UCLA

ECE 219 - Large-Scale Data Mining: Models and Algorithms
Winter 2021

Project 1: Classification Analysis on Textual Data

Authors:
Swapnil Sayan Saha (UID: 605353215)
Grant Young (UID: 505627579)

Question 1:

In this question, we have been asked to plot the class distribution for the training documents in the "20 Newsgroups dataset". The entire dataset contains roughly 18000 newsgroup posts assigned to one of 20 topics, with the training set containing 11314 textual posts evenly distributed across all topics. Figure 1 shows the histogram plot for the training set.

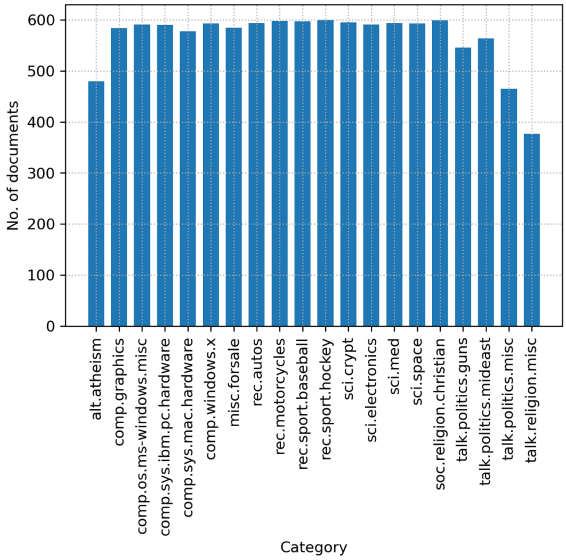


Figure 1: Histogram plot of the number of training documents for all categories in the "20 Newsgroups dataset"

From Figure 1, we can see that the dataset is evenly distributed, with each class containing roughly the same number of documents. For classification tasks, datasets should be balanced to prevent the classifier from overfitting on classes with significantly higher number of samples than classes with lower number of samples.

If the dataset is not balanced, then the following techniques can mitigate the imbalance if collecting new data is not possible:

- *Oversampling or data augmentation*: Synthetically increase the number of training samples for class with lower number of samples by adding slightly modified copies of the minority class samples.
- *Undersampling or resampling*: Remove some of the samples in the majority classes to equate sample count with minority classes, however, this leads to information loss.
- *Choosing appropriate models and loss function:* Decision trees and ensemble of one-vs-all classifiers perform well on unbalanced datasets. One could also modify the loss function of vanilla classifiers to include penalty terms for misclassifying minority classes (known as cost-sensitive learning)
- *Choosing appropriate performance metrics:* Confusion matrix, precision, recall and F1-score provide classifier performance from a class-dependent perspective compared to accuracy.

Question 2:

In this question, we are asked to extract numerical features from the textual data to make the data suitable for use by machine learning algorithms. More specifically, the question asks to convert the training and test set for 8 categories (4 from computer technology and 4 from recreational activity) into Term Frequency-Inverse Document Frequency (TF-IDF) matrices and report the shapes of the resulting matrices. The shapes are:
Training set: (4732, 12161)
Test set: (3150, 12161)

The steps undertaken for feature extraction are as follows:

- Use wordnet *lemmatizer* from NLTK library to convert words within each file to its meaningful root-form (called lemma) considering the specific context. Since a word can have numerous lemmas given context, we also used *'part-of-speech' (POS) tag* to select the correct lemmas based on the semantics and syntactic meaning of the word. Lemmatization reduces the variability of otherwise different forms of the same stem word conveying the same abstract information, decreasing the size of the feature vector to aid computational tractability.
- *Removal of punctuation* characters by comparing against `string.punctuation()` and *removal of numeric characters* using `char.isdigit()` to eliminate the presence of unwanted noise artefacts in the feature vector.
- Use `CountVectorizer()` to convert the lemmatized text into a *bag-of-words* matrix, which contains the frequencies of words in the vocabulary (use fit and transform on training set and transform on test set).
    - To omit common words in the English language (called stop-words) that do not provide any contextual information such as "and", "the", "I" etc., we set the argument `stopwords` to `"english"`.
    - To remove rare words, we set the argument `min_df` to 3. This removes all words that have a count of less than 3 in the training set. Decreasing the value of `min_df` increases the size of the resulting bag-of-words matrix.
- Convert the matrix of token counts to *TF-IDF matrix* (normalized matrix of counts based on importance of certain words) using `TfidfTransformer()` function (use fit and transform on training set and transform on test set). The TF-IDF matrix quantifies the importance of specific discriminating words (called keywords) in each document against words that are common across all documents within all classes due to the context of the classes. In other words, the TF-IDF matrix provides more useful and distinguishing features over bag-of-words for the classifier we want to use.

The tf-idf score for each term *t* in document *d*, with each element in bag-of-words matrix being tf(*t,d*) is:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \log\left(\frac{n}{\text{df}(t)}\right) + 1$$

where *n* = total number of documents, df(*t*) is the number of documents containing term *t*.

Question 3:

In this question, we are asked to reduce the dimensionality of the TF-IDF matrix. As data moves into higher dimensions, the rapid increase in volume causes the data to become sparse in each dimension, causing the amount of data required for accurate classification and representation to increase exponentially for each dimension or feature. This is referred to as the "Curse of Dimensionality". The question mentions two ways of dimensionality reduction:

- *Latent Semantic Indexing* (LSI): LSI reduces the rank of the feature matrix by multiplying the TF-IDF matrix $\mathbf{X}$ with the first k principal components in the feature space, represented in columns of matrix $\mathbf{V}_k$, which is found using singular value decomposition (SVD) of the TF-IDF matrix:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$
$$\mathbf{X}_{\text{reduced}} = \mathbf{X}\mathbf{V}_k$$

  The goal is to reduce the Frobenius norm of the difference between $\mathbf{X}$ and $\mathbf{U}_k \Sigma \mathbf{V}_k^{\mathrm{T}}$. k was chosen as 50. To obtain the SVD of the TF-IDF matrix, we used the `TruncatedSVD()` function, while obtaining the full decomposition using `randomized_svd()` function for finding out the approximation error. We use fit and transform on the training set and transform on the test set.
- *Non-negative Matrix Factorization* (NMF): NMF attempts to find two non-negative matrices $\mathbf{W}$ and $\mathbf{H}$ whose product is as close to $\mathbf{X}$ as possible ($\{\mathbf{W} \in \mathcal{R}^{n\times r}, \mathbf{H} \in \mathcal{R}^{r\times m}\} | \mathbf{X} \in \mathcal{R}^{n\times m}$), with $\mathbf{W}$ being the low-rank feature matrix for $\mathbf{X}$, where r = number of topics. $\mathbf{H}$ (coefficient matrix) provides weighing factors for all the topics in each document, while $\mathbf{W}$ (basis vectors) corresponds to the clusters discovered in the document. $\mathbf{W}$ and $\mathbf{H}$ are found by solving the following optimization problem

$$\min_{\{\mathbf{W},\mathbf{H}\}\geq 0} ||\mathbf{X} - \mathbf{W}\mathbf{H}||_F^2$$

  We chose r as 50 and used `NMF()` function to obtain $\mathbf{W}$. We use fit and transform on the training set and transform on the test set.

We obtained the following error values for LSI and NMF:

- $||\mathbf{X} - \mathbf{U}_k\Sigma\mathbf{V}_k||_F^2$: training set - 4103.0367390859365, test set - 2643.0475837323024
- $||\mathbf{X} - \mathbf{W}\mathbf{H}||_F^2$: training set - 4147.199485161415, test set - 2858.9878541431635

We observe that *the error is lower for LSI than NMF*. This is because NMF only allows positive entries in the reduced-rank feature matrix while LSI has no such restriction. As a result, LSI is able to better represent the higher-dimensional feature matrix, providing a deeper factorization with lower information loss than NMI.

The shape of the resulting matrices: Training set: LSI and NMI (4732, 50) Test set: LSI and NMI (3150, 50).

Question 4:

In this question, we are asked to train hard margin and soft margin support vector machines (SVM) with linear kernels and compare their performances, as well as perform hyperparameter optimization via 5-fold grid-search cross-validation to obtain the optimal value of the tradeoff parameter in the SVM cost function. The feature matrix from Question 4 to 6 corresponds to the TF-IDF LSI matrix. The metrics for comparison include:

- *Confusion matrix*: Shows the count for true positives (TP), false positives (FP) (type 1 error, reject a true null hypothesis due to wrongly observing differences in samples from same class), true negatives (TN) and false negatives (FN, accept a false null hypothesis due to failure of observing differences among samples from different classes) (type 2 error). The rows illustrate the count for true classes while the columns show the count for predicted classes. The diagonal entries represent the counts for correct conclusion.
- *Receiver Operating Characteristic* (ROC) *curve:* The ROC curve shows the plot for TP rate vs FP rate.
- *Accuracy:* Ratio of correct classification count to total number of samples in the test set: (TP+TN)/(TP+FP+FN+TN)
- *Precision* (P)*:* Measure of exactness of a classifier. Low precision indicates high number of FP. Precision is given as: TP/(TP+FP)
- *Recall* (R): Measure of sensitivity or completeness of a classifier. Low recall indicates high number of FN. Recall is given as: TP/(TP+FN)
- *F1-score*: Weighted mean of precision and recall, given by:

  $$F1 = \frac{2 \times R \times P}{R+P}$$

  F1-score is a better metric than accuracy when class distribution is not balanced.

The SVM classifier tries to find a decision boundary (in our case, a straight line) between class labels such that the gap between the training samples and boundary is maximized, thus providing the highest confidence classification. The loss function for multiclass linear SVM is given by:

$$\arg\min_{\mathbf{w},b} 0.5||\mathbf{w}||_2^2 + \frac{\gamma}{m}\sum_{i=1}^{m}\sum_{j\neq y^{(i)}} \max(0, 1 + \mathbf{w}_j^T\mathbf{x}^{(i)} - \mathbf{w}_{y^{(i)}}^T\mathbf{x}^{(i)})$$

where, $\mathbf{x}^{(i)}$ represents *i*th data point, $\mathbf{w}$ represents feature weights, $y^{(i)}$ represents binary class label for $\mathbf{x}^{(i)}$, γ represents a trade-off parameter. The first portion (ridge, Tikhonov or L2 regularization) acts as a regularization term, encouraging maximization of class margins and generalization to prevent overfitting or ill-conditioned classifiers. The second term, known as hinge loss, is 0 when the classification is correct provided a large enough margin, and large for misclassifications. When correct classification is made, $\mathbf{w}_{y^{(i)}}^T\mathbf{x}^{(i)} \geq \mathbf{w}_j^T\mathbf{x}^{(i)}$ for all $j \neq y^{(j)}$, causing $\mathbf{w}_j^T\mathbf{x}^{(i)} - \mathbf{w}_{y^{(i)}}^T\mathbf{x}^{(i)} \leq 0$, ultimately leading to hinge loss being small. If γ is very large, then misclassification is highly penalized and and if γ is very small, then some misclassification is allowed.

To tune the hyperparameter γ, we perform n-fold grid-search cross validation, which performs exhaustive grid search over list of possible hyperparameters coupled with cross validation. For a training set with N examples:

- Split the training set into n equal sets (called fold), each containing N/n samples.
- Train on $n-1$ of these folds and evaluate the performance of the classifier on the remaining fold (validation fold).
- Repeat the steps several times until optimal values of weights and biases are found for the current hyperparameter
- Repeat the steps for all hyperparameters in the list.

Figure 2 and 3 show the ROC curves and confusion matrices for hard and soft-margin SVM.

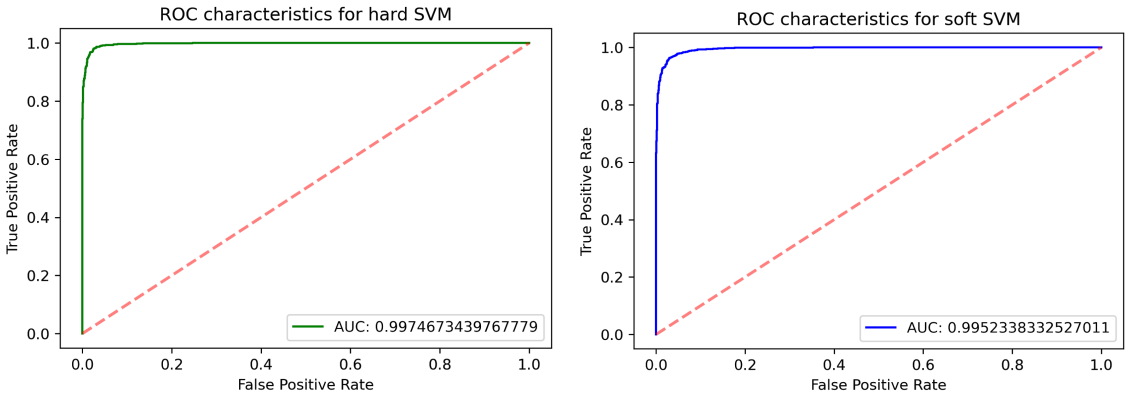


Figure 2: ROC curves for hard and soft margin SVM

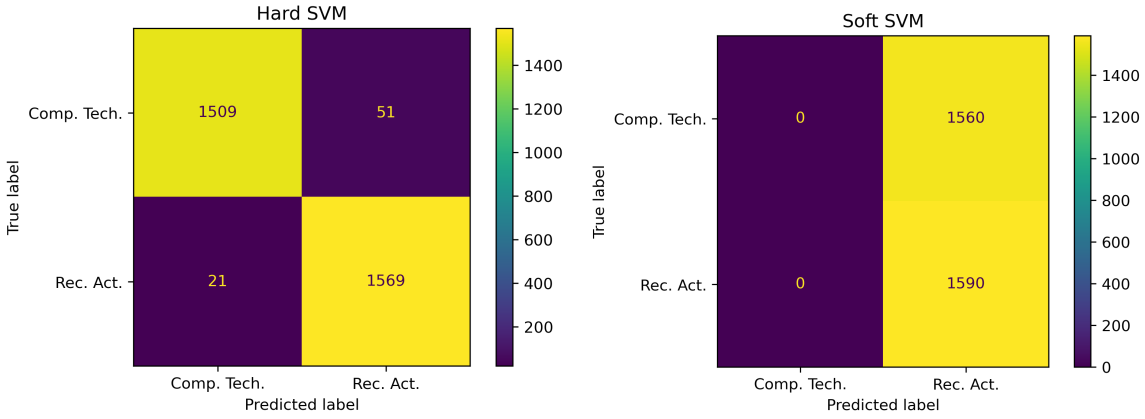


Figure 3: Confusion matrices for hard and soft margin SVM

The accuracy, precision, recall and F1-score for hard and soft margin SVM are reported in Table 1.

Table 1: Performance metrics of hard and soft margin SVM

Metric	**Hard SVM**	**Soft SVM**

Accuracy	0.9771428571428571	0.5047619047619047
Recall	0.9867924528301887	1.0
Precision	0.9685185185185186	0.5047619047619047
F1 score	0.977570093457944	0.6708860759493671

Judging from the confusion matrices in Figure 3 and reported metrics in Table 1, *the hard margin SVM performs better than soft margin SVM*.

In case of soft margin SVM, the goal is to allow the SVM classifier make some misclassifications by allowing a wider margin such that other samples can still be classified when the classes are linearly inseparable. When γ is small, the hinge loss is given less importance and misclassification is not penalized as harshly as when γ is large. In other words, a soft-margin classifier might correctly label all items belonging to a certain class *C*, but it may also erroneously bin items from other classes to class *C* (higher recall than hard SVM) due to finite degree of misclassification allowance. In this case, we want both high recall and high precision, but not just high recall.

If we examine the ROC curve of the soft margin SVM from Figure 2 and the recall from Table 1, we see that it conflicts with the confusion matrix, accuracy, precision and F1 score of the named SVM. From the confusion matrix, we see that the soft SVM has a very low misclassification rate for one of the classes and a very high misclassification rate for the other class, which means that the SVM is failing to observe differences among the classes due to a relaxed and wider margin. Mathematically, the margin *d* is given by:

$$d = \frac{\xi}{||\mathbf{w}||}$$

where ξ is referred to as slack variables. If γ is small, the hinge loss is no longer minimized properly, causing ξ to increase (more support vectors) and ultimately *d* to increase. In other words, the model fails to capture the shape of the data due to the separating hyperplane having low bias and high variance, indicating the bias term *b* in the loss function is not properly optimized.

Figure 4 shows the ROC curve and confusion matrix for the SVM with optimal value of γ found via 5-fold cross validation. The accuracy, precision, recall and F1-score for the SVM are reported in Table 2. The search space for γ was [0.001,0.01,0.1,1,10,100,200,400,600,800,1000], with the average validation accuracies listed as follows:

γ : 0.001 Avg. Validation Accuracy: 0.504860570043154
γ : 0.01 Avg. Validation Accuracy: 0.508875697373033
γ : 0.1 Avg. Validation Accuracy: 0.9632284883162809
γ : 1 Avg. Validation Accuracy: 0.9708359099950663

γ : 10 Avg. Validation Accuracy: 0.976330283012339
γ : 100 Avg. Validation Accuracy: 0.9759070035340264
γ : 200 Avg. Validation Accuracy: 0.9767522229986314
γ : 400 Avg. Validation Accuracy: 0.977174832730934
γ : 600 Avg. Validation Accuracy: 0.9771746094822641
γ : 800 Avg. Validation Accuracy: 0.9769634162404477
γ : 1000 Avg. Validation Accuracy: 0.9771746094822641

The best value of γ is 400.

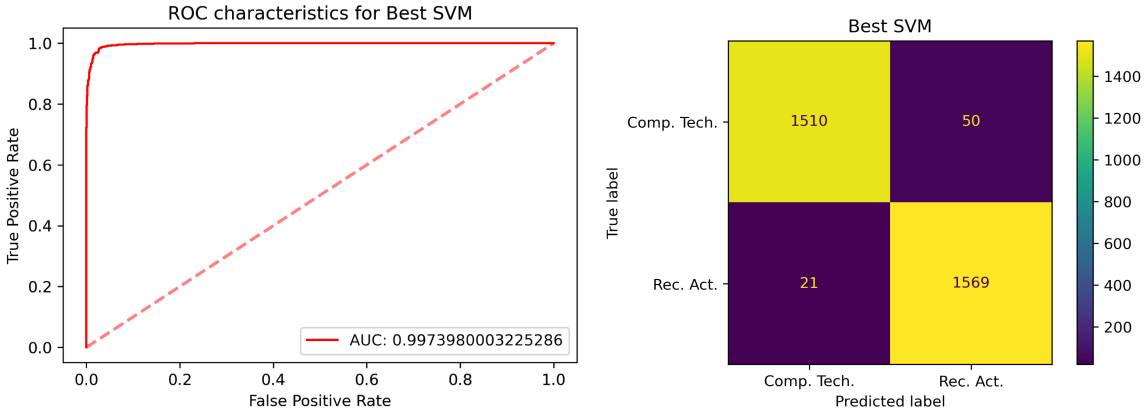


Figure 4: ROC curve and confusion matrix for best SVM found via 5-fold cross validation

Table 2: Performance metrics for best SVM found via 5-fold cross validation

Metric	**Best SVM with γ = 400**
Accuracy	0.9774603174603175
Recall	0.9867924528301887
Precision	0.9691167387276096
F1 score	0.9778747273293862

Question 5:

In this question, we are asked to train and compare performance metrics of logistic regression classifiers with various types of regularization parameters, while also using cross validation to obtain the best regularization parameter.

In logistic regression, a sigmoid function is applied upon a linear combination of feature vectors, with the goal of obtaining the optimal weights **w** (and intercept b) that maximizes the likelihood of correct binary classification. The sigmoid function is given by:

$$h_\Theta(\mathbf{x}) = \frac{1}{1 + e^{(-\Theta^T \mathbf{x})}}, \Theta = f(\mathbf{w}, b)$$

The loss function, which is known as cross entropy loss, is given by:

$$\arg\min_\Theta \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\Theta(\mathbf{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_\Theta(\mathbf{x}^{(i)})) \right] + \lambda \mathrm{R}(f)$$

Each part within the summation (called logistic or log loss) is dedicated to penalize the loss function for misclassification of each class, i.e., if the target class is 0, then the value of the logistic loss associated with class 0 is large when the class is predicted as 1 and vice versa. R(f) is a regularizer term weighted by λ to prevent overfitting and encourage generalization.

Figure 5 shows the ROC curve and confusion matrix for the logistic classifier without regularization. The accuracy, precision, recall and F1-score for the classifier are reported in Table 3. To perform regularization for `sklearn.linear model.LogisticRegression()`, we set the value of parameter `C` to 1000000 (very large). Large values of `C` indicate minimal regularization (`C` is inversely proportional to λ)

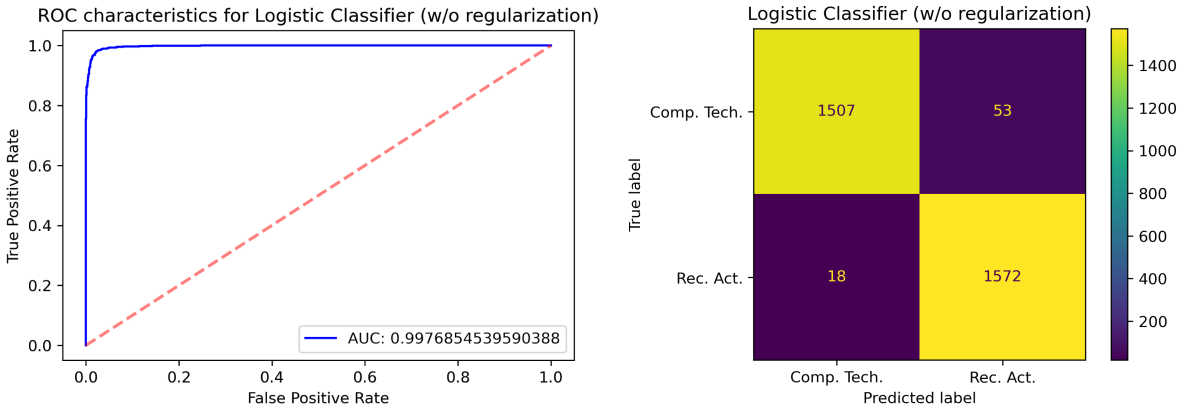


Figure 5: ROC curve and confusion matrix for logistic regression classifier without regularization

Table 3: Performance metrics for logistic regression classifier without regularization

Metric	**Logistic Classifier (w/o regularization)**

Accuracy	0.9774603174603175
Recall	0.9886792452830189
Precision	0.9673846153846154
F1 score	0.9779160186625194

The accuracy, precision, recall and F1-score for logistic classifiers with best regularization strengths obtained using 5-fold cross validation for L1 and L2 regularization terms are reported in Table 4. We have also included information from Table 3 in Table 4 for easier comparison. The search space for `C` was [0.001,0.01,0.1,1,10,100,200,400,600,800,1000], with the average validation accuracies listed as follows:

L1 Reg. Param.: 0.001 Avg. Validation Accuracy: 0.495139429956846
L1 Reg. Param.: 0.01 Avg. Validation Accuracy: 0.9363890867120158
L1 Reg. Param.: 0.1 Avg. Validation Accuracy: 0.9518178022954429
L1 Reg. Param.: 1 Avg. Validation Accuracy: 0.9702023302696174
L1 Reg. Param.: 10 Avg. Validation Accuracy: 0.9780204986928791
L1 Reg. Param.: 100 Avg. Validation Accuracy: 0.977174832730934
L1 Reg. Param.: 200 Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 400 Avg. Validation Accuracy: 0.9775976657119065
L1 Reg. Param.: 600 Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 800 Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 1000 Avg. Validation Accuracy: 0.9773862492214203

L2 Reg. Param.: 0.001 Avg. Validation Accuracy: 0.7499996651269951
L2 Reg. Param.: 0.01 Avg. Validation Accuracy: 0.9448437370934363
L2 Reg. Param.: 0.1 Avg. Validation Accuracy: 0.9636519910432634
L2 Reg. Param.: 1 Avg. Validation Accuracy: 0.969145917563196
L2 Reg. Param.: 10 Avg. Validation Accuracy: 0.9742176808481663
L2 Reg. Param.: 100 Avg. Validation Accuracy: 0.9769636394891178
L2 Reg. Param.: 200 Avg. Validation Accuracy: 0.977174832730934
L2 Reg. Param.: 400 Avg. Validation Accuracy: 0.9773860259727503
L2 Reg. Param.: 600 Avg. Validation Accuracy: 0.9771746094822639
L2 Reg. Param.: 800 Avg. Validation Accuracy: 0.9771746094822639
L2 Reg. Param.: 1000 Avg. Validation Accuracy: 0.9769634162404477

The best regularization strength for L1 logistic classifier was 10 and the best regularization strength for the L2 logistic classifier was 400.

Table 4: Performance metrics for logistic classifier without regularization and logistic classifiers with L1 and L2 regularization

Metric	**Logistic Classifier (w/o regularization)**	**Logistic Classifier (L1 reg.) (C = 10)**	**Logistic Classifier (L2 reg.) (C = 400)**
Accuracy	0.9774603174603175	0.9768253968253968	0.9765079365079365
Recall	0.9886792452830189	0.9886792452830189	0.9886792452830189
Precision	0.9673846153846154	0.9661954517516902	0.9656019656019657
F1 score	0.9779160186625194	0.9773080509791732	0.9770043505282784

From table 4, we see that the logistic classifier without regularization has the highest accuracy, precision and F1 score, with L1 logistic classifier coming second and L2 logistic classifier coming third. This is probably due to the fact that regularization attempts to improve generalizability by increasing estimator bias while reducing variance, causing many of the weights to be very small, leading to a simpler model which performs well on unseen test data but may have lower accuracy and precision. Note that the inclusion of regularization, however, does not make significant difference in the test error for this dataset.

Figure 6 shows the effects of regularization strength on mean value of learned coefficients and test accuracy. Note that, as mentioned earlier, **lower values of C indicate higher regularization strength.**

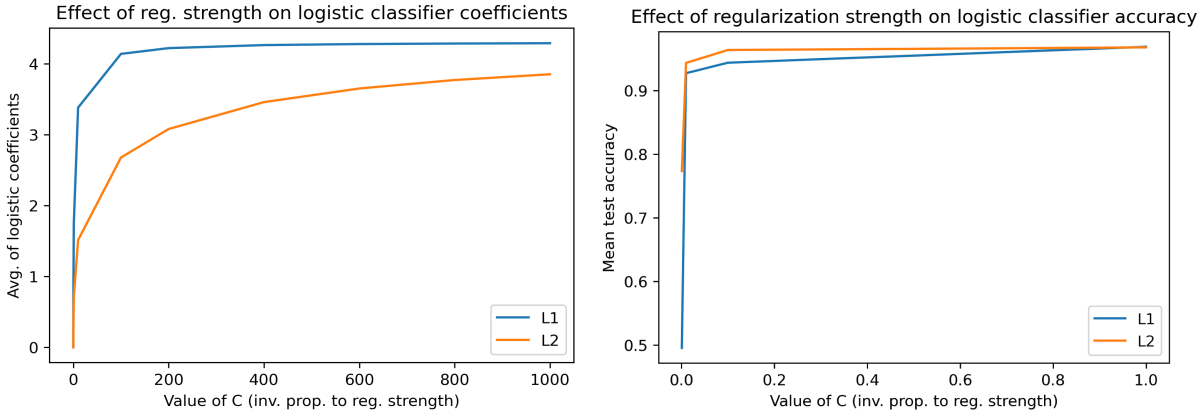


Figure 6: (Left) Effect of C on learned coefficients (Right) Effect of C on test accuracy

From Figure 6, we see that as regularization strength increases (C decreases), the mean value of logistic coefficients and test accuracy drops because:

- As regularization strength increases, more weights in the learned model are set to 0 as mentioned earlier. This is due to the fact that the L1 (Lasso) and L2 (ridge or Tikhonov) regularization are formulated as the following optimization functions:

$$\Omega(\theta)_{L1} = \arg\min_{\mathbf{w}} ||\mathbf{w}||$$

$$\Omega(\theta)_{L2} = \arg\min_{\mathbf{w}} 0.5||\mathbf{w}||^2$$

Thus, to prevent $\Omega(\theta)$ from becoming too large, the optimization program will attempt to minimize the weights, ideally the weights should approach 0.

- As models become more and more simpler, they start losing the most important weights required for distinguishing between classes. As a result, very large values of regularization strength will cause the test accuracy to drop dramatically. However, finite values of regularization strength can help make the model more generalizable on unseen test data, preventing overfitting. Regularization also makes the model more stable by reducing variance and sensitivity to outliers and increasing bias.

One might be interested in the aforementioned regularization terms in the following cases:

- If one wants to make complex models that follow the training data as closely as possible (while being certain that overfitting has not occurred), then no regularization is required.
- L1 regularization is suitable for feature selection and construction of simple and sparse models, where features associated with 0 weights can be discarded. This is because L1 regularization encourages all kinds of weights to shrink to 0, regardless of size of **w**, as the subgradient of ||**w**|| is sign(**w**). L1 regularization is more likely to zero out coefficients than L2 regularization for similar test accuracies because it assumes priors on the weights sampled from an isotropic Laplace distribution (linear descent), which has a much lower Q factor than Gaussian distribution (exponential descent). For example, the mean value of learned coefficients for C = 10 for L1 classifier (best for L1) was 3.3813, while the value for C = 400 for L2 classifier (best for L2) was 3.45937.
- L2 regularization, which assumes priors on the weights sampled from a unit Gaussian distribution, is suitable for reducing the effects of collinear features, which can lead to increased variance (hence instability and sensitivity to outliers) of the model. This is because the subgradient of $||\mathbf{w}||^2$ depends on not just the sign of **w** but also the magnitude of **w**, which can be thought of as scaling the variance of weights, reducing dependence of the model on few features and encouraging distributed contribution of all the features. This leads to models with diffuse weights compared to sparse weights from L1 regularization.

While both linear SVM and logistic classifier yield a linear decision boundary, they have some key differences:
Yielding Decision Boundary

- Logistic regression yields a decision boundary on the basis of maximizing the conditional likelihood (probability) of correct classification on the entire training set, spewing out probabilities specifying confidence in making a decision. SVM, on the other hand, is geometric and deterministic in nature, using only a subset of points closest to the decision boundary called support vectors to find an optimal separating hyperplane that maximizes the distance of the vectors to the margin. As a result, logistic regression does not guarantee it will yield an optimal decision boundary and , whereas SVM yields the

best separating hyperplane (or margin) that reduces misclassification rate. The decision boundary for SVM is not fixed but alters depending upon points near the margin.

Performance

- Logistic regression is more prone to overfitting as its decision is based on observations on the entire training set, whereas SVM provides better generalization. In addition, logistic loss is sensitive to outliers and does not go to zero. Hinge loss can be 0 and is much less sensitive to outliers.
- As discussed earlier, logistic regression provides high confidence in classifying data further away from the hyperplane but fails for samples near the margin.
- SVM is more efficient and fast at handling complex, high-dimensional and unstructured data through the kernel trick. Logistic regression is more suitable for structured datasets.

Question 6:

In this question, we are asked to report the performance metrics for Gaussian Naive Bayes classifier. The Naive Bayes classifier uses Bayes theorem for classification, assuming conditional independence between all the features *x* given class variable value. The loss function is given as:

$$\hat{y} = \arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

P(y) is found using Maximum A Posteriori (MAP) estimation of the labels *y*, whereas for Gaussian Naive Bayes, $P(x_i \mid y)$ is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

The Gaussian Naive Bayes assumes the likelihood of the features to be normally distributed.

Figure 7 shows the ROC curve and confusion matrix for Gaussian Naive Bayes classifier. The accuracy, precision, recall and F1-score for the classifier are reported in Table 5.

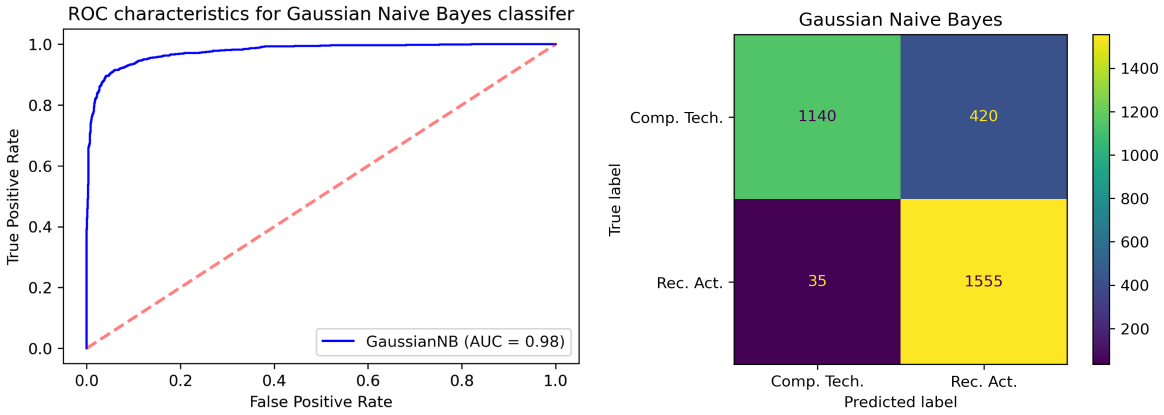


Figure 7: ROC curve and confusion matrix for Gaussian Naive Bayes classifier

Table 5: Performance metrics for logistic regression classifier without regularization

Metric	**Gaussian Naive Bayes**
Accuracy	0.8555555555555555
Recall	0.9779874213836478
Precision	0.7873417721518987
F1 score	0.8723702664796633

Question 7:

In this question, we are asked to find out the optimal combination of feature extraction techniques, dimensionality reduction approaches and classification algorithms using 5-fold grid search cross validation using the `Pipeline` feature in scikit-learn. In addition, we want to check whether removing headers and footers yield any performance improvement.

We designed the following parameters for the pipeline:

- `min_df` for `CountVectorizer()` was chosen as either 3 or 5.
- We redesigned the lemmatizer to be callable as a function by `CountVectorizer()`. In addition, we added a function that just returns the tokenized sentence without lemmatization. We implemented punctuation and number removal within both the lemmatizer and the simple tokenizer.
- For dimensionality reduction, we considered LSI and NMF with 50 components.
- The following classifiers were included:
    - Gaussian Naive Bayes
    - Linear SVM with $\gamma = 400$
    - Logistic Regression with L1 Regularization and `C` = 10
    - Logistic Regression with L2 Regularization and `C` = 400

In addition, we called the pipeline twice, once for data with headers and footers and the other for data without headers and footers.

For the data with no header and footer, the following combination achieved the best test accuracy of 0.966984126984127:

- `min_df` = 5
- Lemmatization: used
- Dimension reduction technique: LSI (SVD)
- Classifier: Logistic Regression with L2 Regularization and C = 400

For the data with header and footer, the following combination achieved the best test accuracy of 0.9774603174603175:

- `min_df` = 3
- Lemmatization: used
- Dimension reduction technique: LSI (SVD)
- Classifier: Linear SVM with $\gamma = 400$

Judging from the test accuracy, we can say that classifiers perform better for data with header and footer, probably because of important discriminating keywords in the headers and footers.

Question 8:

a. The ratio of co-occurrence probabilities are better able to separate those groups of contextual words that are relevant to the target words we are interested in from the irrelevant groups of words. There are two distinct cases to consider:
   - For those groups of words that are related to one of the target words in the co-occurrence ratio, the ratio will be either very large or very small. More specifically, if the word is related to the target word in the numerator of the co-occurrence ratio, the ratio will be large and if the word is related to the target word in the denominator of the co-occurrence ratio, the ratio will be very small.
   - For those groups of words that are either related to both the target words (which does not specify the context very well) or not related to both the target words in the co-occurrence ratio, the ratio will be close to one.

   In other words, the co-occurrence probability ratio filters out the noise from non-distinguishing groups of words such that large values of the ratio helps find contextual words correlating well with the target word in the numerator, while small values of the ratio helps find contextual words relevant to the target word in the denominator. The raw probabilities themselves are vastly affected by the noise and makes it difficult to find relevant words from corpus.

b. The GLoVE embedding will not return the same vector for "running" in both cases as the representation it uses are not independent like one-hot encoding but rather provides a distributed representation with dependency on context words such as "presidency" and "park" coming into play. It uses conditional probability ratios to represent relationships with different words.

c. We provide two explanations for this part, firstly theoretical and secondly empirical.

   Theoretical (expected):

   Consider the vector diagram shown in Figure 8, showing the 'king', 'queen', 'husband' and 'wife' vectors.

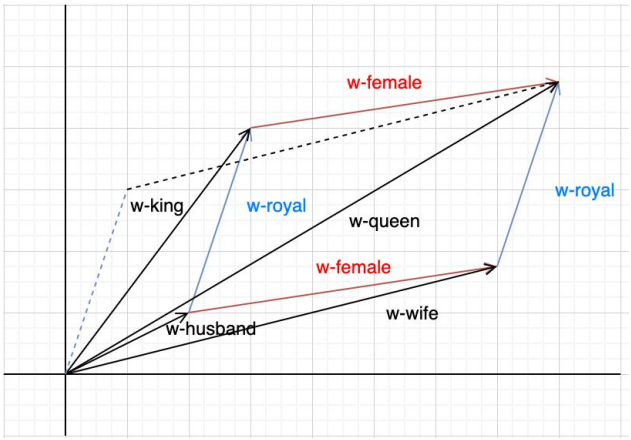


Figure 8: Plot of the embedding vectors for 'queen', 'king', 'husband' and 'wife' and their classical connections in 2D (ideal case)

As shown in Figure 8, ideally, **$||queen - king||_2 \approx ||wife - husband||_2$**, as

$$||queen-king||_2 \approx ||wife - husband||_2 \approx ||female||_2$$

This is because a possible contextual word embedding transforming both "king" to "queen" and "husband" to "wife" is female. Thus, $||queen - king||_2 \approx ||wife - husband||_2 \approx ||female||_2$. Using similar contextual word embedding "royal", we can also observe that:

$$||royal + wife||_2 \approx ||king-husband+wife||_2 \approx ||queen||_2$$

As a result, from the above equation, ideally:

$$||king-husband+wife-queen||_2 \approx \mathbf{||queen-king-wife+husband||_2 \approx 0}$$

The question that remains is why such vector arithmetic in the embedding space works for analogies like "king is to queen as husband is to wife", while not being specifically trained to do so. This has been explained intuitively in [1]. Semantic relationships between words can be thought of as having geometric relationships in the embedding space, with linguistic patterns being represented as linear relations between word vectors. It has been shown that the learned embeddings are equivalent to row projections of the pointwise mutual information (PMI) matrix, which is given as:

$$\mathrm{PMI}(w_i, c_j) = \log \frac{p(w_i, c_j)}{p(w_i)p(c_j)}$$

where, w_i are target words across a corpus and c_j are context words around the target words in the corpus. Using the concept of paraphrases (semantically interchangeable words), which are in turn equivalent to word transformations (which mirrors simple algebra), the authors proved that:

$$\mathrm{PMI}_{king} - \mathrm{PMI}_{man} + \mathrm{PMI}_{woman} \approx \mathrm{PMI}_{queen}$$

The above equation is sufficient to prove additive relationship in the embedding space. In other words, analogies are similar to word transformations describing semantic difference between words, while word transformations themselves are paraphrases which leads to the geometric relationship between PMI vectors as shown above. The authors showed that the relationship holds under low-rank projection of the PMI matrix in the embedding space for GLoVE and Word2Vec.

Reference:

[1]. Allen, C. & Hospedales, T.. (2019). Analogies Explained: Towards Understanding Word Embeddings. *Proceedings of the 36th International Conference on Machine Learning*, in PMLR 97:223-231.

Empirical:

We loaded the glove embeddings using the code provided in the question. Then we performed the arithmetic operations listed in the question. The results are as follows:

- $||queen - king - wife + husband||_2 = 6.165036$
- $||queen - king||_2 = 5.966258$
- $||wife - husband||_2 = 3.1520464$

To explain the results, we converted the GLoVE embeddings to Word2Vec format and loaded the resulting vocabulary into a keyed vector model from Gensim. Then we calculated the similarity score

between ‘queen’ and ‘king’ and then ‘husband’ and ‘wife’ by finding out the closest word embedding of ‘king’ and ‘wife’. The results are as follows:

- ‘king’: most similar word is ‘queen’, with a similarity score of 0.6336
- ‘wife’: most similar word is ‘husband’, with a similarity score of 0.8646

Since the word ‘wife’ is more analogous to ‘husband’ compared to ‘king’ with ‘queen’, we can expect the L2 norm of the vector (‘queen’ - ‘king’) to be greater than the L2 norm of (‘wife’ - ‘husband’). In other words, the spatial distance of ‘king’ from ‘queen’ is more than ‘husband’ and ‘wife’. Now to explain $||\text{queen - king - wife + husband}||_2$, consider the vector diagram shown Figure 9.

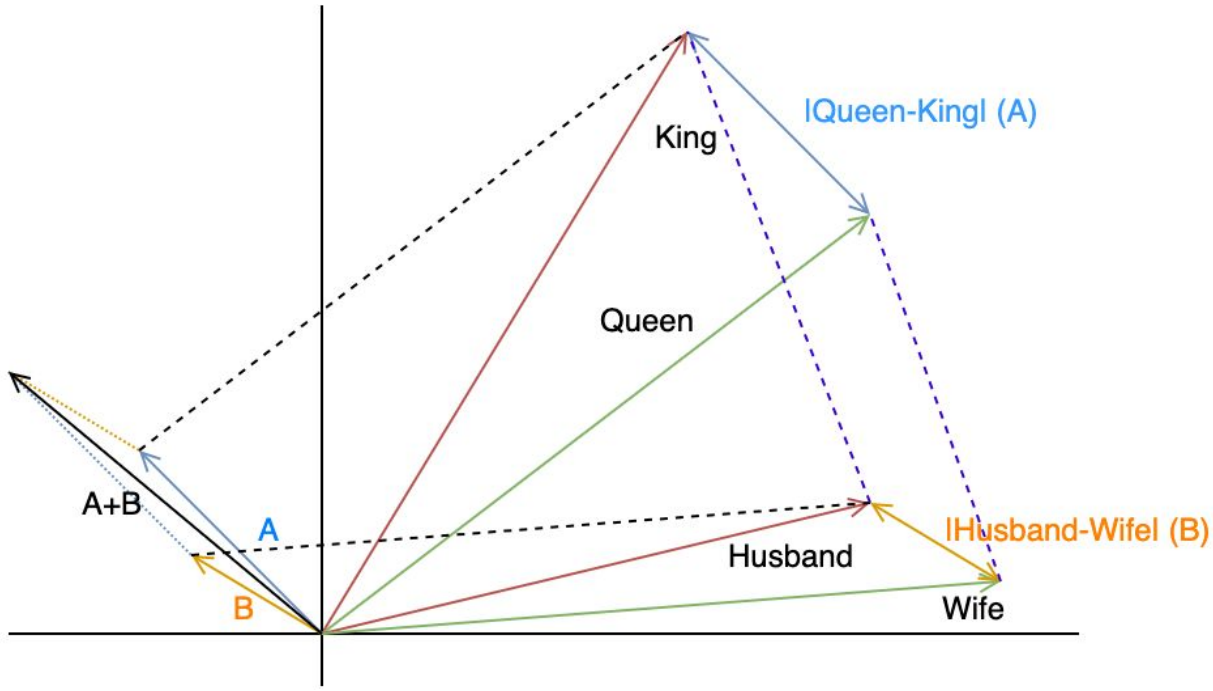


Figure 9: Plot of the embedding vectors for ‘queen’, ‘king’, ‘husband’ and ‘wife’ in 2D (empirical and drawn to scale)

From Figure 9, we observe that the sum of the vectors A and B is slightly larger than A and B (closer to A as A is larger than B). This explains why $||\text{queen - king - wife + husband}||_2$ is almost the same as $||\text{queen - king}||_2$. This goes to show that in practice, analogical word embeddings do not form perfect parallelograms. In fact, when we tried to find the closest embedding to queen-king+husband, we received husband and not wife, although the second closest match was indeed wife.

d. Stemming crudely removes the last few characters of inflectional words with the aim of simplifying the words to their base forms, without taking context into account or performing any morphological analysis. This can often lead to words without any meaning or correct spelling. Lemmatization on the other hand, considers the context, semantics and syntax, as well as parts-of-speech tokens to convert derivatives into meaningful root-forms called lemmas with the help of a corpus. Although stemming is faster than lemmatization, stemming usually has a much higher error rate than lemmatizers (but it depends on application too, sometimes stemming can yield better results). Thus, it is better to choose

lemmatization over stemming if absolutely required (e.g. for making the input space sparser) *to ensure the broader context of words is passed onto the GLoVE embeddings*. However, it has been pointed out in [2] that manual lemmatization does not increase the performance of NLP classifiers for simple-morphology languages such as English when using word embeddings. For languages with rich-morphology (e.g. Russian), the performance increases only slightly but consistently. As a result, lemmatization itself is also not necessary given the large size and diversity of English words in GLoVE embeddings. Furthermore, we noticed that GLoVE is able to automatically lemmatize words without preprocessing.

Reference:

[2]. Kutuzov, Andrey, and Elizaveta Kuzmenko. "To Lemmatize or Not to Lemmatize: How Word Normalisation Affects ELMo Performance in Word Sense Disambiguation." *DL4NLP 2019. Proceedings of the First NLPL Workshop on Deep Learning for Natural Language Processing*, 30 September, 2019, University of Turku, Turku, Finland. No. 163. Linköping University Electronic Press, 2019.

Question 9:

a. The feature engineering process we followed to use pre-trained GLoVE embeddings within the given rules is as follows:
    - Use Gensim to convert sample pre-trained GLoVE embedding file to Word2Vec format.
    - Use a keyed vector model from Gensim to store the pre-trained word embeddings as word vectors, thus representing the embeddings as a standalone structure mapping keys (words) to embedding vectors. In our case, this yielded a vocabulary of shape (400000, 300).
    - Remove punctuation and numeric characters from the training set as described previously in Question 2. Note that this is optional and does not yield significant difference in test accuracy.
    - Define a custom vectorizer class using the keyed vector model that transforms the sentences in the training document into GLoVE features. The goal is to convert each sentence (or in this case, a newsgroup post) into a vector of length 300 (because the dimension of pre-trained GLoVE embedding file we chose is 300), yielding a matrix of size (4732, 300) for the training set, with each row referring to a sentence embedding describing the whole text segment within the dimension of the GLoVE embeddings. In other words, all the features within the vector describing the sentence is based on the pre-trained GLoVE embeddings. To do so, we use the average of the *i*th values in the word embedding of all of the words present within a text segment as hinted in the question to represent the *i*th sentence embedding. This helps highlight important topical words as indicated in the given rules for the question. Finally, we normalize the final vectors to aggregate the words into a single vector.

    The above feature engineering pipeline abides by all of the rules given in the question.

b. For this part of the question, we chose linear SVM as the classifier. We used 5-fold grid search cross validation to obtain the optimal value of γ, which was 1. The accuracy, precision, recall and F1-score for the classifier are reported in Table 6.

Table 6: Performance metrics for Linear SVM classifier with GLoVE embeddings

Metric	**Linear SVM classifier with GLoVE embeddings (**$\gamma = 1$)
Accuracy	0.9714285714285714
Recall	0.9735849056603774
Precision	0.9699248120300752
F1 score	0.9717514124293785

Question 10:

For this question, we retrained the linear SVM classifier in Question 9 ($\gamma = 1$) for various dimensions of word embeddings, namely 50, 100, 200 (as well as 300 from previous question). Figure 10 shows the plot for test accuracy vs. dimension of pre-trained GLoVE embedding.

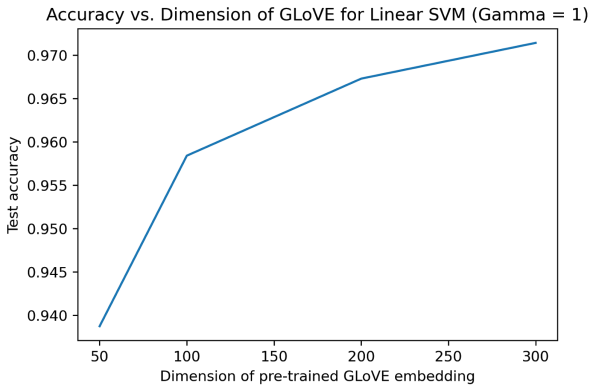


Figure 10: Plot of test accuracy vs. GLoVE embedding dimension for linear SVM classifier.

From Figure 10, we see that as the dimension of pre-trained GLoVE embedding increases, the test accuracy increases. This is expected because larger dimensions of GLoVE embedding capture much more semantic and complex information about distributed representation of words and co-occurrence probability ratios between target and context words in the latent space, which in turn provides more distinguishing features for the classifier to work on.

Question 11:

In this question, we are asked to visualize normalized GLoVE based features for the documents (2 classes: computer technology and recreational activity) on a 2D plane, along with a set of normalized random vectors the same size as GLoVE. Figure 11 (left) shows the plot for GLoVE features in 2D generated using the UMAP library, while Figure 11 (right) shows the plot for random vectors. We kept the dimensions of GLoVE embeddings to 300. We used the Euclidean distance metric in UMAP, with 15 neighbours being used to approximate the geodesic distance with effective minimum distance between embedded points being set to 0.1. For generating random variables, we drew points from Gaussian distribution with mean 0 and standard deviation 1 and then normalized by dividing the matrix with its norm.

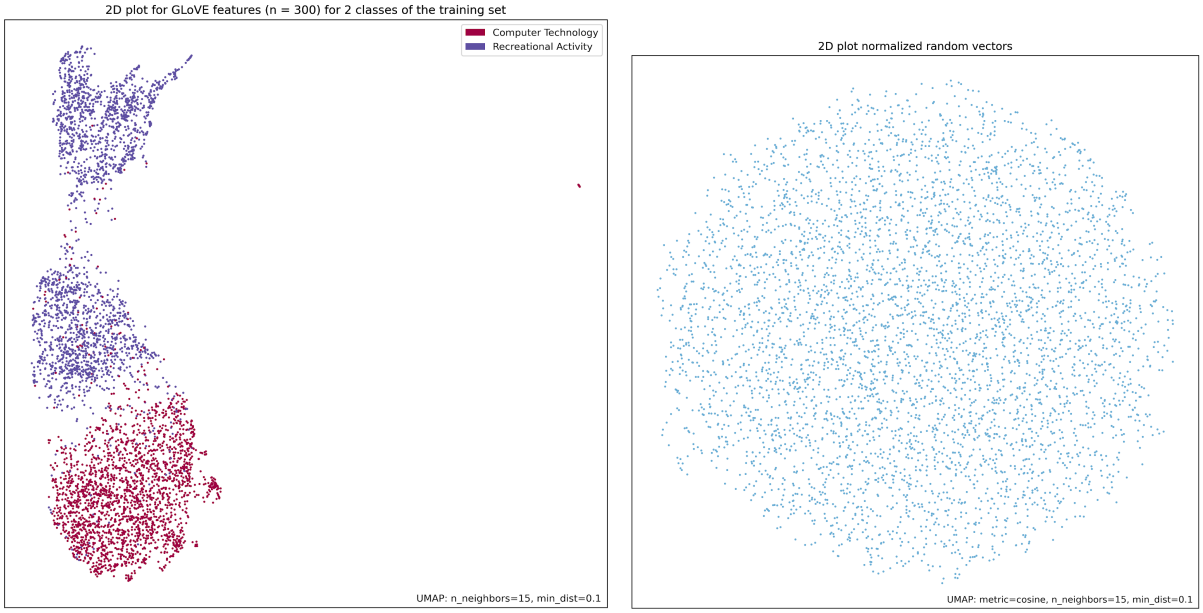


Figure 11: (Left) 2D visualization of GLoVE embeddings (n = 300) for two classes in the newsgroup dataset. (Right) 2D visualization of random vectors of the same dimension as GLoVE embeddings.

From Figure 11, we can see clusters forming only in the plot for the actual dataset. This is a good sign, because it means that the GLoVE embeddings are providing enough distinguishing features for classifiers to learn. As expected, we do not see any clusters forming in randomly generated points and all the points are evenly distributed.

Question 12:

In this question, we are asked to perform multiclass classification with linear SVM (one-vs-one and one-vs-all) and Naive Bayes and compare the performance metrics for 4 classes. We kept headers and footers, performed lemmatization (`min_df = 3`) with punctuation and number removal and extracted TF-IDF matrix with LSI dimensionality reduction for the training set.

For Naive Bayes, we decided to stick with Gaussian Naive Bayes.

For SVM, there are two approaches to work with multiclass classification.

- In one-vs-one classification for n class labels, the entire dataset is partitioned into $n!$ binary datasets and $n!$ classifiers are trained on the dataset, with the decision fused via majority vote or confidence levels.
- In one-vs-rest classification for n class labels, the entire dataset is partitioned into n datasets, with each dataset containing a positive class and all other classes assigned as negative. n classifiers are trained on the entire dataset.

To obtain ideal value of γ for the SVM, we used 5-fold grid search cross validation. For 1-1 linear SVM, the best estimator had a γ of 10, while for 1-all linear SVM, the best estimator had a γ of 1.

Figure 12 shows the confusion matrices for all three multiclass classifiers. The accuracy, precision, recall and F1-score for the classifiers are reported in Table 7.

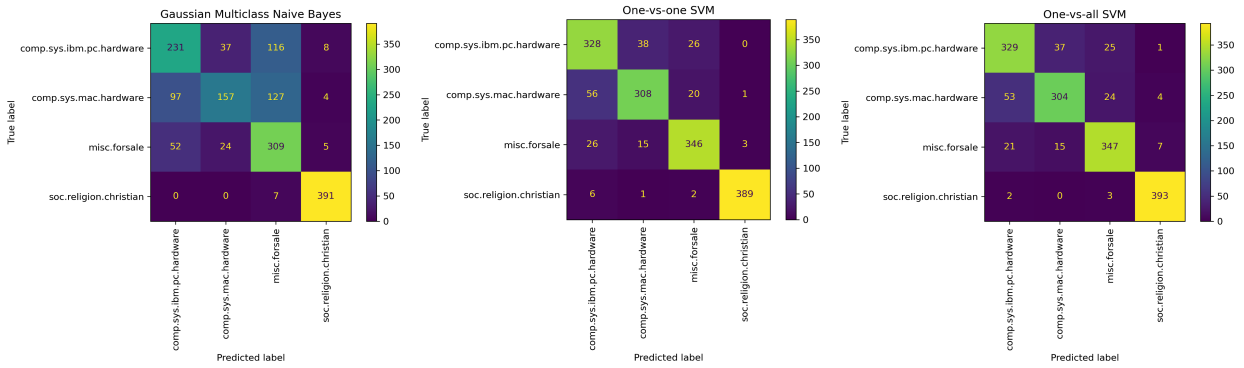


Figure 12: Confusion matrices for multiclass Gaussian Naive Bayes, 1-1 SVM and 1-all SVM (linear kernel)

Table 7: Performance metrics for multiclass Gaussian Naive Bayes, 1-1 SVM and 1-all SVM (linear kernel)

Metric	**Gaussian Naive Bayes**	**One-vs-one linear SVM ($\gamma = 10$)**	**One-vs-rest linear SVM ($\gamma = 1$)**
Accuracy	0.6952076677316293	0.876038338658147	0.8773162939297124

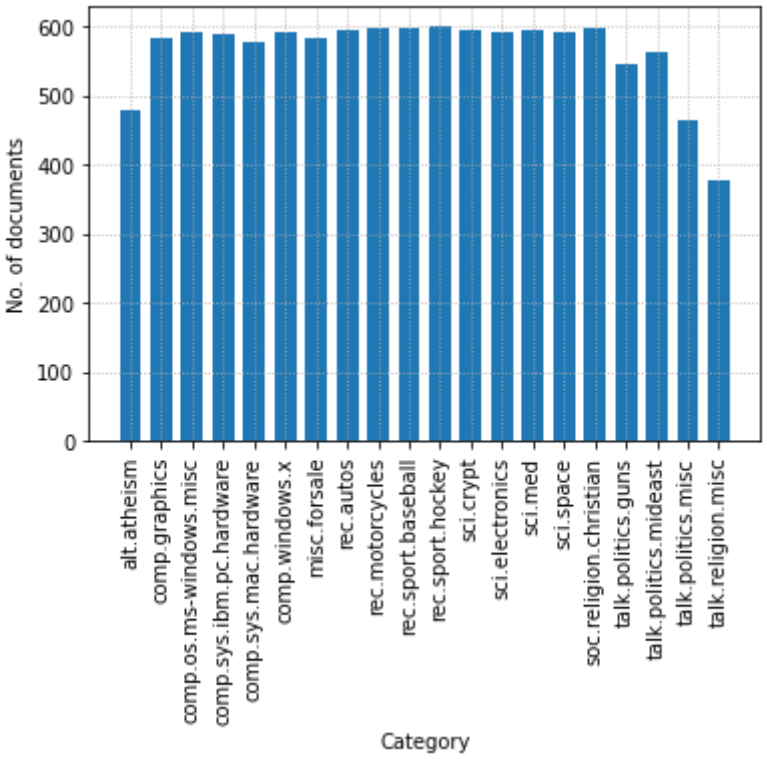
Recall	0.6952076677316293	0.876038338658147	0.8773162939297124
Precision	0.7109031572264817	0.8773689475462186	0.8770504094919451
F1 score	0.6870243376226386	0.8763158458707359	0.876771235927221

In [21]:
```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfTrans
from sklearn.decomposition import TruncatedSVD, NMF, randomized_svd
from sklearn.metrics import auc, roc_curve, plot_roc_curve, plot_confus
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from scipy import spatial
from gensim.scripts.glove2word2vec import glove2word2vec
from gensim.models import KeyedVectors
from tempfile import mkdtemp
from shutil import rmtree
from joblib import Memory
from matplotlib import pyplot as plt
import numpy as np
import random
from nltk.stem import WordNetLemmatizer
from nltk import pos_tag, word_tokenize
from nltk.corpus import wordnet
import nltk
import string
from string import punctuation
import os
import pandas as pd
import umap
import umap.plot
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
np.random.seed(42)
random.seed(42)
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/swapnilsayansaha/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/swapnilsayansaha/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /Users/swapnilsayansaha/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Question 1

In [2]:
```
newsgroups_train = fetch_20newsgroups(subset='train')
u, inv = np.unique(newsgroups_train.target, return_inverse=True)
plt.bar(u, np.bincount(inv), width=0.7)
locs, labels = plt.xticks()
plt.grid(linestyle=':')
plt.xticks(np.arange(20), np.array(newsgroups_train.target_names),rotat
plt.ylabel('No. of documents')
plt.xlabel('Category')
plt.savefig('Q1.png',dpi=300,bbox_inches='tight')
plt.show()
```



Question 2

In [3]:
```
categories = ['comp.graphics','comp.os.ms-windows.misc','comp.sys.ibm.p
              'comp.sys.mac.hardware','rec.autos','rec.motorcycles','re
train_dataset = fetch_20newsgroups(subset = 'train', categories = categ
test_dataset = fetch_20newsgroups(subset = 'test', categories = categor
```

In [4]:
```
lemmatizer = WordNetLemmatizer()
vectorizer = CountVectorizer(stop_words='english',min_df=3)
tfidf_transformer = TfidfTransformer()

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

def lemmatizer_func(sentence):
    lemmatized_sen = []
    lemma_list = [lemmatizer.lemmatize(word.lower(), pos=penn2morphy(ta
            for word, tag in pos_tag(word_tokenize(sentence))]
    for lemma in lemma_list:
        if (not any(char in lemma for char in punctuation) and not any(
            lemmatized_sen.append(lemma.lower())
    return ' '.join(lemmatized_sen)
```

In [5]:
```
train_data_proc = []
test_data_proc = []
for i in range(len(train_dataset.data)):
    train_data_proc.append(lemmatizer_func(train_dataset.data[i]))
for i in range(len(test_dataset.data)):
    test_data_proc.append(lemmatizer_func(test_dataset.data[i]))

train_data_feat_vec = vectorizer.fit_transform(train_data_proc)
test_data_feat_vec = vectorizer.transform(test_data_proc)
train_data_feat = tfidf_transformer.fit_transform(train_data_feat_vec)
test_data_feat = tfidf_transformer.transform(test_data_feat_vec)
```

In [6]:
```
print(train_data_feat.shape)
print(test_data_feat.shape)
```

```
(4732, 12161)
(3150, 12161)
```

Question 3

In [7]:
```
svd = TruncatedSVD(n_components=50, random_state=42)
nmf = NMF(n_components=50, init='random', random_state=42)

train_data_LSI = svd.fit_transform(train_data_feat)
print('LSI Train Data Shape:', train_data_LSI.shape)
train_data_NMF = nmf.fit_transform(train_data_feat)
print('NMF Train Data Shape:', train_data_NMF.shape)
U_tr,S_tr,V_tr = randomized_svd(train_data_feat,n_components=50,random_
print('LSI (train) error:',np.sum(np.array(train_data_feat - (U_tr.dot(
print('NMF (train) error:',np.sum(np.array(train_data_feat - train_data

test_data_LSI = svd.transform(test_data_feat)
print('LSI Test Data Shape:', test_data_LSI.shape)
test_data_NMF = nmf.transform(test_data_feat)
print('NMF Test Data Shape:', test_data_NMF.shape)
U_te,S_te,V_te = randomized_svd(test_data_feat,n_components=50,random_s
print('LSI (test) error:',np.sum(np.array(test_data_feat - (U_te.dot(np
print('NMF (test) error:',np.sum(np.array(test_data_feat - test_data_NM
```

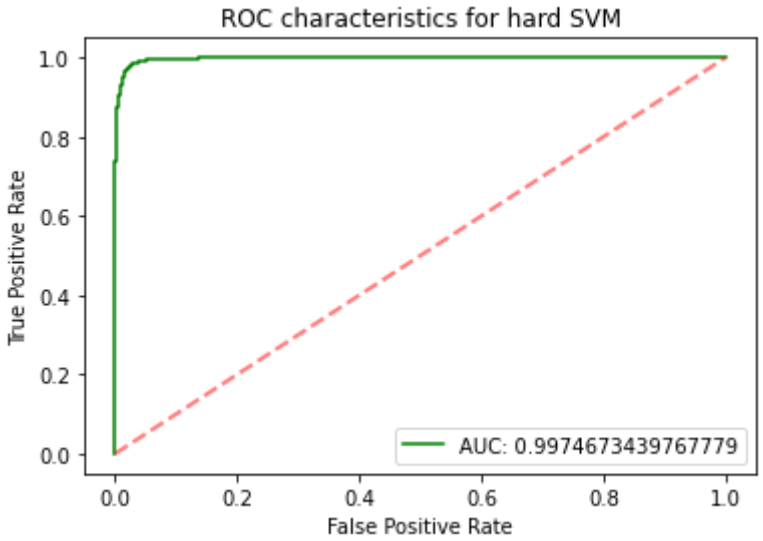
```
LSI Train Data Shape: (4732, 50)
NMF Train Data Shape: (4732, 50)
LSI (train) error: 4103.0367390859365
NMF (train) error: 4147.199485161415
LSI Test Data Shape: (3150, 50)
NMF Test Data Shape: (3150, 50)
LSI (test) error: 2643.0475837323024
NMF (test) error: 2858.9878541431635
```

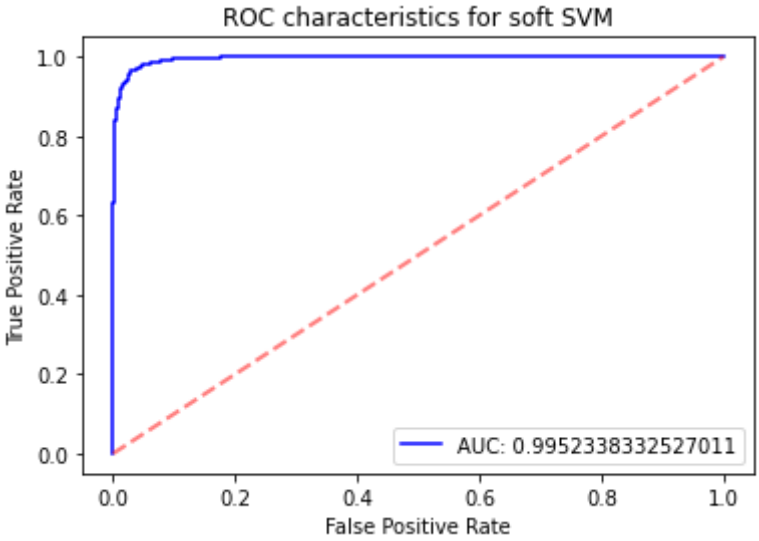
Question 4

In [8]:
```
y_train = []
y_test = []
for label in train_dataset.target:
    if label < 4:
        y_train.append(0)
    else:
        y_train.append(1)
for label in test_dataset.target:
    if label < 4:
        y_test.append(0)
    else:
        y_test.append(1)
```

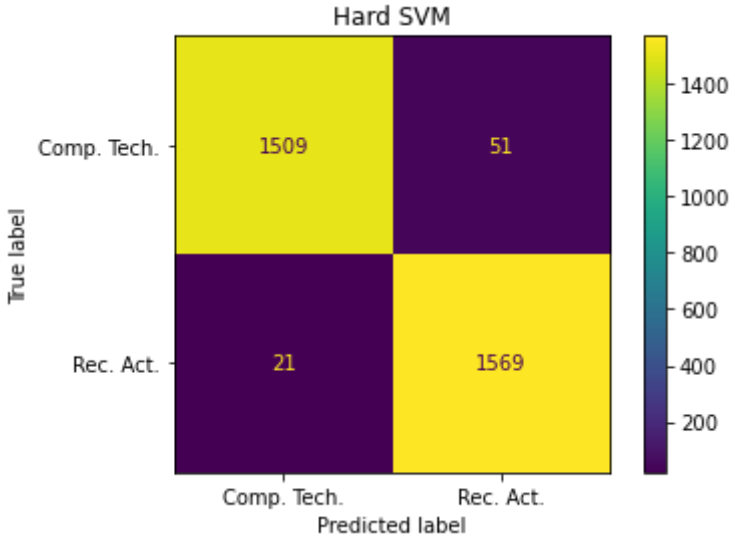
In [9]:
```
clf_hard = svm.SVC(kernel='linear',C=1000,random_state=42)
clf_soft = svm.SVC(kernel='linear',C=0.0001,random_state=42)
pred_hard = clf_hard.fit(train_data_LSI, y_train).predict(test_data_LSI
pred_soft = clf_soft.fit(train_data_LSI, y_train).predict(test_data_LSI
```

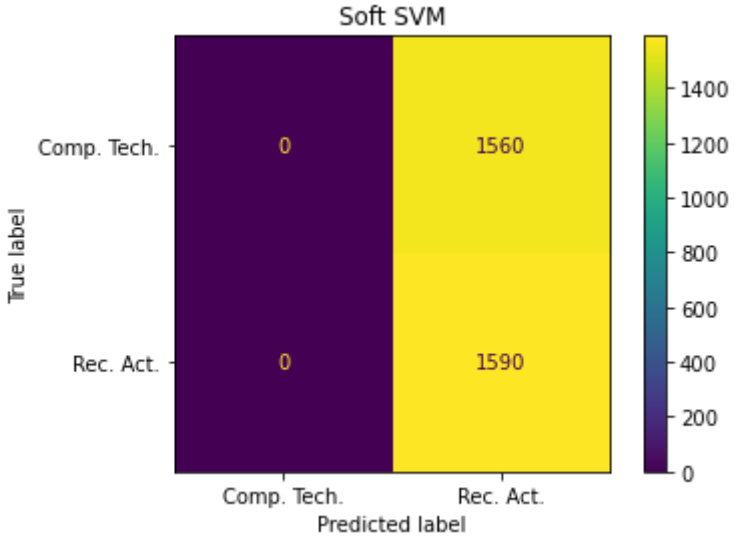
In [56]:
```
fig, ax = plt.subplots()
fpr, tpr, _ = roc_curve(y_test, clf_hard.decision_function(test_data_LS
plot_roc_curve(clf_hard, test_data_LSI, y_test, ax=ax, color='g',label=
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.5)
plt.title('ROC characteristics for hard SVM')
plt.savefig('Q41.png',dpi=300,bbox_inches='tight')
plt.show()


fig, ax = plt.subplots()
fpr, tpr, _ = roc_curve(y_test, clf_soft.decision_function(test_data_LS
plot_roc_curve(clf_soft, test_data_LSI, y_test, ax=ax, color='b',label=
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.5)
plt.title('ROC characteristics for soft SVM')
plt.savefig('Q42.png',dpi=300,bbox_inches='tight')
plt.show()
```




ROC characteristics for soft SVM
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC: 0.9952338332527011

In [28]:
```
plot_confusion_matrix(clf_hard, test_data_LSI, y_test,display_labels=['
plt.title('Hard SVM')
plt.savefig('Q43.png',dpi=300,bbox_inches='tight')
plt.show()
plot_confusion_matrix(clf_soft, test_data_LSI, y_test,display_labels=['
plt.title('Soft SVM')
plt.savefig('Q44.png',dpi=300,bbox_inches='tight')
plt.show()
```





In [29]:
```
print("Accuracy (hard SVM):", accuracy_score(y_test,pred_hard))
print("Recall (hard SVM):", recall_score(y_test,pred_hard))
print("Precision (hard SVM):", precision_score(y_test,pred_hard))
print("F1-Score (hard SVM):", f1_score(y_test,pred_hard))
print("Accuracy (soft SVM):", accuracy_score(y_test,pred_soft))
print("Recall (soft SVM):", recall_score(y_test,pred_soft))
print("Precision (soft SVM):", precision_score(y_test,pred_soft))
print("F1-Score (soft SVM):", f1_score(y_test,pred_soft))
```

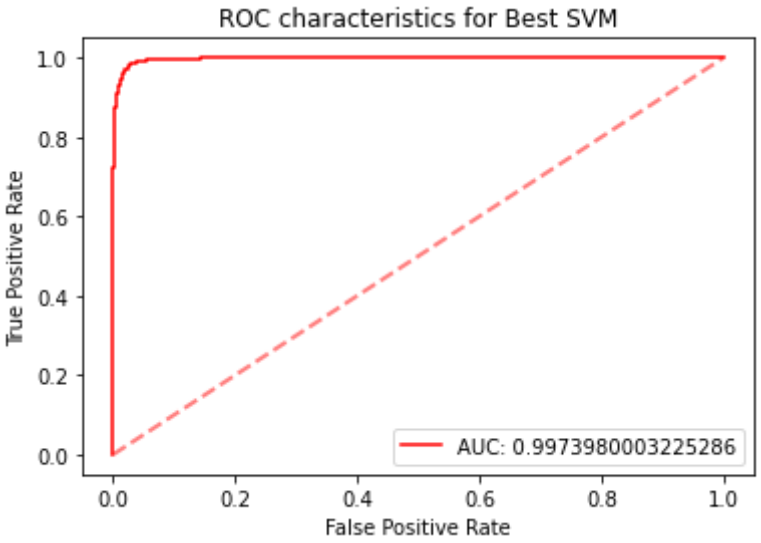
```
Accuracy (hard SVM): 0.9771428571428571
Recall (hard SVM): 0.9867924528301887
Precision (hard SVM): 0.9685185185185186
F1-Score (hard SVM): 0.977570093457944
Accuracy (soft SVM): 0.5047619047619047
Recall (soft SVM): 1.0
Precision (soft SVM): 0.5047619047619047
F1-Score (soft SVM): 0.6708860759493671
```

In [30]:
```
clf_cv = svm.SVC(random_state=42)
param_grid = {'C': [0.001,0.01,0.1,1,10,100,200,400,600,800,1000],
              'kernel': ['linear']}
grid = GridSearchCV(clf_cv,param_grid,cv=5,scoring='accuracy')
grid.fit(train_data_LSI,y_train)
pred_cv = grid.best_estimator_.predict(test_data_LSI)
```

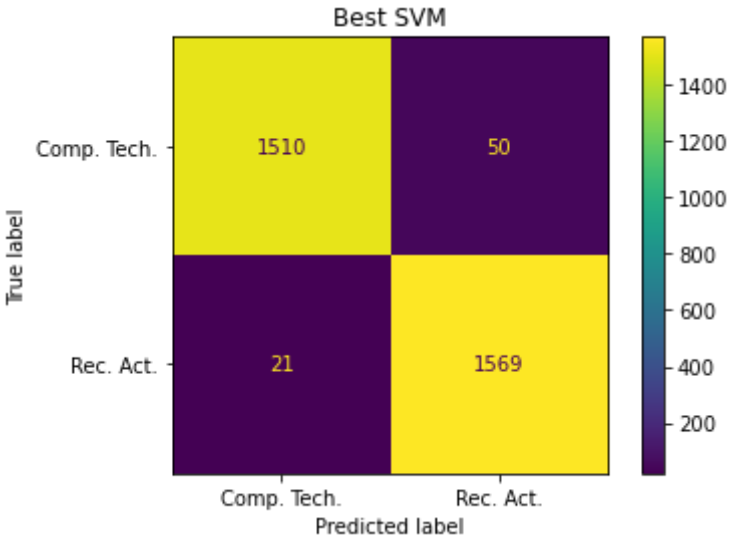
In [31]:
```
print('Best Value of gamma:',grid.best_params_['C'])
for l, n in zip(param_grid['C'],grid.cv_results_['mean_test_score']):
    print(f'Gamma: {l}\t',f'Avg. Validation Accuracy: {n}')
```

```
Best Value of gamma: 400
Gamma: 0.001     Avg. Validation Accuracy: 0.504860570043154
Gamma: 0.01      Avg. Validation Accuracy: 0.508875697373033
Gamma: 0.1       Avg. Validation Accuracy: 0.9632284883162809
Gamma: 1         Avg. Validation Accuracy: 0.9708359099950663
Gamma: 10        Avg. Validation Accuracy: 0.976330283012339
Gamma: 100       Avg. Validation Accuracy: 0.9759070035340264
Gamma: 200       Avg. Validation Accuracy: 0.9767522229986314
Gamma: 400       Avg. Validation Accuracy: 0.977174832730934
Gamma: 600       Avg. Validation Accuracy: 0.9771746094822641
Gamma: 800       Avg. Validation Accuracy: 0.9769634162404477
Gamma: 1000      Avg. Validation Accuracy: 0.9771746094822641
```

In [55]:
```
fig, ax = plt.subplots()
fpr, tpr, _ = roc_curve(y_test, grid.best_estimator_.decision_function(
plot_roc_curve(grid.best_estimator_, test_data_LSI, y_test, ax=ax, colo
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.5)
plt.title('ROC characteristics for Best SVM')
plt.savefig('Q45.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [33]:
```
plot_confusion_matrix(grid.best_estimator_, test_data_LSI, y_test,displ
plt.title('Best SVM')
plt.savefig('Q46.png',dpi=300,bbox_inches='tight')
plt.show()
```



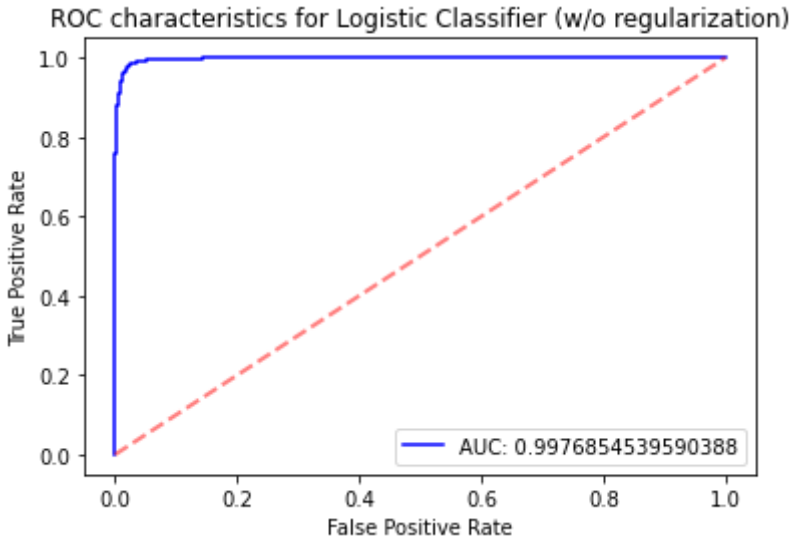
In [34]:
```
print("Accuracy (best SVM):", accuracy_score(y_test,pred_cv))
print("Recall (best SVM):", recall_score(y_test,pred_cv))
print("Precision (best SVM):", precision_score(y_test,pred_cv))
print("F1-Score (best SVM):", f1_score(y_test,pred_cv))
```

```
Accuracy (best SVM): 0.9774603174603175
Recall (best SVM): 0.9867924528301887
Precision (best SVM): 0.9691167387276096
F1-Score (best SVM): 0.9778747273293862
```

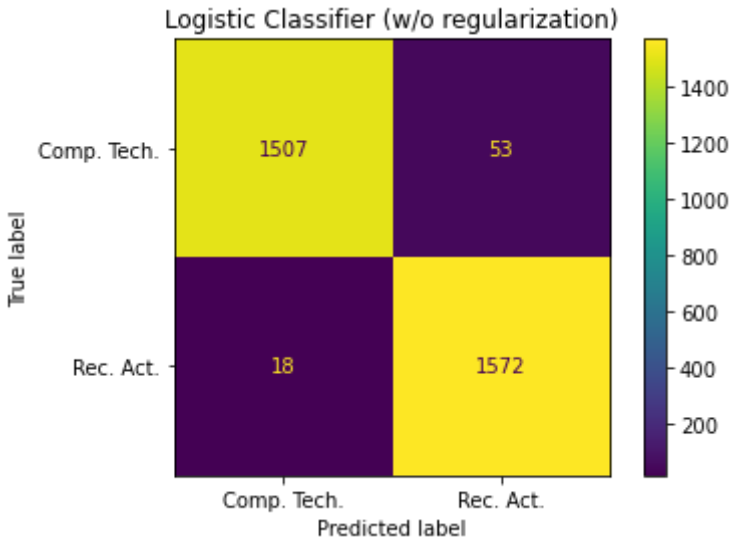
Question 5

In [35]:
```
clf_lr_wor = LogisticRegression(C=1000000,random_state=42,max_iter=1000
pred_lr_wor = clf_lr_wor.fit(train_data_LSI,y_train).predict(test_data_
```

In [54]:
```
fig, ax = plt.subplots()
fpr, tpr, _ = roc_curve(y_test, clf_lr_wor.decision_function(test_data_
plot_roc_curve(clf_lr_wor, test_data_LSI, y_test, ax=ax, color='b',labe
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.5)
plt.title('ROC characteristics for Logistic Classifier (w/o regularizat
plt.savefig('Q51.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [37]:
```
plot_confusion_matrix(clf_lr_wor, test_data_LSI, y_test,display_labels=
plt.title('Logistic Classifier (w/o regularization)')
plt.savefig('Q52.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [38]:
```
print("Accuracy (Logistic Classifier - w/o regularization):", accuracy_
print("Recall (Logistic Classifier - w/o regularization):", recall_scor
print("Precision (Logistic Classifier - w/o regularization):", precisio
print("F1-Score (Logistic Classifier - w/o regularization):", f1_score(
```

```
Accuracy (Logistic Classifier - w/o regularization): 0.9774603174603175
Recall (Logistic Classifier - w/o regularization): 0.9886792452830189
Precision (Logistic Classifier - w/o regularization): 0.9673846153846154
F1-Score (Logistic Classifier - w/o regularization): 0.9779160186625194
```

In [39]:
```
clf_lr_l1 = LogisticRegression(penalty='l1',random_state=42,solver='lib
param_grid = {'C': [0.001,0.01,0.1,1,10,100,200,400,600,800,1000]}
grid_l1 = GridSearchCV(clf_lr_l1,param_grid,cv=5,scoring='accuracy')
grid_l1.fit(train_data_LSI,y_train)
pred_cv_lr_l1 = grid_l1.best_estimator_.predict(test_data_LSI)

clf_lr_l2 = LogisticRegression(penalty='l2',solver='liblinear',random_s
grid_l2 = GridSearchCV(clf_lr_l2,param_grid,cv=5,scoring='accuracy')
grid_l2.fit(train_data_LSI,y_train)
pred_cv_lr_l2 = grid_l2.best_estimator_.predict(test_data_LSI)
```

In [40]:
```
print('Best Value of L1 Regularization Parameter:',grid_l1.best_params_
for l, n in zip(param_grid['C'],grid_l1.cv_results_['mean_test_score'])
    print(f'L1 Reg. Param.: {l}\t',f'Avg. Validation Accuracy: {n}')

print('Best Value of L2 Regularization Parameter:',grid_l2.best_params_
for l, n in zip(param_grid['C'],grid_l2.cv_results_['mean_test_score'])
    print(f'L2 Reg. Param.: {l}\t',f'Avg. Validation Accuracy: {n}')
```

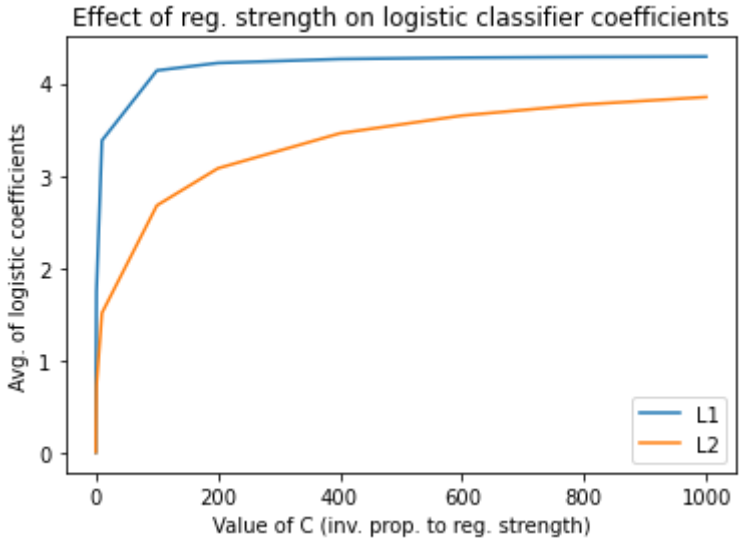
```
Best Value of L1 Regularization Parameter: 10
L1 Reg. Param.: 0.001    Avg. Validation Accuracy: 0.495139429956846
L1 Reg. Param.: 0.01     Avg. Validation Accuracy: 0.9363890867120158
L1 Reg. Param.: 0.1      Avg. Validation Accuracy: 0.9518178022954429
L1 Reg. Param.: 1        Avg. Validation Accuracy: 0.9702023302696174
L1 Reg. Param.: 10       Avg. Validation Accuracy: 0.9780204986928791
L1 Reg. Param.: 100      Avg. Validation Accuracy: 0.977174832730934
L1 Reg. Param.: 200      Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 400      Avg. Validation Accuracy: 0.9775976657119065
L1 Reg. Param.: 600      Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 800      Avg. Validation Accuracy: 0.9773862492214203
L1 Reg. Param.: 1000     Avg. Validation Accuracy: 0.9773862492214203
Best Value of L2 Regularization Parameter: 400
L2 Reg. Param.: 0.001    Avg. Validation Accuracy: 0.7499996651269951
L2 Reg. Param.: 0.01     Avg. Validation Accuracy: 0.9448437370934363
L2 Reg. Param.: 0.1      Avg. Validation Accuracy: 0.9636519910432634
L2 Reg. Param.: 1        Avg. Validation Accuracy: 0.969145917563196
L2 Reg. Param.: 10       Avg. Validation Accuracy: 0.9742176808481663
L2 Reg. Param.: 100      Avg. Validation Accuracy: 0.9769636394891178
L2 Reg. Param.: 200      Avg. Validation Accuracy: 0.977174832730934
L2 Reg. Param.: 400      Avg. Validation Accuracy: 0.9773860259727503
L2 Reg. Param.: 600      Avg. Validation Accuracy: 0.9771746094822639
L2 Reg. Param.: 800      Avg. Validation Accuracy: 0.9771746094822639
L2 Reg. Param.: 1000     Avg. Validation Accuracy: 0.9769634162404477
```

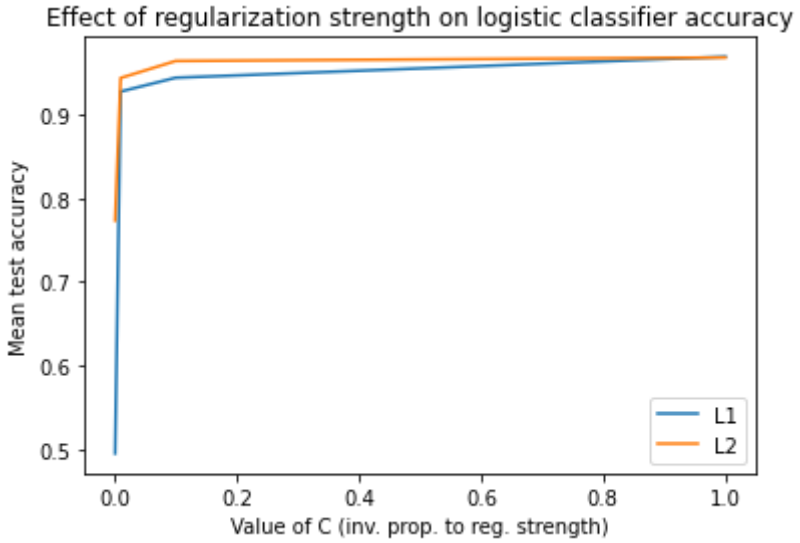
In [41]:
```
print("Accuracy (best logistic classifer with L1 regularization):", acc
print("Recall (best logistic classifer with L1 regularization):", recal
print("Precision (best logistic classifer with L1 regularization):", pr
print("F1-Score (best logistic classifer with L1 regularization):", f1_
print("Accuracy (best logistic classifer with L2 regularization):", acc
print("Recall (best logistic classifer with L2 regularization):", recal
print("Precision (best logistic classifer with L2 regularization):", pr
print("F1-Score (best logistic classifer with L2 regularization):", f1_
```

```
Accuracy (best logistic classifer with L1 regularization): 0.976825396825
3968
Recall (best logistic classifer with L1 regularization): 0.98867924528301
89
Precision (best logistic classifer with L1 regularization): 0.96619545175
16902
F1-Score (best logistic classifer with L1 regularization): 0.977308050979
1732
Accuracy (best logistic classifer with L2 regularization): 0.976507936507
9365
Recall (best logistic classifer with L2 regularization): 0.98867924528301
89
Precision (best logistic classifer with L2 regularization): 0.96560196560
19657
F1-Score (best logistic classifer with L2 regularization): 0.977004350528
2784
```

In [42]:
```
C_list = [0.001,0.01,0.1,1,10,100,200,400,600,800,1000]
accu_coeff_l1 = []
mean_coeff_l1 = []
accu_coeff_l2 = []
mean_coeff_l2 = []
for j in C_list:
    clf_lr_l1_coeff = LogisticRegression(C=j,penalty='l1',random_state=
    pred_lr_l1_coeff = clf_lr_l1_coeff.fit(train_data_LSI,y_train).pred
    accu_coeff_l1.append(accuracy_score(y_test,pred_lr_l1_coeff))
    mean_coeff_l1.append(np.mean(clf_lr_l1_coeff.coef_))
    clf_lr_l2_coeff = LogisticRegression(C=j,penalty='l2',random_state=
    pred_lr_l2_coeff = clf_lr_l2_coeff.fit(train_data_LSI,y_train).pred
    accu_coeff_l2.append(accuracy_score(y_test,pred_lr_l2_coeff))
    mean_coeff_l2.append(np.mean(clf_lr_l2_coeff.coef_))
```

In [43]:
```
fig, ax = plt.subplots()
plt.title('Effect of reg. strength on logistic classifier coefficients'
plt.plot(C_list,mean_coeff_l1,label='L1')
plt.plot(C_list,mean_coeff_l2,label='L2')
plt.xlabel('Value of C (inv. prop. to reg. strength)')
plt.ylabel('Avg. of logistic coefficients')
plt.legend()
plt.savefig('Q53.png',dpi=300,bbox_inches='tight')
plt.show()

fig, ax = plt.subplots()
plt.title('Effect of regularization strength on logistic classifier acc
plt.plot(C_list[0:4],accu_coeff_l1[0:4],label='L1')
plt.plot(C_list[0:4],accu_coeff_l2[0:4],label='L2')
plt.xlabel('Value of C (inv. prop. to reg. strength)')
plt.ylabel('Mean test accuracy')
plt.legend()
plt.savefig('Q54.png',dpi=300,bbox_inches='tight')
plt.show()
```





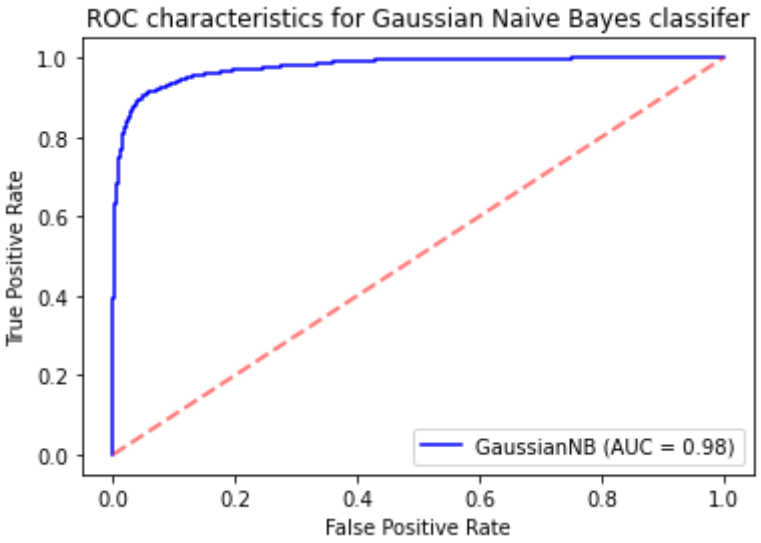
In [44]:
```
print(np.mean(grid_l1.best_estimator_.coef_))
print(np.mean(grid_l2.best_estimator_.coef_))
```

```
3.3813965396277426
3.4593770888580417
```

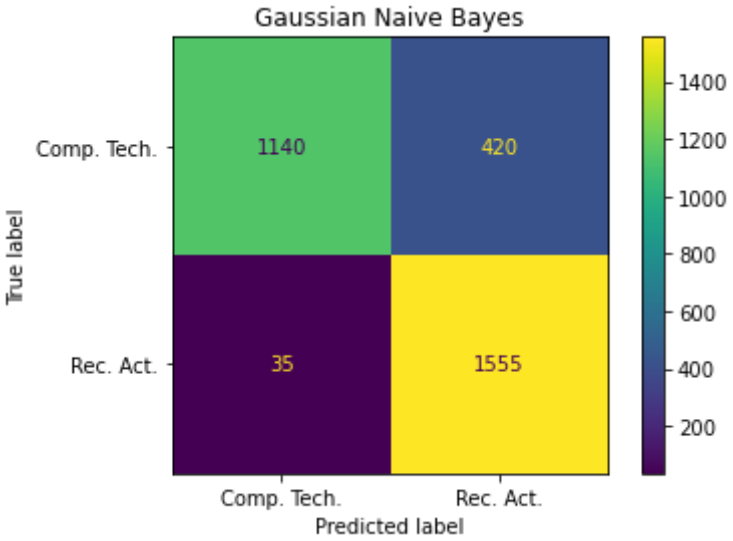
Question 6

In [45]:
```
clf_NB = GaussianNB()
pred_NB = clf_NB.fit(train_data_LSI, y_train).predict(test_data_LSI)
```

In [48]:
```
fig, ax = plt.subplots()
plot_roc_curve(clf_NB, test_data_LSI, y_test, ax=ax, color='b')
ax.plot([0, 1], [0, 1], linestyle='--', lw=2, color='r',
        label='Chance', alpha=.5)
plt.title('ROC characteristics for Gaussian Naive Bayes classifer')
plt.savefig('Q61.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [30]:
```
plot_confusion_matrix(clf_NB, test_data_LSI, y_test,display_labels=['Co
plt.title('Gaussian Naive Bayes')
plt.savefig('Q62.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [31]:
```
print("Accuracy (Gaussian Naive Bayes):", accuracy_score(y_test,pred_NB
print("Recall (Gaussian Naive Bayes):", recall_score(y_test,pred_NB))
print("Precision (Gaussian Naive Bayes):", precision_score(y_test,pred_
print("F1-Score (Gaussian Naive Bayes):", f1_score(y_test,pred_NB))
```

```
Accuracy (Gaussian Naive Bayes): 0.8555555555555555
Recall (Gaussian Naive Bayes): 0.9779874213836478
Precision (Gaussian Naive Bayes): 0.7873417721518987
F1-Score (Gaussian Naive Bayes): 0.8723702664796633
```

Question 7

In [32]:
```
def lemmatized(sentence):
    lemmatized_sen = []
    lemma_list = [lemmatizer.lemmatize(word.lower(), pos=penn2morphy(ta
                  for word, tag in pos_tag(word_tokenize(sentence))]
    for lemma in lemma_list:
        if (not any(char in lemma for char in punctuation) and not any(
            lemmatized_sen.append(lemma.lower())
    return lemmatized_sen

def non_lemmatized(sentence):
    non_lemmatized_sen = []
    lemma_list = word_tokenize(sentence)
    for lemma in lemma_list:
        if (not any(char in lemma for char in punctuation) and not any(
            non_lemmatized_sen.append(lemma.lower())
    return non_lemmatized_sen
```

In [33]:
```
cachedir = mkdtemp()
memory = Memory(location=cachedir, verbose=10)

pipeline = Pipeline([
    ('vect', CountVectorizer(stop_words='english')),
    ('tfidf', TfidfTransformer()),
    ('reduce_dim', None),
    ('clf', None),
],
memory=memory
)
param_grid = [
    {
        'vect__min_df': (3,5),
        'vect__analyzer': (lemmatized,non_lemmatized),
        'reduce_dim': (TruncatedSVD(n_components=50, random_state=42),
        'clf': (svm.SVC(kernel='linear',C=400,random_state=42),
                GaussianNB(),
                LogisticRegression(penalty='l1',C=10,random_state=42,so
                LogisticRegression(C=400,penalty='l2',random_state=42,s
    }
]
```

In [34]:
```
train_dataset_nhf = fetch_20newsgroups(subset = 'train', categories = c
test_dataset_nhf = fetch_20newsgroups(subset = 'test', categories = cat
y_train_nhf = []
y_test_nhf = []
for label in train_dataset_nhf.target:
    if label < 4:
        y_train_nhf.append(0)
    else:
        y_train_nhf.append(1)
for label in test_dataset_nhf.target:
    if label < 4:
        y_test_nhf.append(0)
    else:
        y_test_nhf.append(1)
```

In [35]:
```
grid_all_nhf = GridSearchCV(pipeline,cv=5,param_grid=param_grid,scoring
grid_all_nhf.fit(train_dataset_nhf.data, y_train_nhf)
```

```
_____
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer=<function lemmatized at 0x7
fb5c9c96710>, min_df=3,
                stop_words='english'),
[ 'Hank Greenberg was probably the greatest ever.  He was also subject
to a\n'
  'lot of heckling from bigots on the opposing teams and in the stands,
but\n'
  'it never seemed to affect his performance negatively.',
  'On March 21, 1993 Roger Maynard wrote (in reply to an article by Gra
ham\n'
  'Hudson):\n'
  '\n'
  '>> will still have the Jennings Trophy at the end of the year.  Potv
in is '
  'very\n'
  '>> good, and I do believe that he will be a star, but I want to see
```

In [36]:
```
print(grid_all_nhf.best_estimator_)
```

```
Pipeline(memory=Memory(location=/var/folders/k6/1m9hk1l17pqf2dl7bllzhg0c0
000gn/T/tmpbjnhap8p/joblib),
         steps=[('vect',
                 CountVectorizer(analyzer=<function lemmatized at 0x7fb5c
9c96710>,
                                 min_df=5, stop_words='english')),
                ('tfidf', TfidfTransformer()),
                ('reduce_dim', TruncatedSVD(n_components=50, random_state
=42)),
                ('clf',
                 LogisticRegression(C=400, random_state=42,
                                    solver='liblinear'))])
```

In [37]:
```
vectorizer_nhf = CountVectorizer(stop_words='english',min_df=5)
tfidf_transformer_nhf = TfidfTransformer()

train_data_nhf_proc = []
test_data_nhf_proc = []
for i in range(len(train_dataset_nhf.data)):
    train_data_nhf_proc.append(lemmatizer_func(train_dataset_nhf.data[i
for i in range(len(test_dataset_nhf.data)):
    test_data_nhf_proc.append(lemmatizer_func(test_dataset_nhf.data[i])

train_data_feat_nhf_vec = vectorizer_nhf.fit_transform(train_data_nhf_p
test_data_feat_nhf_vec = vectorizer_nhf.transform(test_data_nhf_proc)
train_data_feat_nhf = tfidf_transformer_nhf.fit_transform(train_data_fe
test_data_feat_nhf = tfidf_transformer_nhf.transform(test_data_feat_nhf
svd_nhf = TruncatedSVD(n_components=50, random_state=42)
train_data_LSI_nhf = svd_nhf.fit_transform(train_data_feat_nhf)
test_data_LSI_nhf = svd_nhf.transform(test_data_feat_nhf)
```

In [38]:
```
clf_best_nhf = LogisticRegression(penalty='l2',C=400,solver='liblinear'
pred_best_nhf = clf_best_nhf.fit(train_data_LSI_nhf,y_train_nhf).predic
print("Test accuracy of best classifier w/o heading and footer in data:
```

```
Test accuracy of best classifier w/o heading and footer in data: 0.966984
126984127
```

In [39]:
```
grid_all_whf = GridSearchCV(pipeline,cv=5,param_grid=param_grid,scoring
grid_all_whf.fit(train_dataset.data, y_train)
```

```
_____
_________
[Memory] Calling sklearn.pipeline._fit_transform_one...
_fit_transform_one(CountVectorizer(analyzer=<function lemmatized at 0x7
fb5c9c96710>, min_df=3,
                stop_words='english'),
[ 'Subject: Re: Jewish Baseball Players?\n'
  'From: rbd@flash.ece.uc.edu (Bobby Davis)\n'
  'Organization: University of Cincinnati\n'
  'NNTP-Posting-Host: flash.ece.uc.edu\n'
  'Lines: 5\n'
  '\n'
  'Hank Greenberg was probably the greatest ever.  He was also subject
to a\n'
  'lot of heckling from bigots on the opposing teams and in the stands,
but\n'
  'it never seemed to affect his performance negatively.\n'
  '\n'
  'Bob Davis\trbd@thor.ece.uc.edu\n',
```

In [40]:
```
print(grid_all_whf.best_estimator_)
```

```
Pipeline(memory=Memory(location=/var/folders/k6/1m9hk1l17pqf2dl7bllzhg0c0
000gn/T/tmpbjnhap8p/joblib),
         steps=[('vect',
                 CountVectorizer(analyzer=<function lemmatized at 0x7fb5c
9c96710>,
                                 min_df=3, stop_words='english')),
                ('tfidf', TfidfTransformer()),
                ('reduce_dim', TruncatedSVD(n_components=50, random_state
=42)),
                ('clf', SVC(C=400, kernel='linear', random_state=42))])
```

In [41]:
```
clf_best_whf = svm.SVC(kernel='linear',C=400,random_state=42)
pred_best_whf = clf_best_whf.fit(train_data_LSI, y_train).predict(test_
print("Test accuracy of best classifier with heading and footer in data
```

```
Test accuracy of best classifier with heading and footer in data: 0.97746
03174603175
```

Question 8 (Part C)

In [42]:
```
#Place the glove.6B.300d.txt file in a folder named glove in the projec
embeddings_dict = {}
dimension_of_glove = 300
with open("glove/glove.6B.300d.txt", 'r') as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], "float32")
        embeddings_dict[word] = vector
```

In [43]:
```
print(np.linalg.norm(embeddings_dict['queen']-embeddings_dict['king']-e
print(np.linalg.norm(embeddings_dict['queen']-embeddings_dict['king']))
print(np.linalg.norm(embeddings_dict['wife']-embeddings_dict['husband']


```

```
6.165036
5.966258
3.1520464
```

In [44]:
```
root_folder='.'
glove_folder_name='glove'
glove_filename='glove.6B.300d.txt'
glove_path = os.path.abspath(os.path.join(root_folder, glove_folder_nam
word2vec_output_file = glove_filename+'.word2vec'
glove2word2vec(glove_path, word2vec_output_file)
model = KeyedVectors.load_word2vec_format(word2vec_output_file, binary=
```

In [45]:
```
result = model.similar_by_word("king")
print("king is similar to {}: {:.4f}".format(*result[0]))
result = model.similar_by_word("wife")
print("wife is similar to {}: {:.4f}".format(*result[0]))
```

```
king is similar to queen: 0.6336
wife is similar to husband: 0.8646
```

In [46]:
```
def find_closest_embeddings(embedding):
    return sorted(embeddings_dict.keys(), key=lambda word: spatial.dist

print(find_closest_embeddings(embeddings_dict["queen"] - embeddings_dic
```

```
['husband', 'wife', 'mother', 'daughter', 'grandmother']
```

Question 9

In [47]:
```
def punc_num_remover(sentence):
    non_lemmatized_sen = []
    lemma_list = word_tokenize(sentence)
    for lemma in lemma_list:
        if (not any(char in lemma for char in punctuation) and not any(
            non_lemmatized_sen.append(lemma.lower())
    return ' '.join(non_lemmatized_sen)

X_train = []
X_test = []
for i in range(len(train_dataset.data)):
    X_train.append(punc_num_remover(train_dataset.data[i]))
for i in range(len(test_dataset.data)):
    X_test.append(punc_num_remover(test_dataset.data[i]))
```

In [48]:
```
#Reference:
#https://edumunozsala.github.io/BlogEms/jupyter/nlp/classification/embe
class Word2VecVectorizer:
    def __init__(self, model):
        print("Loading in word vectors...")
        self.word_vectors = model
        print("Finished loading in word vectors")

    def fit(self, data):
        pass

    def transform(self, data):
        v = self.word_vectors.get_vector('king')
        self.D = v.shape[0]

        X = np.zeros((len(data), self.D))
        n = 0
        emptycount = 0
        for sentence in data:
            tokens = sentence.split()
            vecs = []
            m = 0
            for word in tokens:
                try:
                    vec = self.word_vectors.get_vector(word)
                    vecs.append(vec)
                    m += 1
                except KeyError:
                    pass
            if len(vecs) > 0:
                vecs = np.array(vecs)
                X[n] = vecs.mean(axis=0)
            else:
                emptycount += 1
            n += 1
        print("Number of samples with no words found: %s / %s" % (empty
        return X

    def fit_transform(self, data):
        self.fit(data)
        return self.transform(data)
```

In [49]:
```
vectorizer = Word2VecVectorizer(model)
```

```
Loading in word vectors...
Finished loading in word vectors
```

In [50]:
```
Xtrain = vectorizer.fit_transform(X_train)
Ytrain = y_train
Xtest = vectorizer.transform(X_test)
Ytest = y_test
print(Xtrain.shape,Xtest.shape)
```

```
Number of samples with no words found: 0 / 4732
Number of samples with no words found: 0 / 3150
(4732, 300) (3150, 300)
```

In [51]:
```
clf_cv_Glove = svm.SVC(random_state=42)
param_grid = {'C': [0.001,0.01,0.1,1,10,100,200,400,600,800,1000],
              'kernel': ['linear']}
Glove_model = GridSearchCV(clf_cv_Glove,param_grid,cv=5,scoring='accura
y_pred_glove = Glove_model.best_estimator_.predict(Xtest)
```

In [52]:
```
print(Glove_model.best_estimator_)
```

```
SVC(C=1, kernel='linear', random_state=42)
```

In [53]:
```
print("Accuracy (Best GLoVE classifier):", accuracy_score(Ytest,y_pred_
print("Recall (Best GLoVE classifier):", recall_score(Ytest,y_pred_glov
print("Precision (Best GLoVE classifier):", precision_score(Ytest,y_pre
print("F1-Score (Best GLoVE classifier):", f1_score(Ytest,y_pred_glove)
```

```
Accuracy (Best GLoVE classifier): 0.9714285714285714
Recall (Best GLoVE classifier): 0.9735849056603774
Precision (Best GLoVE classifier): 0.9699248120300752
F1-Score (Best GLoVE classifier): 0.9717514124293785
```

Question 10

In [54]:
```
filenames_glove = ['glove.6B.50d.txt','glove.6B.100d.txt','glove.6B.200
accu_list_glove = []
for filename in filenames_glove:
    print('Training for: ', filename)
    glove_filename=filename
    glove_path = os.path.abspath(os.path.join(root_folder, glove_folder
    word2vec_output_file = glove_filename+'.word2vec'
    glove2word2vec(glove_path, word2vec_output_file)
    model = KeyedVectors.load_word2vec_format(word2vec_output_file, bin
    vectorizer = Word2VecVectorizer(model)
    Xtrain = vectorizer.fit_transform(X_train)
    Ytrain = y_train
    Xtest = vectorizer.transform(X_test)
    Ytest = y_test
    clf_cur = svm.SVC(kernel='linear',C=1,random_state=42)
    pred_cur = clf_cur.fit(Xtrain, Ytrain).predict(Xtest)
    accu_list_glove.append(accuracy_score(Ytest,pred_cur))
```

```
Training for:  glove.6B.50d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 4732
Number of samples with no words found: 0 / 3150
Training for:  glove.6B.100d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 4732
Number of samples with no words found: 0 / 3150
Training for:  glove.6B.200d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 4732
Number of samples with no words found: 0 / 3150
Training for:  glove.6B.300d.txt
Loading in word vectors...
Finished loading in word vectors
Number of samples with no words found: 0 / 4732
Number of samples with no words found: 0 / 3150
```

In [55]:
```
dim_list = [50,100,200,300]
plt.plot(dim_list,accu_list_glove)
plt.title('Accuracy vs. Dimension of GLoVE for Linear SVM (Gamma = 1)')
plt.xlabel('Dimension of pre-trained GLoVE embedding')
plt.ylabel('Test accuracy')
plt.savefig('Q101.png',dpi=300,bbox_inches='tight')
plt.show()
```

Question 11

In [56]:
```
reduced_dim_embedding = umap.UMAP(n_components=2, metric='euclidean').f
print(reduced_dim_embedding.embedding_.shape)
```

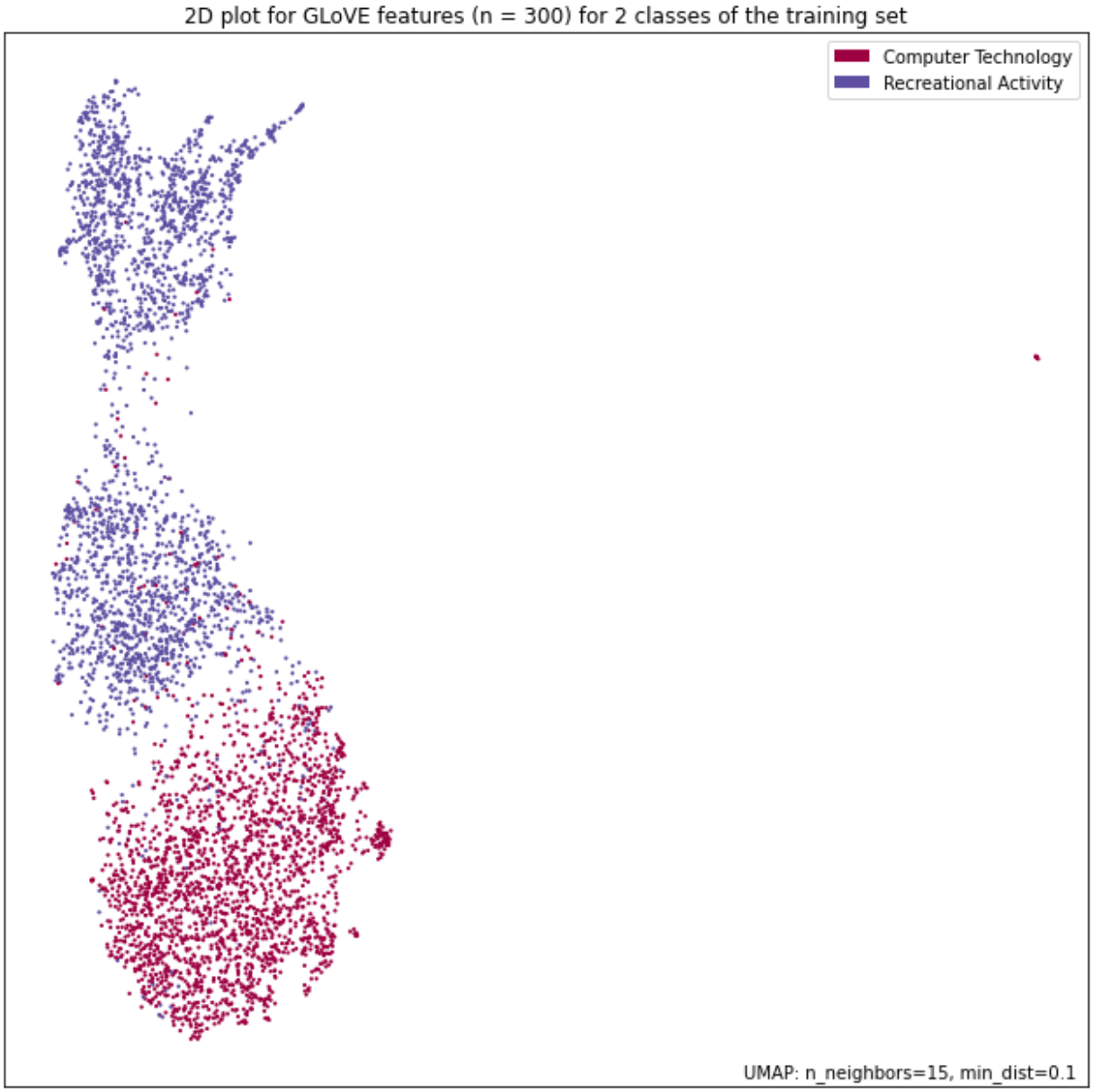
```
(4732, 2)
```

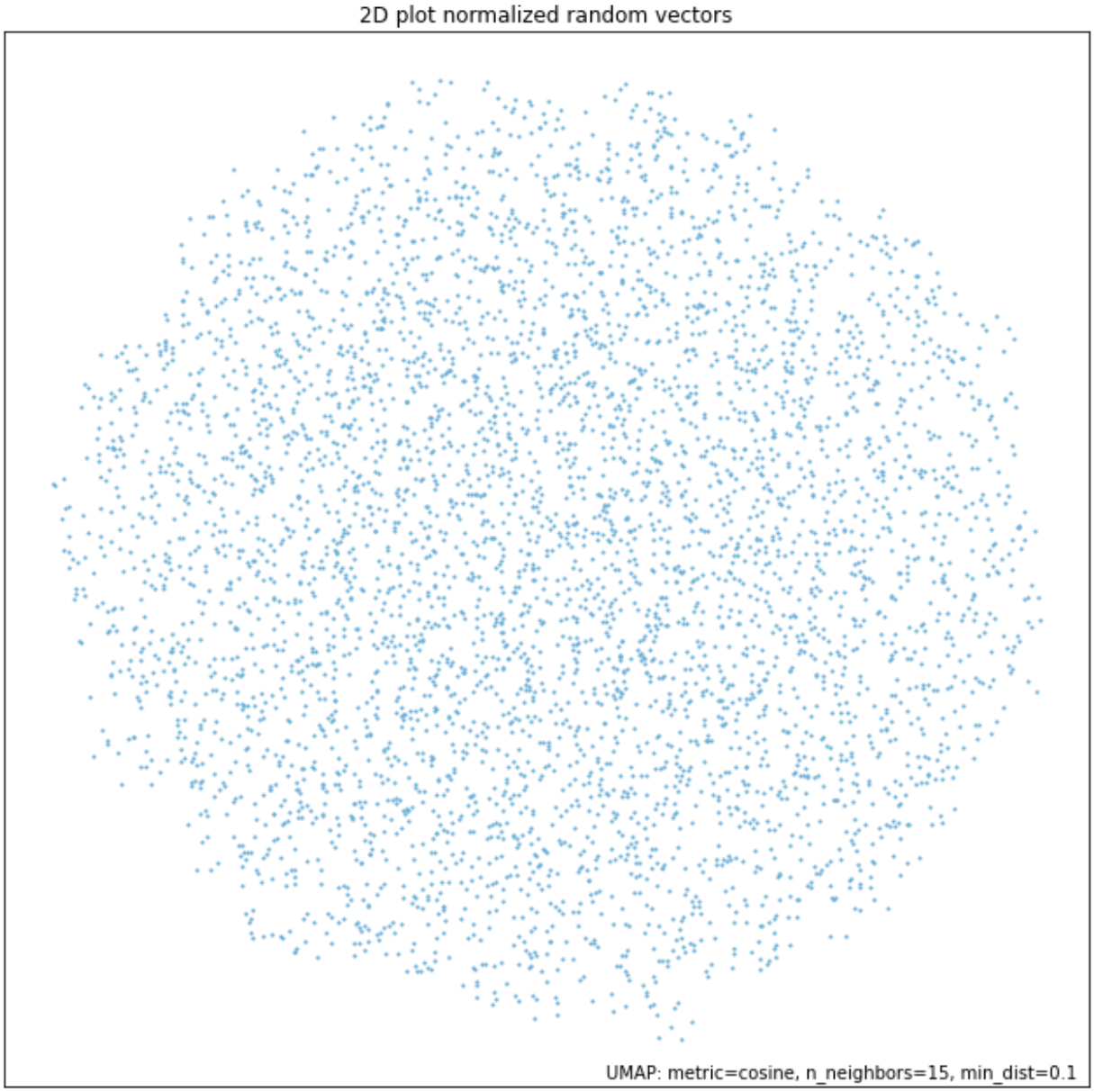
In [57]:
```
YtrainTextLabel = []
for label in Ytrain:
    if(label==0):
        YtrainTextLabel.append('Computer Technology')
    else:
        YtrainTextLabel.append('Recreational Activity')
```

In [58]:
```
s = np.random.normal(0, 1, [4732,300])
s = s / np.linalg.norm(s)
reduced_dim_s = umap.UMAP(n_components=2, metric='cosine').fit(s)
```

In [59]:
```
f = umap.plot.points(reduced_dim_embedding, labels=np.array(YtrainTextL
plt.title('2D plot for GLoVE features (n = 300) for 2 classes of the tr
plt.savefig('Q111.png',dpi=300,bbox_inches='tight')
g = umap.plot.points(reduced_dim_s)
plt.title('2D plot normalized random vectors')
plt.savefig('Q112.png',dpi=300,bbox_inches='tight')
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which shou
ld be avoided as value-mapping will have precedence in case its length ma
tches with *x* & *y*.  Please use the *color* keyword-argument or provide
a 2-D array with a single row if you intend to specify the same RGB or RG
BA value for all points.
```





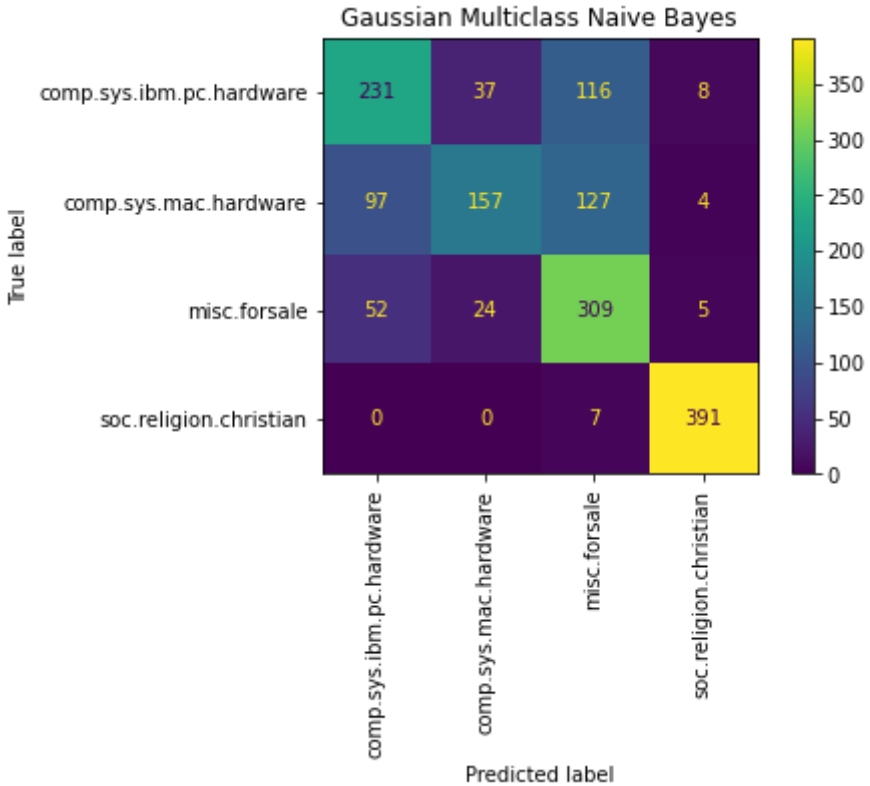
Question 12

In [73]:
```
categories_mc = ['comp.sys.ibm.pc.hardware','comp.sys.mac.hardware','mi
train_dataset_mc = fetch_20newsgroups(subset = 'train', categories = ca
test_dataset_mc = fetch_20newsgroups(subset = 'test', categories = cate
```

In [76]:
```
vectorizer_mc = CountVectorizer(stop_words='english',min_df=3)
tfidf_transformer_mc = TfidfTransformer()
train_data_proc_mc = []
test_data_proc_mc = []
for i in range(len(train_dataset_mc.data)):
    train_data_proc_mc.append(lemmatizer_func(train_dataset_mc.data[i]))
for i in range(len(test_dataset_mc.data)):
    test_data_proc_mc.append(lemmatizer_func(test_dataset_mc.data[i]))

train_data_feat_vec_mc = vectorizer_mc.fit_transform(train_data_proc_mc
test_data_feat_vec_mc = vectorizer_mc.transform(test_data_proc_mc)
train_data_feat_mc = tfidf_transformer_mc.fit_transform(train_data_feat
test_data_feat_mc = tfidf_transformer_mc.transform(test_data_feat_vec_m
svd_mc = TruncatedSVD(n_components=50, random_state=42)
train_data_LSI_mc = svd_mc.fit_transform(train_data_feat_mc)
test_data_LSI_mc = svd_mc.transform(test_data_feat_mc)
```

In [77]:
```
clf_NB_mc = GaussianNB()
pred_NB_mc = clf_NB_mc.fit(train_data_LSI_mc, train_dataset_mc.target).
```

In [78]:
```
plot_confusion_matrix(clf_NB_mc, test_data_LSI_mc, test_dataset_mc.targ
plt.xticks(rotation=90)
plt.title('Gaussian Multiclass Naive Bayes')
plt.savefig('Q121.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [79]:
```
print("Accuracy (Multiclass Gaussian Naive Bayes):", accuracy_score(tes
print("Recall (Multiclass Gaussian Naive Bayes):", recall_score(test_da
print("Precision (Multiclass Gaussian Naive Bayes):", precision_score(t
print("F1-Score (Multiclass Gaussian Naive Bayes):", f1_score(test_data
```

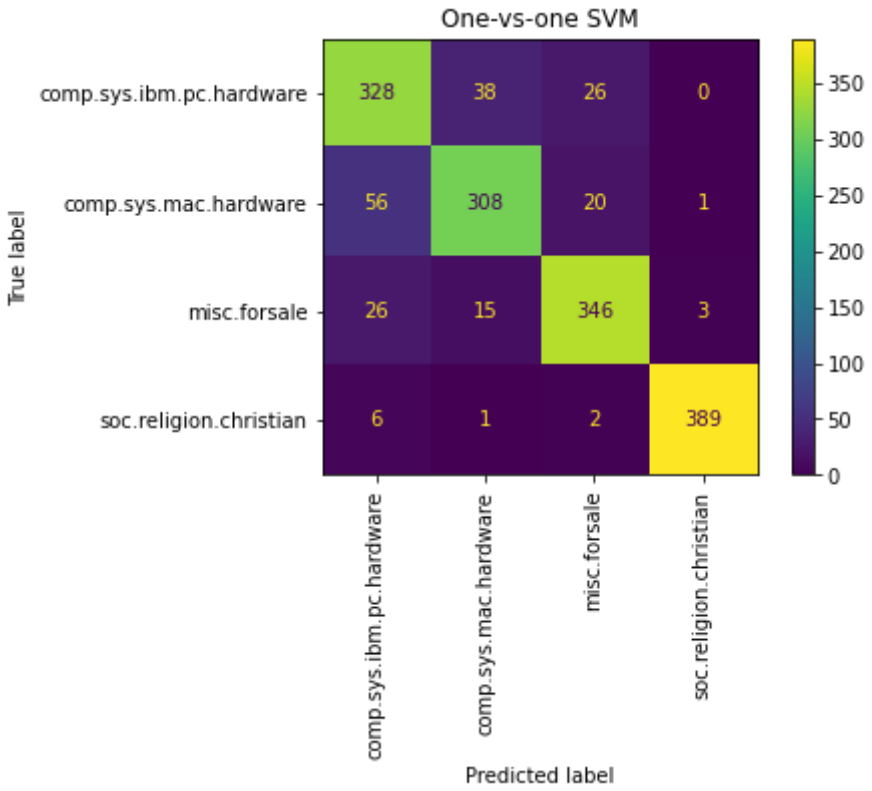
```
Accuracy (Multiclass Gaussian Naive Bayes): 0.6952076677316293
Recall (Multiclass Gaussian Naive Bayes): 0.6952076677316293
Precision (Multiclass Gaussian Naive Bayes): 0.7109031572264817
F1-Score (Multiclass Gaussian Naive Bayes): 0.6870243376226386
```

In [80]:
```
clf_svm_mc = OneVsOneClassifier(svm.SVC(random_state=42))
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,200,400,600,800,
              'estimator__kernel': ['linear']}
grid_svm_mc = GridSearchCV(clf_svm_mc,param_grid,cv=5,scoring='accuracy
grid_svm_mc.fit(train_data_LSI_mc, train_dataset_mc.target)
pred_svm_mc = grid_svm_mc.best_estimator_.predict(test_data_LSI_mc)
```

In [81]:
```
print(grid_svm_mc.best_estimator_)
```

```
OneVsOneClassifier(estimator=SVC(C=10, kernel='linear', random_state=42))
```

In [82]:
```
plot_confusion_matrix(grid_svm_mc.best_estimator_, test_data_LSI_mc, te
plt.xticks(rotation=90)
plt.title('One-vs-one SVM')
plt.savefig('Q122.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [83]:
```
print("Accuracy (One-vs-one SVM):", accuracy_score(test_dataset_mc.targ
print("Recall (One-vs-one SVM):", recall_score(test_dataset_mc.target,p
print("Precision (One-vs-one SVM):", precision_score(test_dataset_mc.ta
print("F1-Score (One-vs-one SVM):", f1_score(test_dataset_mc.target,pre
```

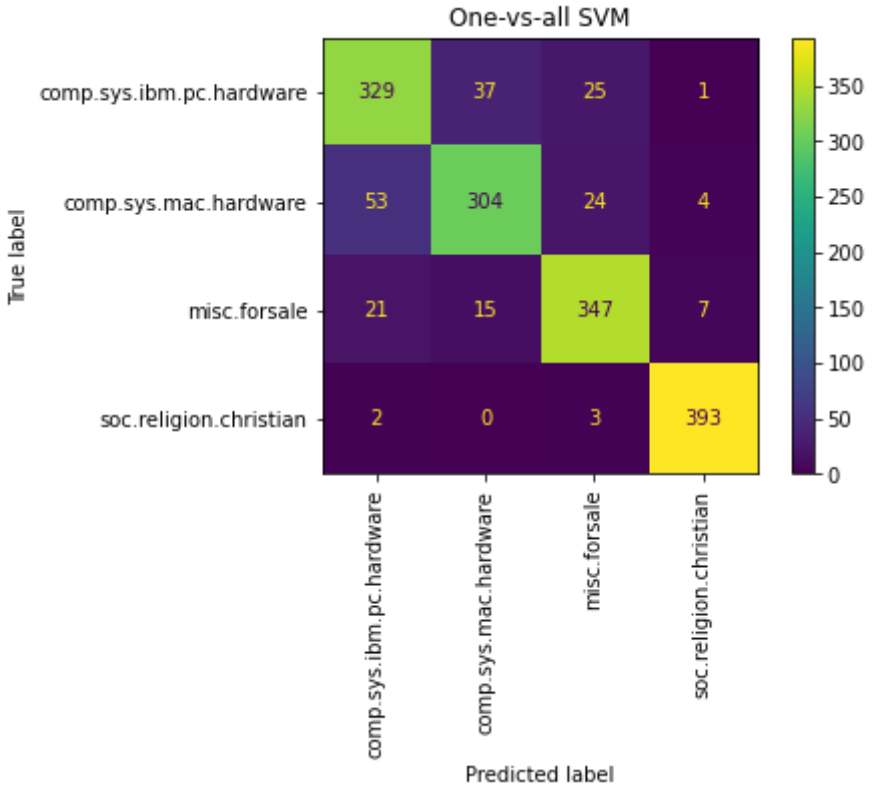
```
Accuracy (One-vs-one SVM): 0.876038338658147
Recall (One-vs-one SVM): 0.876038338658147
Precision (One-vs-one SVM): 0.8773689475462186
F1-Score (One-vs-one SVM): 0.8763158458707359
```

In [84]:
```
clf_svm_mc_oa = OneVsRestClassifier(svm.SVC(random_state=42))
param_grid = {'estimator__C': [0.001,0.01,0.1,1,10,100,200,400,600,800,
              'estimator__kernel': ['linear']}
grid_svm_mc_oa = GridSearchCV(clf_svm_mc_oa,param_grid,cv=5,scoring='ac
grid_svm_mc_oa.fit(train_data_LSI_mc, train_dataset_mc.target)
pred_svm_mc_oa = grid_svm_mc_oa.best_estimator_.predict(test_data_LSI_m
```

In [85]:
```
print(grid_svm_mc_oa.best_estimator_)
```

```
OneVsRestClassifier(estimator=SVC(C=1, kernel='linear', random_state=42))
```

In [86]:
```
plot_confusion_matrix(grid_svm_mc_oa.best_estimator_, test_data_LSI_mc,
plt.xticks(rotation=90)
plt.title('One-vs-all SVM')
plt.savefig('Q123.png',dpi=300,bbox_inches='tight')
plt.show()
```



In [87]:
```
print("Accuracy (One-vs-all SVM):", accuracy_score(test_dataset_mc.targ
print("Recall (One-vs-all SVM):", recall_score(test_dataset_mc.target,p
print("Precision (One-vs-all SVM):", precision_score(test_dataset_mc.ta
print("F1-Score (One-vs-all SVM):", f1_score(test_dataset_mc.target,pre
```

```
Accuracy (One-vs-all SVM): 0.8773162939297124
Recall (One-vs-all SVM): 0.8773162939297124
Precision (One-vs-all SVM): 0.8770504094919451
F1-Score (One-vs-all SVM): 0.876771235927221
```

In [ ]:
```

```