ECE219 Large-Scale Data Mining: Models and Algorithms **(Due: Fri, 01/24/20)**

Project 1: Classification Analysis on Textual Data

Instructor: Vwani Roychowdhury Zhaoxi Yu(005432230) & Yuhao Yin(104880239)

1 Getting Familiar with the Dataset

The dataset that we are working with in this project, is called "20 Newsgroups", a collection of approximately 20,000 documents partitioned into 20 different topics, including Computer Technology, Recreational Activity, Science, etc. Based on the textual content, we are expected to identify the document topics, which is frequently referred to as a statistical classification task.

In a classification problem, one should make sure to properly handle any imbalance in the relative sizes of the data sets corresponding to different classes. To do so, one can either modify the penalty function (i.e., assign more weight to errors from minority classes), or alternatively, down-sample the majority classes, to have the same number of instances as minority classes.

QUESTION 1. To get started, plot a histogram of the number of training documents for each of the 20 categories to check if they are evenly distributed.

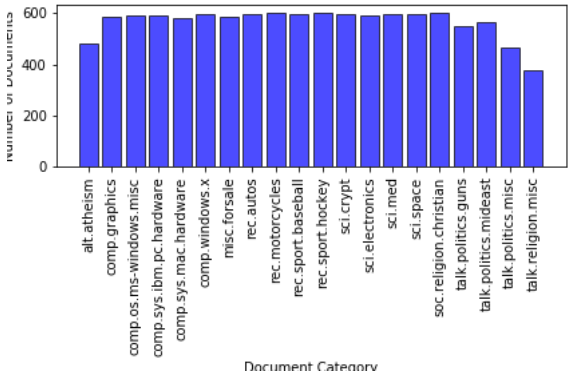


Figure 1: Histogram of Training Data

From the histogram, we can safely conclude that the "20 Newsgroups" dataset is already balanced (especially for the categories we'll mainly work on).

2 Binary Classification

We start with binary classification, i.e., the main task here would be to classify the documents into two classes "Computer Technology" vs "Recreational Activity". Since we originally have 8 different topic categories, we first convert them to binary target, 0 representing Computer Technology and 1 representing Recreational Activity, following the projection rule summarized in the following table.

Computer Technology	Recreational Activity
comp.graphics	rec.autos
comp.os.ms-windows.misc	rec.motorcycles
comp.sys.ibm.pc.hardware	rec.sport.baseball
comp.sys.mac.hardware	rec.sport.hockey

For example, the first 20 training data before and after this transformation look like,

$$Original\ Target: [6, 7, 4, 2, 1, 3, 0, 7, 5, 3, 0, 5, 5, 5, 3, 1, 3, 0, 0, 2]$$
$$Binary\ Target: [1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0]$$

2.1 Feature Extraction

The primary step in classifying a corpus of text is choosing a proper document representation. A good representation should retain enough information that enable us to perform the classification, yet in the meantime, be concise to avoid computational intractability and over-fitting.

In this project, we choose the "Bag of Words" representation, where a document is represented as a histogram of term frequencies, and normalize the count of vocabulary words using the "Term Frequency-Inverse Document Frequency(TF-IDF)" metric.

QUESTION 2. Use the following specs to extract features from the textual data:

- Use the `"english"` stopwords of the `CountVectorizer`
- Exclude terms that are numbers (e.g. "123", "-45", "6.7" etc.)
- Perform lemmatization with `nltk.wordnet.WordNetLemmatizer` and `pos_tag`
- Use `min_df=3`

Report the shape of the TF-IDF matrices of the train and test subsets respectively.

Following the instructions provided, we extract term frequency features from the raw textual data, drop terms that are very rare, by setting `min_df=3` in the `CountVectorizer`, perform lemmatization with `nltk.wordnet.WordNetLemmatizer` and `pos_tag`.

It's worth noting that we exclude not only **numbers**, but also **any term that is not alphabetic**, with the assumptions that non-letter terms do not provide actual information concerning the topic class.

The shape of the TF-IDF matrices of the train and test subsets are reported as follows,

Train	(4732, 12609)
Test	(3150, 12609)

Table 1: Shape of Document-term TF-IDF Matrices

2.2 Dimensionality Reduction

The dimensionality of TF-IDF vectors above ranges in the order of thousands. Since the document-term TF-IDF matrix is sparse and low-rank, we can transform the features into a lower dimensional space in order to circumvent the curse of dimensionality.

In this project, we use two dimensionality reduction methods: **Latent Semantic Indexing (LSI)** and **Non-negative Matrix Factorization (NMF)**, both of which minimize mean squared residual between the original data and a reconstruction from its low-dimensional approximation.

QUESTION 3. Reduce the dimensionality of the data using the method above

- Apply LSI to the TF-IDF matrix corresponding to the 8 categories with $k = 50$; so each document is mapped to a 50-dimensional vector.
- Also reduce dimensionality through NMF ($k = 50$) and compare with LSI: Which one is larger, the $\|X - WH\|_F^2$ in NMF or the $\|X - U_k \Sigma_k V_k^\top\|_F^2$ in LSI? Why is the case?

We employ functions `TruncatedSVD` and `NMF` both from `sklearn.decomposition`, to perform LSI and NMF respectively on the document-term IF-IDF matrices. After that, we further compute the squared frobenius norm of difference matrices between the original data and a reconstruction from their respective low-dimensional approximation. The results are reported as below,

Squared Frobenius Norm for LSI	4120.1546
Squared Frobenius Norm for NMF	4153.7582

Table 2: Squared Frobenius Norm for LSI and NMF

Remark. We conclude that squared frobenius norm for NMF is larger, since reconstruction from largest k eigenvalues as well as their corresponding left and right eigenvectors mathematically guarantees to have the smallest mean squared residuals compared to the original data.

2.3 Classification Algorithms

In this part, we use the dimension-reduced training data from LSI to train various types of classifiers, and evaluate the trained classifiers on test data with different classification measures.

2.3.1 SVM

Linear Support Vector Machines have been proved efficient when dealing with sparse high dimensional datasets, including textual data. They have been shown to have good generalization accuracy, while having low computational complexity.

The learning process of the parameter $\mathbf{w}$ and b involves solving the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \ & \frac{1}{2}\|\mathbf{w}\|_2^2 + \gamma \sum_{i=1}^{n} \xi_i \\ s.t. \ & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \quad \forall i \in \{1, \cdots, n\} \end{aligned}$$

where $\mathbf{x}_i$ is the ith data point, and $y_i \in \{0, 1\}$ is the class label of it. Note that the tradeoff parameter γ controls relative importance of minimizing the loss function on the training data versus maximizing the margin between the two classes.

QUESTION 4. Hard margin and soft margin linear SVMs:

- Train two linear SVMs and compare:
  - Train one SVM with $\gamma = 1000$ (hard margin), another with $\gamma = 0.0001$ (soft margin).
  - Plot the **ROC curve**, report the **confusion matrix** and calculate the **accuracy**, **recall**, **precision** and **F-1 score** of both SVM classifier. Which one performs better?
  - What happens for the soft margin SVM? Why is the case?
    - Does the ROC curve of the soft margin SVM look good? Does this conflict with other metrics?
- Use cross-validation to choose γ (use average validation accuracy to compare):
  Using a 5-fold cross-validation, find the best value of the parameter γ in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$. Again, plot the ROC curve and report the confusion matrix and calculate the accuracy, recall precision and F-1 score of this best SVM.

We employ `SVC` from `sklearn.svm` and set the parameter `kernel='linear'` in order to build up the linear SVM classifiers.

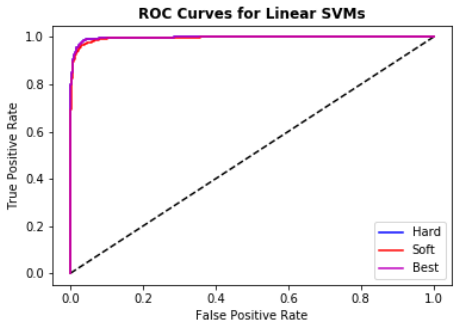
By setting the parameter `C=1000` or `C=0.0001`, we end up with hard and soft margin linear SVMs.

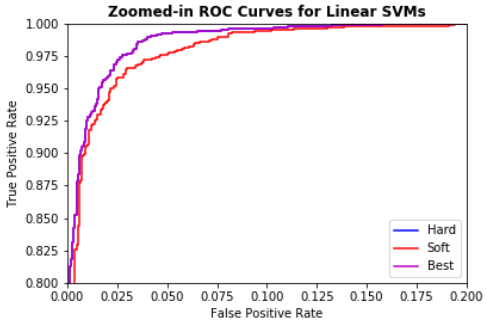
Furthermore, to choose the best tradeoff parameter γ, we then further perform a 5-fold cross-validation to find the best value of γ in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$ using `GridSearchCV`. The average validation accuracy is summarized in the following table,

	param_C	mean_test_score
0	0.001	0.504861
1	0.01	0.507396
2	0.1	0.964074
3	1	0.972104
4	10	0.975063
5	100	0.976330
6	1000	0.977597

which indicates that the optimal tradeoff parameter through cross-validation is `C=1000`.

Report ROC curves and its zoom-in version, attempting to amplify the differences:





As well as confusion matrices:

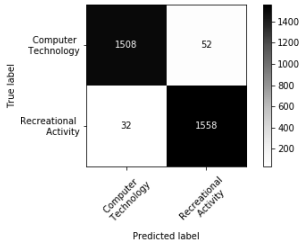


Figure 2: Hard Margin(C=1000)

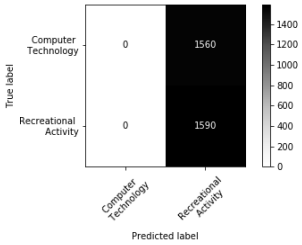


Figure 3: Soft Margin(C=0.0001)

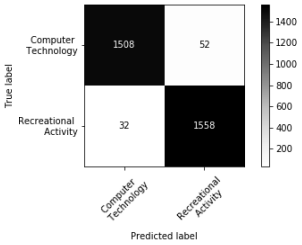


Figure 4: Best(C=1000)

Together with classification measures:

	Accuracy	Recall	Precision	F-1 Score
Hard Margin	0.9733	0.9799	0.9677	0.9738
Soft Margin	0.5048	1.0	0.5048	0.6709
Best	0.9733	0.9799	0.9677	0.9738

Table 3: Classification Measures for Linear SVMs

Remark. Hard margin SVM performs better compared to soft margin SVM among all performance metrics. For the soft margin SVM, the classifier simply predicts all documents in testing data as "Recreational Activity", since when γ is pretty small, the first term $\frac{1}{2}\|\mathbf{w}\|_2^2$ in the objective function dominates. Therefore, the soft margin SVM is very lenient towards misclassification of a few individual points as long as most data points are well separated. That's why for the soft margin SVM, most of the weight variables $\mathbf{w}$ are close to zero vector. However, the ROC curve of the soft margin SVM looks reasonable, since its true positive rate (TPR) or Recall is surprisingly and consistently high (almost 1) when the threshold varies.

2.3.2 Logistic Regression

In logistic regression, a logistic function ($\sigma(\phi) = 1/(1+\exp(-\phi))$) acting on a linear function of the features ($\phi(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$) is usead to calculate the probability that the data point belongs to class 1, and during the training process, $\mathbf{w}$ and b that maximizes the likelihood of the training data are learnt.

One can also add regularization term in the objective function, so that the goal of the training process is not only maximizing the likelihood, but also minimizing the regularization term, which is often some norm of the parameter vector $\mathbf{w}$. Adding regularization helps prevent ill-conditioned results and overfitting, and facilitate generalization ability of the classifier.

QUESTION 5. Logistic classifier:

- Train a logistic classifier without regularization (you may need to come up with some way to approximate this if you use `sklearn.linear_model.LogisticRegression`); plot the ROC curve and report the confusion matrix and calculate the **accuracy, recall precision** and **F-1 score** of this classifier.
- Regularization:
  - Using 5-fold cross-validation on the dimension-reduced-by-svd training data, find the best regularization strength in the range $\{10^k | -3 \leq k \leq 3, k \in \mathbb{Z}\}$ for logistic regression with L1 regularization and logistic regression L2 regularization, respectively.
  - Compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classifiers: w/o regularization, w/L1 regularization and w/L2 regularization (with the best parameters you found from the part above), using test data.
  - How does the regularization parameter affect the test error? How are the learned coefficients affected? Why might one be interested in each type of regularization?
  - Both logistic regression and linear SVM are trying to classify data points using a linear decision boundary, then what's the difference between their ways to find this boundary? Why their performance differ?

We use `sklearn.linear_model.LogisticRegression` to build up logistic classifiers. Specifically, by setting the the inverse of regularization strength `C` to be infinite, e.g. `C=10**15`, we end up with a logistic classifier without regularization, since in this situation, maximizing the log-likelihood term will dominate the whole objective function.

In addition, we apply 5-fold cross-validation on the dimension-reduced training data to find the best regularization strength for logistic classifiers with both L1 and L2 regularization. The average validation accuracy for different values of regularization strength is summarized as follows,

	param_C	mean_test_score
0	0.001	0.495139
1	0.01	0.936601
2	0.1	0.951184
3	1	0.970414
4	10	0.975273
5	100	0.978020
6	1000	0.978231

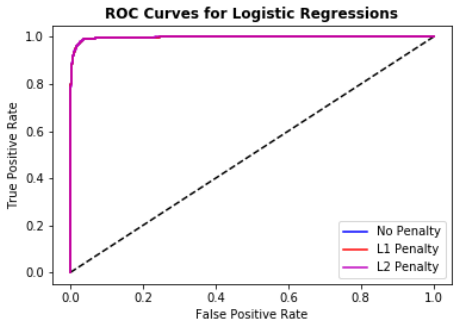
Figure 5: L1 Regularization

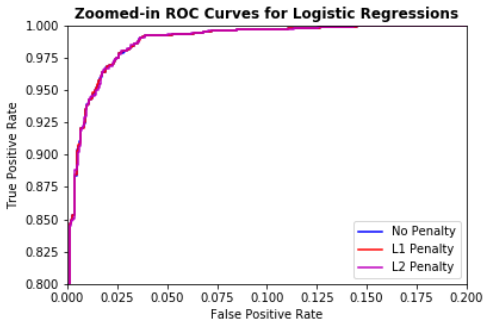
	param_C	mean_test_score
0	0.001	0.542476
1	0.01	0.938293
2	0.1	0.963440
3	1	0.969357
4	10	0.974641
5	100	0.977598
6	1000	0.977809

Figure 6: L2 Regularization

Hence, the optimal inverse of regularization strength for both L1 and L2 regularization is `C=1000`.

Report the ROC curves for 3 logistic classifiers as well as the detailed version:





Report the confusion matrices:

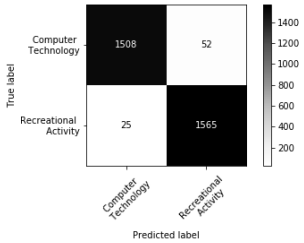


Figure 7: No Penalty

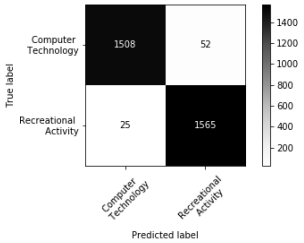


Figure 8: Optimized L1 Penalty

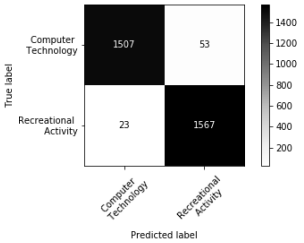


Figure 9: Optimized L2 Penalty

Last but not least, also report various classification measures:

	Accuracy	**Recall**	**Precision**	**F-1 Score**
No Penalty	0.9756	0.9843	0.9678	0.9760
Optimized L1 Penalty	0.9756	0.9843	0.9678	0.9760
Optimized L2 Penalty	0.9759	0.9855	0.9673	0.9763

Table 4: Classification Measures for Logistic Regressions

Remark. Adding both L1 and L2 regularization decreases the testing accuracy, causing logistic classifiers underfit this textual dataset. In terms of the learned coefficients, weights in logistic regression L2 regularization have both smaller mean absolute value and standard deviation compared to that in logistic regression L1 regularization. The reason is straightforward: L2 regularization punishes large weights more by squaring them, and hence will always enforce the learned coefficients shrink to zeros.

Remark 2. Because of that, one might prefer L2 regularization when he/she wants to implement weight shrinkage so as to prevent the model overfitting problem. L1 regularization functions similarly, but it's more robust against outliers in the training stage. Moreover, by properly setting the regularization strength, one may end up with some weights being exactly zero, which facilitates feature selection due to the weight sparsity.

Remark 3. Linear SVM finds the linear decision boundary by means of maximizing the margin subject to a total distance of points on the wrong side of their margin being no more than some constant. While logistic regression directly models the posterior probability of document topic given its textual content through logit link function on top of the linear transformation. Parameters are learned by optimizing the likelihood, i.e. probability of observing training data.

2.3.3 Naive Bayes

Naive Bayes classifiers use the assumption that features are statistically independent of each other when conditioned by the class the data point belongs to, to simplify the calculation for the Maximum A Posteriori (MAP) estimation of the labels. That is,

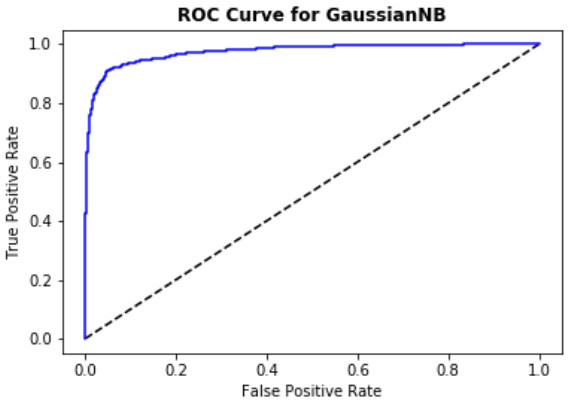
$$\mathbb{P}(x_i|y, x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_m) = \mathbb{P}(x_i|y), \quad \forall i = 1, \cdots, m.$$

where $x_i's$ are features, i.e. components of a data point, and y is the label of the data point.

QUESTION 6. Naive Bayes classifier:

- Train a `GaussianNB` classifier;
- Plot the ROC curve and report the confusion matrix and calculate the **accuracy, recall, precision** and **F-1 score** of this classifier.

Similarly, we use `GaussianNB` to build a Gaussian Naive Bayes classifier with all default options. Report the ROC curve:



Confusion matrix:

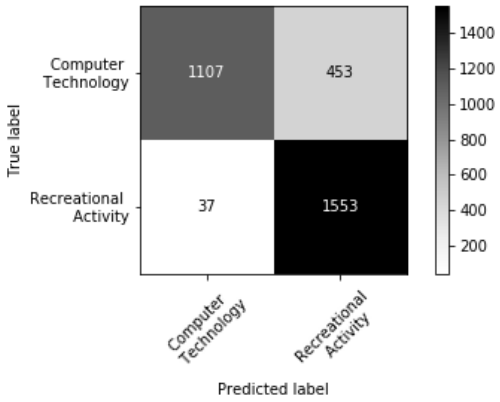


Figure 10: Confusion Matrix for GaussianNB

Together with all kinds of classification performance measures:

	Accuracy	**Recall**	**Precision**	**F-1 Score**
GaussianNB	0.8444	0.9767	0.7742	0.8637

Table 5: Classification Measures for Naive Bayes

Remark. In terms of all classification measures, Naive Bayes classifier performs worse compared to both linear SVM and logistic regression. One major drawback of this model is the independence assumption, since for this particular textual dataset, some vocabularies tend to appear together more often than others, leading to dependency among features.

2.4 Grid Search of Parameters

Now we have gone through the complete process of training and testing a classifier. However, there are lots of parameters that we can tune. In this part, we fine-tune the parameters.

QUESTION 7. Grid search of parameters:

- Construct a `Pipeline` that performs feature extraction, dimensionality reduction and classification;
- Do grid search with 5-fold cross-validation to compare the following (use test accuracy as the score to compare):

Table 2: Options to compare

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 with the best γ previously found vs Logistic Regression: L1 regularization vs L2 regularization, with the best regularization strength previously found vs `GaussianNB`
Other options	Use default

- What is the best combination?

To accomplish the objective of question 7, we construct a pipeline and perform 5-fold cross-validation to grid-search for the optimal parameter settings in terms of highest mean testing accuracy score. The best 5 parameter combinations are reported in the following table:

mean_test_score	classifier	reduce_dim	min_df	lemmatized	with_headfoot
0.977388	L2 Logistic(C=1000)	LSI	5	TRUE	TRUE
0.977176	L2 Logistic(C=1000)	LSI	3	TRUE	TRUE
0.976965	L1 Logistic(C=1000)	LSI	5	TRUE	TRUE
0.976331	SVC(C=1000)	LSI	5	TRUE	TRUE
0.976120	SVC(C=1000)	LSI	3	TRUE	TRUE

Table 6: Best Five Parameter Combinations

Remark. Therefore, the best combination is loading the textual data with "headers" and "footers", using lemmatization and setting the parameter `min_df=5` for `CountVectorizer`, reducing the dimensionality through SVD and selecting logistic regression with L2 regularization as the classifier.

Remark 2. Based on the aforementioned grid search result, we better understand that in order to increase the classification accuracy, one should prefer LSI compared to NMF in dimensionality reduction. Moreover, larger min_df, lemmatization and including headers and footers always tend to increase the model performance.

3 Multiclass Classification

So far, we have been dealing with classifying the data points into two classes. In this part, we explore multiclass classification techniques through different algorithms.

Some classifiers perform the multiclass classification inherently. As such, Naive Bayes algorithm finds the class with maximum likelihood given the data, regardless of the number of classes. In fact, the probability of each class label is computed in the usual way, then the class with the highest probability is picked; that is

$$\hat{c} = \arg\min_{c \in \mathcal{C}} \mathbb{P}(c|\mathbf{x})$$

For SVM, however, one needs to extend the binary classification techniques when there are multiple classes by adopting either an one-vs-one or one-vs-rest approach.

For the one-vs-one classification, we give a document the class that is assigned with the majority vote. In case there is more than one class with the highest vote, the class with the highest total classification confidence levels in the binary classifiers is picked.

For the one-vs-rest approach, we fit one classifier per class against all the other classes. Note that in this case, the unbalanced number of documents in each class should be handled. By learning a single classifier for each class, one can get insights on the interpretation of the classes based on the features.

QUESTION 8. In this part, we aim to learn classifiers on the documents belonging to the classes: `comp.sys.ibm.pc.hardware, comp.sys.mac.hardware`, `misc.forsale, soc.religion.christian`.

Perform Naive Bayes classification and multiclass SVM classification (with both One VS One and One VS Rest methods described above) and report the **confusion matrix** and calculate the **accuracy, recall, precision** and **F-1 score** of your classifiers.

We use `GaussianNB` to build up multiclass Naive Bayes classifier, and `OneVsOneClassifier(SVC())`, `OneVsRestClassifier(SVC())` to construct One-Vs-One and One-Vs-Rest multiclass SVM classification respectively.

Furthermore, we explore classification performances across the original document-term TF-IDF metrics, dimension-reduced LSI features, as well as dimension-reduced NMF features. Respective confusion matrices are summarized below,

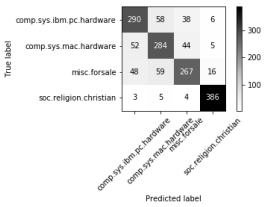
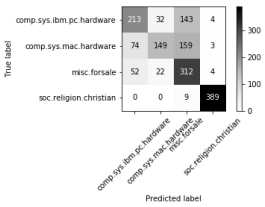
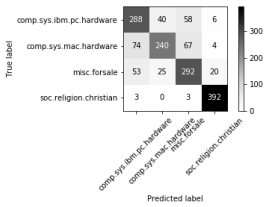
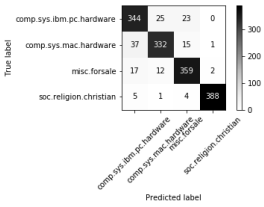
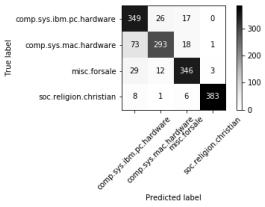
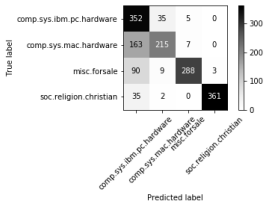
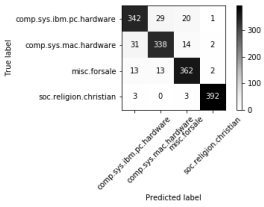
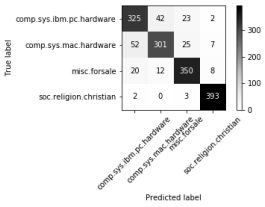
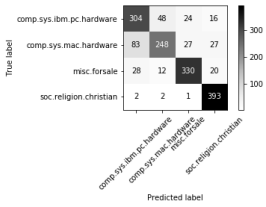
	TF-IDF	**LSI**	**NMF**
Naive Bayes			
One-Vs-One SVM			
One-Vs-Rest SVM			

Table 7: Confusion Matrix for Multiclass Classification

Moreover, we further calculate all classification measures. Since we are dealing with multiclass classification tasks, recall, precision and F-1 score will be reported in the form of **macro-averages**.

		Accuracy	Recall	Precision	F-1 Score
Naive Bayes	**TF-IDF**	0.7840	0.7830	0.7821	0.7819
	LSI	0.6792	0.6769	0.7089	0.6701
	NMF	0.7744	0.7729	0.7750	0.7710
O-v-O SVM	**TF-IDF**	0.9093	0.9088	0.9096	0.9090
	LSI	0.8760	0.8752	0.8816	0.8760
	NMF	0.7770	0.7755	0.8314	0.7825
O-v-R SVM	**TF-IDF**	0.9163	0.9159	0.9158	0.9158
	LSI	0.8748	0.8739	0.8734	0.8732
	NMF	0.8147	0.8133	0.8137	0.8101

Table 8: Classification Measures for Multiclass Classification

Remark. For multiclass classification task, both One-Vs-One and One-Vs-Rest SVMs perform better compared to Naive Bayes classifier, mainly due to the inappropriate assumption of the Naive Bayes model.
Remark 2. In terms of feature engineering, the original document-term TF-IDF metrics has the highest testing accuracy, recall, precision and F-1 score across all implemented models, indicating multiclass classification tasks won't suffer too much from input matrix sparsity, as well as curse of dimensionality.