradeoff, key=tradeoff.get)
svm_best_pipline = Pipeline([
    ('clf', LinearSVC(C=svm_C)),
])

# roc curve
fit_predict_and_plot_roc('q4_roc_best.png', svm_best_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(svm_best_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q4_confusion_matrix_best.png')

calculate_socre(svm_best_pipline["clf"],
                "linear SVMs with best    = {}".format(svm_C))

# Q5
result_file.write("============= Question 5 =============\n")

# Note: the instruction "you may need to come up with some way toapproximate this if you use sklearn.linear model.LogisticRegression"
#       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.

# Logistic regression
logistic_pipline = Pipeline([
    ('clf', LogisticRegression(penalty="none")),
])
# roc curve
fit_predict_and_plot_roc('q5_roc_none.png', logistic_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)
# confusion matrix
plot_confusion_matrix(logistic_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q5_confusion_matrix_none.png')

calculate_socre(logistic_pipline["clf"], "logistic classifier")

# cross validation on L1
tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
for c in tradeoff:
    clf = LogisticRegression(C=c, penalty="l1", solver="liblinear")
    fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
    tradeoff[c] = mean(fold_score)

# pipeline for the best L1
log1_C = max(tradeoff, key=tradeoff.get)
log_bestl1_pipline = Pipeline([
    ('clf', LogisticRegression(C=log1_C, penalty="l1", solver="liblinear")),
])

# roc curve
fit_predict_and_plot_roc('q5_roc_bestL1.png', log_bestl1_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(log_bestl1_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q5_confusion_matrix_bestL1.png')

calculate_socre(
    log_bestl1_pipline["clf"], "logistic classifier with L1, best    = {}".format(log1_C))

# cross validation on L2
tradeoff = {1000: 0, 100: 0, 10: 0, 1: 0, 0.1: 0, 0.01: 0, 0.001: 0}
for c in tradeoff:
    clf = LogisticRegression(C=c, penalty="l2")
    fold_score = cross_val_score(clf, reduced_train, twenty_train.target, cv=5)
    tradeoff[c] = mean(fold_score)

# pipeline for the best L2
log2_C = max(tradeoff, key=tradeoff.get)
log_bestl2_pipline = Pipeline([
```

```
    ('clf', LogisticRegression(C=log2_C, penalty="l2")),
])

# roc curve
fit_predict_and_plot_roc('q5_roc_bestL2.png', log_best12_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(log_best12_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q5_confusion_matrix_bestL2.png')

calculate_socre(
    log_best12_pipline["clf"], "logistic classifier with L2, best    = {}".format(log2_C))


# Q6
result_file.write("============= Question 6 =============\n")

# Naive Bayes
bayes_pipline = Pipeline([
    ('clf', GaussianNB()),
])

# roc curve
fit_predict_and_plot_roc('q6_roc.png', bayes_pipline, reduced_train,
                         twenty_train.target, reduced_test, twenty_test.target)

# confusion matrix
plot_confusion_matrix(bayes_pipline["clf"], reduced_test, twenty_test.target, display_labels=[
                      "computer technology", "recreational activity"])
plt.savefig('q6_confusion_matrix.png')

calculate_socre(bayes_pipline["clf"], "naive bayes classifier")


# Q7

def stem_rmv_punc_nonlemmatize(doc):
    return (word for word in CountVectorizer().build_analyzer()(doc) if word not in stop_words and not word.isdigit())


# load removed header footer data and convert to 2 category
twenty_train_nohf = fetch_20newsgroups(
    subset='train', categories=categories, shuffle=True, random_state=None, remove=('headers', 'footers'))
twenty_train_nohf.target = np.array(
    list(map(lambda label: TARGET_MAP[label], twenty_train_nohf.target)))

cachedir = mkdtemp()
memory = Memory(cachedir=cachedir, verbose=10)

grid_pipline = Pipeline([
    ('vect', CountVectorizer()),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', TruncatedSVD(n_components=50)),
    ('clf', GaussianNB()),
],
    memory=memory
)

param_grid = [
    {
        'vect__min_df': [3, 5],
        'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
        'reduce_dim': [TruncatedSVD(), NMF()],
        'reduce_dim__n_components': [50],
        'clf': [LinearSVC(C=svm_C)],
        'clf__C': [svm_C]
    },
    {
        'vect__min_df': [3, 5],
        'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
        'reduce_dim': [TruncatedSVD(), NMF()],
        'reduce_dim__n_components': [50],
        'clf': [LogisticRegression()],
        'clf__C': [log1_C],
        'clf__penalty': ["l1"],
        'clf__solver': ["liblinear"]
    },
    {
        'vect__min_df': [3, 5],
        'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
        'reduce_dim': [TruncatedSVD(), NMF()],
        'reduce_dim__n_components': [50],
        'clf': [LogisticRegression()],
        'clf__C': [log2_C],
```

```
        'clf__penalty': ["l2"],
    },
    {
        'vect__min_df': [3, 5],
        'vect__analyzer': [stem_rmv_punc, stem_rmv_punc_nonlemmatize],
        'reduce_dim': [TruncatedSVD(), NMF()],
        'reduce_dim__n_components': [50],
        'clf': [GaussianNB()],
    },
]

grid = GridSearchCV(grid_pipline, cv=5, n_jobs=1,
                    param_grid=param_grid, scoring='accuracy')
grid.fit(twenty_train.data, twenty_train.target)
df = pd.DataFrame(grid.cv_results_)
df.to_csv('q7_header_footer.csv')

grid.fit(twenty_train_nohf.data, twenty_train_nohf.target)
df = pd.DataFrame(grid.cv_results_)
df.to_csv('q7_no_header_footer.csv')
rmtree(cachedir)

# Q12
result_file.write("============= Question 12 ============\n")

categories = ['comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware',
              'misc.forsale', 'soc.religion.christian']

twenty_train = fetch_20newsgroups(
    subset='train', categories=categories, shuffle=True, random_state=None)

twenty_test = fetch_20newsgroups(
    subset='test', categories=categories, shuffle=True, random_state=None)

reduced_train = LSI_preprocessing_pipline.fit_transform(twenty_train.data)
reduced_test = LSI_preprocessing_pipline.transform(twenty_test.data)

# Naive Bayes
clf = GaussianNB().fit(reduced_train, twenty_train.target)

# confusion matrix
plot_confusion_matrix(clf, reduced_test, twenty_test.target,
                      display_labels=categories)
plt.savefig('q12_confusion_matrix_NB.png')

calculate_socre(clf, "naive bayes classifier", average='micro')

#  multiclass SVM classification
# one vs one
clf = OneVsOneClassifier(LinearSVC()).fit(reduced_train, twenty_train.target)
plot_confusion_matrix(clf, reduced_test, twenty_test.target,
                      display_labels=categories)
plt.savefig('q12_confusion_matrix_svm1to1.png')

calculate_socre(
    clf, "multiclass SVM classification with one VS one", average='micro')


# one vs rest
clf = OneVsRestClassifier(LinearSVC()).fit(reduced_train, twenty_train.target)
plot_confusion_matrix(clf, reduced_test, twenty_test.target,
                      display_labels=categories)
plt.savefig('q12_confusion_matrix_svm1tores.png')

calculate_socre(
    clf, "multiclass SVM classification with one VS rest", average='micro')

result_file.close()
```

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

```
#!/usr/bin/env python
# coding: utf-8

# In[2]:


print((37.0))


# In[3]:


#tweny_train.data[0]
#tweny_train.target[0]
```

```
import numpy as np
embeddings_dict = {}
dimension_of_glove = 300
with open("C:/Users/sriva/Desktop/glove.6B/glove.6B.300d.txt", '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


# In[30]:


glove_keys = embeddings_dict.keys()
print(type(glove_keys))
print(len(glove_keys))


# In[31]:


print(type(embeddings_dict['computer']))


# In[32]:


from sklearn.model_selection import cross_val_score
from sklearn.metrics import plot_confusion_matrix
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.naive_bayes import GaussianNB
import nltk
import random
from sklearn.datasets import fetch_20newsgroups
import numpy as np
from matplotlib import pyplot as plt
from nltk import pos_tag
from sklearn.metrics import auc
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.decomposition import NMF
from sklearn.decomposition import TruncatedSVD
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics import roc_curve
from sklearn.linear_model import LogisticRegression
from statistics import mean

np.random.seed(42)
random.seed(42)

TARGET_MAP = {
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 1,
    5: 1,
    6: 1,
    7: 1,
}
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']

# load data and convert to 2 category
twenty_train = fetch_20newsgroups(
    subset='train', categories=categories, shuffle=True, random_state=None)
twenty_train.target = np.array(
    list(map(lambda label: TARGET_MAP[label], twenty_train.target)))
twenty_test = fetch_20newsgroups(
    subset='test', categories=categories, shuffle=True, random_state=None)
twenty_test.target = np.array(
    list(map(lambda label: TARGET_MAP[label], twenty_test.target)))

# In[6]:


def plot_roc(figure_name, fpr, tpr):
    # roc curve plot
    fig, ax = plt.subplots()
    roc_auc = 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)
    fig.savefig(figure_name)
def fit_predict_and_plot_roc(figure_name, pipe, train_data, train_label, test_data, test_label):
    # fit, predict and plot roc
    pipe.fit(train_data, train_label)
    if hasattr(pipe, 'decision_function'):
        prob_score = pipe.decision_function(test_data)
        fpr, tpr, _ = roc_curve(test_label, prob_score)
    else:
        prob_score = pipe.predict_proba(test_data)
        fpr, tpr, _ = roc_curve(test_label, prob_score[:, 1])
    plot_roc(figure_name, fpr, tpr)


# In[33]:


# configure stop words
stop_words = CountVectorizer(stop_words='english').get_stop_words()
def rmv_stp_digits(doc):
    return (word for word in CountVectorizer().build_analyzer()(doc) if word not in stop_words and not word.isdigit())


# In[36]:


def get_glove (sen):
    #Gives the average of golve representation of words in subject and keyword lines.
    glove_vec = np.zeros((300,))
    sen = sen.lower()
    count = 0
    for item in sen.split("\n"):
        ww = item.split()
        if  len(ww)>0 and ww[0] in ["subject:","keywords:"]:
            for i in ww:
                if i in glove_keys and i not in stop_words:
                    #print(i)
                    glove_vec = glove_vec + embeddings_dict[i]
                    count = count + 1

    if count == 0:
        return (glove_vec)
    else:
        avg_vec = glove_vec/count
        return (avg_vec/np.linalg.norm(avg_vec))


# In[37]:


Xtrain_glove = np.zeros((len(twenty_train.target),300))

for ind in range(len(twenty_train.target)):
    Xtrain_glove[ind] = get_glove(twenty_train.data[ind])

print(Xtrain_glove.shape)

#Remove documents that gave 0 mean glove vectors
twenty_train.target = twenty_train.target[~np.all(Xtrain_glove == 0, axis = 1)]
Xtrain_glove = Xtrain_glove[~np.all(Xtrain_glove == 0, axis = 1)]
print(Xtrain_glove.shape)


# In[38]:


#Modifing test set data too:
Xtest_glove = np.zeros((len(twenty_test.target),300))

for ind in range(len(twenty_test.target)):
    Xtest_glove[ind] = get_glove(twenty_test.data[ind])

print(Xtest_glove.shape)

#Remove documents that gave 0 mean glove vectors
twenty_test.target = twenty_test.target[~np.all(Xtest_glove == 0, axis = 1)]
Xtest_glove = Xtest_glove[~np.all(Xtest_glove == 0, axis = 1)]
print(Xtest_glove.shape)


```

```
# In[13]:


svd = TruncatedSVD(n_components=100, random_state=0)#100 components chosen
Xtrain_reduced = svd.fit_transform(Xtrain_glove)
print(Xtrain_reduced.shape)
Xtest_reduced = svd.transform(Xtest_glove)
print(Xtest_reduced.shape)


# In[14]:


def apply_svm(r_val, train_data, train_labels, test_data, test_labels):

    svm_pipline = Pipeline([
        ('clf', LinearSVC(C=r_val)),
    ])
    # roc curve
    fit_predict_and_plot_roc("q9_roc_r{}.png".format(r_val), svm_pipline, train_data,
                             train_labels, test_data, test_labels)
    # confusion matrix
    plot_confusion_matrix(svm_pipline["clf"], test_data, test_labels, display_labels=[
        "computer technology", "recreational activity"])
    plt.savefig('q9_confusion_matrix_r{}.png'.format(r_val))


# In[15]:


apply_svm(r_val = 1000, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
      target)


# In[16]:


apply_svm(r_val = 0.1, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target
      )


# In[17]:


apply_svm(r_val = 100, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target
      )
apply_svm(r_val = 10, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target)
apply_svm(r_val = 1, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.target)


# In[18]:


apply_svm(r_val = 0.01, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
      target)
apply_svm(r_val = 0.001, train_data = Xtrain_reduced, train_labels = twenty_train.target, test_data = Xtest_reduced, test_labels = twenty_test.
      target)


# In[19]:


def apply_svm_ncomp(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):

    svd = TruncatedSVD(n_components= n_comp, random_state=0)#100 components chosen
    train_data = svd.fit_transform(Xtrain)
    #print(Xtrain_reduced.shape)
    test_data = svd.transform(Xtest)
    #print(Xtest_reduced.shape)

    svm_pipline = Pipeline([
        ('clf', LinearSVC(C=r_val)),
    ])

    # roc curve
    fit_predict_and_plot_roc("q10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipline, train_data,
                             train_labels, test_data, test_labels)
    # confusion matrix
    plot_confusion_matrix(svm_pipline["clf"], test_data, test_labels, display_labels=[
        "computer technology", "recreational activity"])
    plt.savefig('q10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))


# In[20]:


```

```
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)


# In[21]:


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)
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)
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)
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)
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)
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)
#=====================================================================
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)
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)
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)
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)
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)
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)
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)
#=====================================================================
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)
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)
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)
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)
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)
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)
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)
#=====================================================================
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)
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)
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)
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)
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)
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)
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)


# In[39]:


from sklearn import metrics
def svm_ncomp_score(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):

    svd = TruncatedSVD(n_components= n_comp, random_state=0)#100 components chosen
    train_data = svd.fit_transform(Xtrain)
    #print(Xtrain_reduced.shape)
    test_data = svd.transform(Xtest)
    #print(Xtest_reduced.shape)

    svm_pipline = Pipeline([
        ('clf', LinearSVC(C=r_val)),
    ])

    # roc curve
    fit_predict_and_plot_roc("tq10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipline, train_data,
                             train_labels, test_data, test_labels)
    # confusion matrix
```

```
    plot_confusion_matrix(svm_pipline["clf"], test_data, test_labels, display_labels=[
        "computer technology", "recreational activity"])
    plt.savefig('tq10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))

    pred_y = svm_pipline["clf"].predict(test_data)
    return([ metrics.accuracy_score(test_labels, pred_y, normalize = True), metrics.recall_score(test_labels, pred_y), metrics.precision_score(
      test_labels, pred_y), metrics.f1_score(test_labels, pred_y)])


# In[23]:


svm_ncomp_score(n_comp = 100, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove, test_labels =
      twenty_test.target)


# In[40]:


from sklearn import metrics
def svm_score(n_comp, r_val, Xtrain, train_labels, Xtest, test_labels):

    svd = TruncatedSVD(n_components= n_comp, random_state=0)#100 components chosen
    train_data = svd.fit_transform(Xtrain)
    #print(Xtrain_reduced.shape)
    test_data = svd.transform(Xtest)
    #print(Xtest_reduced.shape)

    #svm_pipline = Pipeline([
        #('clf', LinearSVC(C=r_val)),
    #])

    clf = LinearSVC(C=r_val).fit(train_data, train_labels)

    # roc curve
    #fit_predict_and_plot_roc("tq10_roc_r{}_ncomp{}.png".format(r_val, n_comp), svm_pipline, train_data,
                            #train_labels, test_data, test_labels)
    # confusion matrix
    #plot_confusion_matrix(svm_pipline["clf"], test_data, test_labels, display_labels=[
        #"computer technology", "recreational activity"])
    #plt.savefig('tq10_confusion_matrix_r{}_ncomp{}.png'.format(r_val, n_comp))

    pred_y = clf.predict(test_data)
    acc_score =  metrics.accuracy_score(test_labels, pred_y, normalize = True)
    #return([ metrics.accuracy_score(test_labels, pred_y, normalize = True), metrics.recall_score(test_labels, pred_y), metrics.precision_score(
      test_labels, pred_y), metrics.f1_score(test_labels, pred_y)])
    return acc_score



# In[25]:


ncomp_list = np.arange(10,300,10)
acc_list = []
for num in ncomp_list:
    acc_list.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))


# In[26]:


plt.plot(ncomp_list, acc_list)#Test accuracy for r_val = 0. 1
plt.xlabel('No. of components')
plt.ylabel('Test accuracy')
plt.savefig('q11.png')


# In[46]:


ncomp_list = np.arange(10,300,5)
acc_listr01 = []
acc_listr1 = []
acc_listr10 = []
for num in ncomp_list:
    acc_listr01.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))
for num in ncomp_list:
    acc_listr1.append(svm_score(n_comp = num, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))
for num in ncomp_list:
    acc_listr10.append(svm_score(n_comp = num, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))
#plt.plot(ncomp_list, acc_list)#Test accuracy for r_val = 0. 1
```

```
#plt.xlabel('No. of components')
#plt.ylabel('Test accuracy')
#plt.savefig('q11_r{}.png'.format(r_val))


# In[47]:


plt.plot(ncomp_list, acc_listr01, 'r-',label = 'r_0.1')
plt.plot(ncomp_list, acc_listr1, 'b-',label = 'r_1')
plt.plot(ncomp_list, acc_listr10, 'k-',label = 'r_10')
plt.legend()
plt.xlabel('No. of components')
plt.ylabel('Test accuracy')
plt.savefig('q11_allr.png')


# In[48]:


ncomp_list = np.arange(10,300,1)
acc_listr01a = []
acc_listr1a = []
acc_listr10a = []
for num in ncomp_list:
    acc_listr01a.append(svm_score(n_comp = num, r_val = 0.1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))
for num in ncomp_list:
    acc_listr1a.append(svm_score(n_comp = num, r_val = 1, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))
for num in ncomp_list:
    acc_listr10a.append(svm_score(n_comp = num, r_val = 10, Xtrain = Xtrain_glove, train_labels = twenty_train.target, Xtest = Xtest_glove,
      test_labels = twenty_test.target))


# In[53]:


fig = plt.figure()
plt.plot(ncomp_list, acc_listr01a, 'r-',label = 'r_0.1')
plt.legend()
plt.xlabel('No. of components')
plt.ylabel('Test accuracy')
plt.savefig('q10_r_0.1.png')
fig = plt.figure()
plt.plot(ncomp_list, acc_listr1a, 'b-',label = 'r_1')
plt.legend()
plt.xlabel('No. of components')
plt.ylabel('Test accuracy')
plt.savefig('q10_r_1.png')
fig = plt.figure()
plt.plot(ncomp_list, acc_listr10a, 'k-',label = 'r_10')
plt.legend()
plt.xlabel('No. of components')
plt.ylabel('Test accuracy')
plt.savefig('q10_r_10.png')


# In[27]:


#12
import numpy as np
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
get_ipython().run_line_magic('matplotlib', 'inline')

sns.set(style='white', context='notebook', rc={'figure.figsize':(14,10)})


# In[28]:


import umap
reducer = umap.UMAP()
scaled_data = StandardScaler().fit_transform(Xtrain_glove)
print(scaled_data.shape)
embedding = reducer.fit_transform(scaled_data)
print(embedding.shape)


# In[29]:
```

```

plt.scatter(embedding[:, 0], embedding[:, 1], c=twenty_train.target, cmap='Spectral', s=5)
plt.gca().set_aspect('equal', 'datalim')
#plt.colorbar(boundaries=[0,1])
plt.title('UMAP projection of the GloVe representations of each document', fontsize=24);
plt.savefig('q12_data.png')


# In[30]:


reducer = umap.UMAP()
embedding = reducer.fit_transform(Xtrain_glove)
print(embedding.shape)
plt.scatter(embedding[:, 0], embedding[:, 1], c=twenty_train.target, cmap='Spectral', s=5)
plt.gca().set_aspect('equal', 'datalim')
#plt.colorbar(boundaries=[0,1])
plt.title('UMAP projection of the GloVe representations of each document', fontsize=24);
plt.savefig('q12_data_noscaling.png')


# In[31]:


from random import gauss

def make_rand_vector(dims):
    vec = [gauss(0, 1) for i in range(dims)]
    mag = sum(x**2 for x in vec) ** .5
    return [x/mag for x in vec]


# In[32]:


random_vects = np.zeros((4476, 300))
for i in range(4476):
    random_vects[i] = make_rand_vector(300)


# In[33]:


reducer = umap.UMAP()
scaled_data = StandardScaler().fit_transform(random_vects)
print(scaled_data.shape)
embedding = reducer.fit_transform(scaled_data)
print(embedding.shape)
plt.scatter(embedding[:, 0], embedding[:, 1], cmap='Spectral', s=5)
plt.gca().set_aspect('equal', 'datalim')
#plt.colorbar(boundaries=[0,1])
plt.title('UMAP projection of the normalized random vectors', fontsize=24);
plt.savefig('q12_data_random.png')


# In[34]:


reducer = umap.UMAP()
embedding = reducer.fit_transform(random_vects)
print(embedding.shape)
plt.scatter(embedding[:, 0], embedding[:, 1], cmap='Spectral', s=5)
plt.gca().set_aspect('equal', 'datalim')
#plt.colorbar(boundaries=[0,1])
plt.title('UMAP projection of the normalized random vectors', fontsize=24);
plt.savefig('q12_data_random_Noscaling.png')

# In[162]:


aa = embeddings_dict["queen"] - embeddings_dict["king"] - embeddings_dict["wife"] + embeddings_dict["husband"]
print(np.linalg.norm(aa))


# In[164]:


aa1 = embeddings_dict["queen"] - embeddings_dict["king"]
aa2 = embeddings_dict["wife"] - embeddings_dict["husband"]
print(np.linalg.norm(aa1))
print(np.linalg.norm(aa2))


# In[4]:

```

```

aa = embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['husband']
bb = embeddings_dict['queen'] - embeddings_dict['king']
cc = embeddings_dict['wife'] - embeddings_dict['husband']
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'])]
print(np.linalg.norm(aa))
print(np.linalg.norm(bb))
print(np.linalg.norm(cc))
print(dd)


# In[14]:


from sklearn.metrics.pairwise import cosine_similarity
print(cosine_similarity([bb],[cc]))


# In[7]:


print(embeddings_dict['queen'][1:5])
print(embeddings_dict['king'][1:5])
print((embeddings_dict['queen']-embeddings_dict['king'])[1:5])


# In[18]:

print("****")
print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['mango']))
print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['king'] - embeddings_dict['mango'] + embeddings_dict['husband']))
print(np.linalg.norm(embeddings_dict['queen'] - embeddings_dict['mango'] - embeddings_dict['wife'] + embeddings_dict['husband']))
print(np.linalg.norm(embeddings_dict['mango'] - embeddings_dict['king'] - embeddings_dict['wife'] + embeddings_dict['husband']))

# In[10]:


ee = embeddings_dict['wife'] - embeddings_dict['mango']
ff = embeddings_dict['king'] - embeddings_dict['mango']
gg = embeddings_dict['queen'] - embeddings_dict['mango']
hh = embeddings_dict['husband'] - embeddings_dict['mango']
print(np.linalg.norm(ee))
print(np.linalg.norm(ff))
print(np.linalg.norm(gg))
print(np.linalg.norm(hh))


# In[17]:


ee = embeddings_dict['wife'] - embeddings_dict['mango']
ff = embeddings_dict['queen'] - embeddings_dict['king']
print(cosine_similarity([ee],[ff]))
ee = embeddings_dict['mango'] - embeddings_dict['husband']
ff = embeddings_dict['queen'] - embeddings_dict['king']
print(cosine_similarity([ee],[ff]))
ee = embeddings_dict['wife'] - embeddings_dict['husband']
ff = embeddings_dict['mango'] - embeddings_dict['king']
print(cosine_similarity([ee],[ff]))
ee = embeddings_dict['wife'] - embeddings_dict['husband']
ff = embeddings_dict['queen'] - embeddings_dict['mango']
print(cosine_similarity([ee],[ff]))
```

Listing 3: Code for Question 8, 9, 11