ECE 219: Large-Scale Data Mining: Models and Algorithms
Project 1: Classification Analysis on Textual Data

Prof. Vwani Roychowdhury
UCLA, Department of ECE

ECE 219 Winter 2020

Siyuan Peng 805426447 Haonan Huang 605322629
Yinghan Cui 005430212 Yunzheng Zhu 505231998

1 Getting familiar with the dataset

We work with "20 Newsgroups" dataset, which is a collection of approximately 20,000 documents, partitioned (nearly) evenly across 20 different newsgroups (newsgroups are discussion groups like forums, which originated during the early age of the Internet), each corresponding to a different topic.

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.

We first fetch all training document categories and plot the histogram of them.

While 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. We can see from the histogram that all 20 categories are roughly balanced. What is more, the eight categories we use is already balanced.

To get started, we work with a well separable portion of data, and see if we can train a classifier that distinguishes two classes well. Concretely, we take all the documents in the following classes:

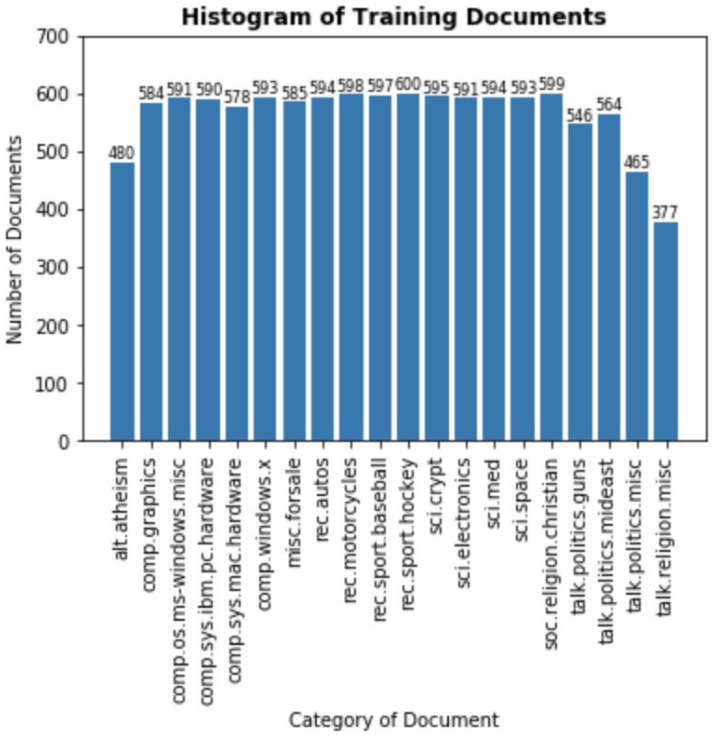

Figure 1: Histogram of training document.

Table 1: Two well-separated classes

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

2 Binary Classification

2.1 Feature Extraction

We use "Bag of Words", where a document is represented as a histogram of term frequencies, or other statistics of the terms, within a fixed vocabulary. As such, a corpus of text can be summarized into a term-document matrix whose entries are some statistic of the terms.

A popular numerical statistic to capture the importance of a word to a document in a corpus is the "Term Frequency-Inverse Document Frequency (TF-IDF)" metric. It is defined as $tf-idf(d,t) = tf(t,d) \times idf(t)$, where $tf(d,t)$ represents the frequency of term t in document d, and inverse document frequency is defined as: $idf(t) = log(\frac{n}{df(t)}) + 1$, where n is the total number of documents, and $df(t)$ is the document frequency, i.e. the number of documents that contain the term t.

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

---

In [7]:
```
import nltk
from nltk import pos_tag

wnl = nltk.wordnet.WordNetLemmatizer()

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

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

In [8]:
```
from sklearn.feature_extraction.text import CountVectorizer

analyzer = CountVectorizer().build_analyzer()

def ana_notnum(doc):
    return (word for word in lemmatize_sent(analyzer(doc)) if not word.isdigit())
```

In [9]:
```
vectorizer = CountVectorizer(min_df=3,
                             analyzer=ana_notnum,
                             stop_words='english')

# fit_tranform train data
X_train_counts = vectorizer.fit_transform(train_dataset.data)
print("train count size ", X_train_counts.shape)

X_test_counts = vectorizer.transform(test_dataset.data)
print("test count size ", X_test_counts.shape)
```

```
train count size  (4732, 16600)
test count size  (3150, 16600)
```

Figure 2: ROC of hard margin

As is shown in Fig 2, we use such way to achieve lemmatization and CountVectorizer while eliminating pure numbers. And we further use TfidfTransformer function from sklearn.feature_extraction.text package to transform CountVectorizer into ”Term Frequency-Inverse Document Frequency (TF-IDF)” matrix.

We constructed an analyzer by lowering words and lemmatizing based on part of speech tags, with pure numbers excluded. Stop words are chosen as ”english” and minimum document frequency

chosen as 3. We run the CountVectorizer first to get total counts and then apply the TF-IDF transformation. Note that the number of items in test set is the same as train set, which indicates a projection from test set to document-item space.

2.2 Dimensionality Reduction

2.2.1 LSI

The LSI representation is obtained by computing left and right singular vectors corresponding to the top k largest singular values of the term-document TF-IDF matrix X. We perform SVD to the matrix X, resulting in $X = U\Sigma V^T$, U and V orthogonal. Let the singular values in Σ be sorted in descending order, then the first k columns of U and V are called U_k and V_k respectively. V_k consists of the principle components in the feature space. Then we use XV_k (which is also equal to ($U_k\Sigma_k$) as the dimension-reduced data matrix, where rows still correspond to documents, only that they can have (far) lower dimension. In this way, the number of features is reduced. LSI is similar to Principal Component Analysis (PCA), and you can see the lecture notes for their relationships. Having learnt U and V, to reduce the test data, we just multiply the test TF-IDF matrix X_t by V_k , i.e. $X_{t,reduced} = X_t V_k$. By doing so, we actually project the test TF-IDF vectors to the principle components, and use the projections as the dimension-reduced data.

2.2.2 NMF

NMF tries to approximate the data matrix $X \in R^{n\times m}$ (i.e. we have n docs and m terms) with WH ($W \in R^{n\times r}$, $H \in R^{r\times m}$). Concretely, it finds the non-negative matrices W and H s.t. $\|X - WH\|_F^2$ is minimized. ($\|A\|_F \equiv \sqrt{\sum_{i.j} A_{ij}^2}$) Then we use W as the dim-reduced data matrix, and in the fit step, we calculate both W and H. The intuition behind this is that we are trying to describe the documents (the rows in X) as a (non-negative) linear combination of r topics:

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix} \approx WH = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1r} \\ \vdots & \vdots & \vdots & \vdots \\ w_{n1} & w_{n2} & \cdots & w_{nr} \end{bmatrix} \begin{bmatrix} h_1^T \\ \vdots \\ h_r^T \end{bmatrix} \tag{1}$$

And we calculate the dim-reduced test data matrix by solving this optimization problem

$$\min_{W_t \geq 0} \|X_t - W_t H\|_F^2 \tag{2}$$

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 $\|X - WH\|_F^2$ or the $\left\|X - U_k\Sigma_k V_k^T\right\|_F^2$ in LSI? Why is the case?

By importing functions of Truncated SVD and NMF in sklearn.decomposition, we reduced the dimension of TF-IDF matrix of train dataset from size 4732×16600 to size 4732×50. To measure the error from dimensionality reduction, we recovered the reduced matrix into the original document-item space through an inverse fitting function, and then computed the Frobenius norm between the recovered matrices and the original matrix corresponding to different methods as shown in table 1. The "error" from LSI is smaller than it from NMF, which meets our expectation since that in NMF, the matrices W and H have one strict requirement that each component inside matrices should be non-negative. On the other hand, the LSI kept the larger singular values but truncated the smaller ones, which means LSI actually preserved the significant features of the matrix.

The Squared Frobenius Norm for LSI: 3895.418679693327
The Squared Frobenius Norm for NMF: 3940.3425139328747

3 Classification Algorithms

3.1 Classification measures

Classification quality can be evaluated using different measures such as precision, recall, F-score, etc. Refer to the discussion material to find their definition.

Depending on application, the true positive rate (TPR) and the false positive rate (FPR) have different levels of significance. In order to characterize the trade-off between the two quantities, we plot the receiver operating characteristic (ROC) curve. For binary classification, the curve is created by plotting the true positive rate against the false positive rate at various threshold settings on the probabilities assigned to each class (let us assume probability p for class 0 and $1-$ p for class 1). In particular, a threshold t is applied to value of p to select between the two classes. The value of threshold t is swept from 0 to 1, and a pair of TPR and FPR is got for each value of t. The ROC is the curve of TPR plotted against FPR.

3.2 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. Linear Support Vector Machines aim to learn a vector of feature weights, w, and an intercept, b, given the training dataset. Once the weights are learned, the label of a data point is determined by thresholding $w^T x + b$ with 0, i.e. sign($w_T x + b$). Alternatively, one produce probabilities that the data point belongs to either class, by applying a logistic function instead of hard thresholding, i.e. calculating $\sigma(w_T x + b)$. The learning process of

the parameter w and b involves solving the following optimization problem:

$$\begin{aligned} &\min_{w,b} \ \frac{1}{2}\|w\|_2^2 + \gamma \sum_{i=1}^{n} \xi_i \\ &s.t.\ y_i(w^T x_i + b) \geq 1 - \xi_i \\ &\quad \xi_i \geq 0 \\ &\quad \forall i \in \{1, ..., n\} \end{aligned} \tag{3}$$

where x_i is the ith data point, and $y_i \in \{0, 1\}$ is the class label of it. Minimizing the sum of the slack variables corresponds to minimizing the loss function on the training data. On the other hand, minimizing the first term, which is basically a regularization, corresponds to maximizing the margin between the two classes. Note that in the objective function, each slack variable represents the amount of error that the classifier can tolerate for a given data sample. The tradeoff parameter γ controls relative importance of the two components of the objective function. For instance, when $\gamma \gg 1$, misclassification of individual points is highly penalized, which is called "Hard Margin SVM". In contrast, a "Soft Margin SVM", which is the case when $\gamma \ll 1$, is very lenient towards misclassification of a few individual points as long as most data points are well separated.

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 $\{\ 10^k | 3 \leq k \leq 3, k \in Z\ \}$. 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 use SVC function in sklearn.svm package to realize 2-category classification with SVC. And we use ROC, confusion matrix, accuracy, precision, recall and F-1 score to see our result. In detail, we use roc_curve function in sklearn.metrics package to draw ROC, and auc function in the same package to calculate auc. Furthermore, when calculating confusion matrix, accuracy, precision, recall and F-1 score, we use confusion_matrix, accuracy_score, recall_score, precision_score, and f1_score functions to do them, and we get good results.

AUC of hard margin is 0.995, while AUC of soft margin is 0.975, as shown in Fig 4. Other results can also be found in our jupyter notebook, all of which show that hard margin linear SVM is better than soft margin linear SVM.

In finding a best γ, we divide 10^{-3} and 10^3 into 12 areas. By comparing their validation accuracy

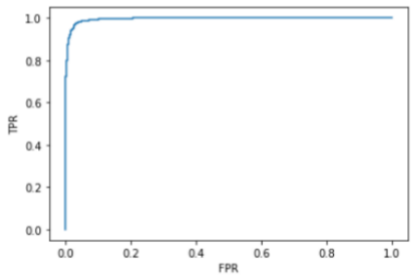

Figure 3: ROC of hard margin

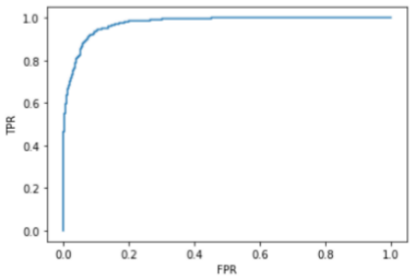

Figure 4: ROC of soft margin

separately, we conclude that the best γ is 33.0, which has the highest validation accuracy whose mean is 0.9759. Results of this γ can be found in our jupyter notebook code.

3.3 Logistic Regression

Although its name contains "regression", logistic regression is a probability model that is used for binary classification. In logistic regression, a logistic function ($\sigma(\phi) = 1/(1+e^{\phi})$) acting on a linear function of the features ($\phi(x) = w^T x + b$) is used to calculate the probability that the data point belongs to class 1, and during the training process, 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 w. Adding regularization helps prevent ill-conditioned results and over-fitting, and facilitate generalization ability of the classifier. A coefficient is used to control the trade-off between maximizing likelihood and minimizing the regularization term.

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 $\{ 10^k | 3 \leq k \leq 3, k \in Z \}$ 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 learnt 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 deci-

sion boundary, then what's the difference between their ways to find this boundary? Why their performance differ?

We use LogisticRegression function in sklearn.linear_model package to realize 2-category classification with Logistic Regression. We take ROC, confusion matrix, accuracy, precision, recall and F-1 score as indices to estimate the model.

First, we calculate the indices of the case without regularization. In this case, accuracy is 0.970, which is okay because we carefully select parameters of LogisticRegression function.

And then, we do L1 regularization and L2 regularization separately. And select regularization strength in the range $\{ 10^k | 3 \leq k \leq 3, k \in Z \}$ based on accuracy. The best γ for L1 regularization is 10.0 while the best γ for L2 regularization is 100.0. And we test two regression model with all the indices and compare the three. The result is shown in Chart 2.

Table 2: Indices of three cases of Logistic Regression

	accuracy	recall	precision	F-1 score
No regularization	0.970	0.981	0.962	0.971
L1 regularization	0.970	0.982	0.959	0.971
L2 regularization	0.970	0.982	0.960	0.971

3.4 Naive Bayes classifier

Scikit-learn provides a type of classifiers called "Naive Bayes classifiers". They include MultinomialNB, BernoulliNB, and GaussianNB. 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,

$$P(x_i|y, x_1, ..., x_{i1}, x_{i+1}, ..., x_m) = P(x_i|y), i \in 1, ..., m \tag{4}$$

wherex_i's are features, i.e. components of a data point, and y is the label of the data point. Now that we have this assumption, a probabilistic model is still needed; the difference between MultinomialNB, BernoulliNB, and GaussianNB is that they use different models.

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.

We use GaussianNB function in sklearn.naive_bayes package to realize Naive Bayes classifier. Again, we use ROC, confusion matrix, accuracy, precision, recall and F-1 score to see our result. And results can be seen in out jupyter notebook code.

3.5 Grid search of 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)

In this part, out task is to tune the parameters of the algorithms and compare the outcome of the program. We construct a pipeline implement 5-fold cross-validation to compare the options we are given, and find out which is the best combination. The options are shown below.

Table 3: Options of 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

Since the outcome can be very complicated and the data size is huge, so we check the head and tail

of the outcome form, to find out the best solution of parameter combination.

Table 4: Head of the outcome.

	with_headers_footers	min_df	lemmatized	Reduce dim	classification_method	mean_test_score
0	True	3	True	LSI	L2 Logistic	0.975063
1	True	3	True	LSI	L2 Logistic	0.975063
2	True	5	True	LSI	L1 Logistic	0.974852
3	True	5	True	LSI	L2 Logistic	0.974429
4	True	3	True	LSI	SVC with Parameter 33	0.974219

Table 5: Tail of the outcome.

	with_headers_footers	min_df	lemmatized	Reduce dim	classification_method	mean_test_score
59	True	5	True	LSI	Gaossian NB	0.892853
60	False	5	True	LSI	Gaossian NB	0.797129
61	False	5	True	LSI	Gaossian NB	0.787193
62	False	5	False	LSI	Gaossian NB	0.772400
63	False	3	False	LSI	Gaossian NB	0.761192

Note: According to the outcome data, we can see the best combination of parameters is with headers and footers, using lemmatization, mindf=3, the reducing dimensional method is LSI and using L2 Logistic. The mean test score is 0.975063, which is very high in our opinion.

3.6 Multiclass Classification

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

In this part, our task is to implement Naive Bayes classification and multiclass SVM classification (with both One VS One and One VS the rest methods) for classifying multiple document classes. Respective measurements and confusion matrices are shown below.

Table 6: Measurements of Naive Bayes, SVM one VS one, SVM one VS rest.

	Accuracy	Recall	Precision	F-1 Score
Naïve Bayes	0.7412	0.7389	0.7407	0.7260
SVM one VS one	0.8715	0.8708	0.8743	0.8715
SVM one VS rest	0.8722	0.8715	0.8710	0.8712

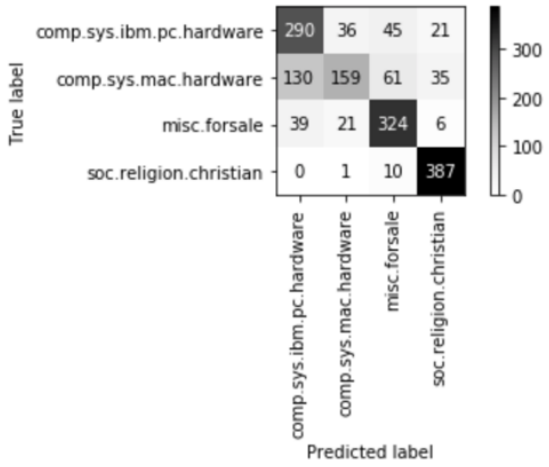

Figure 5: Confusion Matrix of Naive Bayes.

Note: For the data we acquired from the program, SVM (both one VS one and one VS rest) performs better then Naive Bayes. Since Naive Bayes is based on independent assumption, and the dataset may have items that are not independent to each other. On the other hand, the assumption of Gaussian distribution may also cause inaccuracy.

Although the difference between SVM one VS one and one VS rest is not so big, it is possible that with the number of classes increasing, the difference may be more obvious.

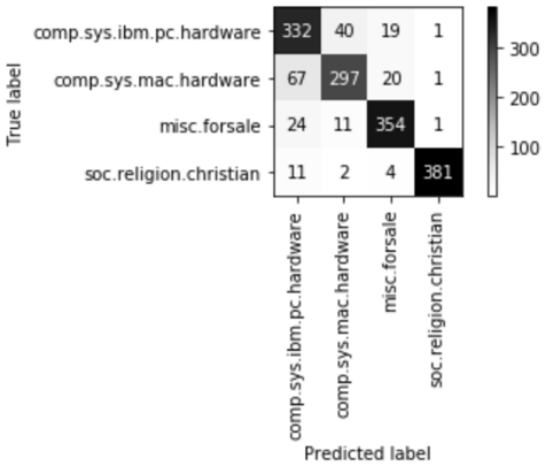

Figure 6: Confusion Matrix of SVM one VS one.

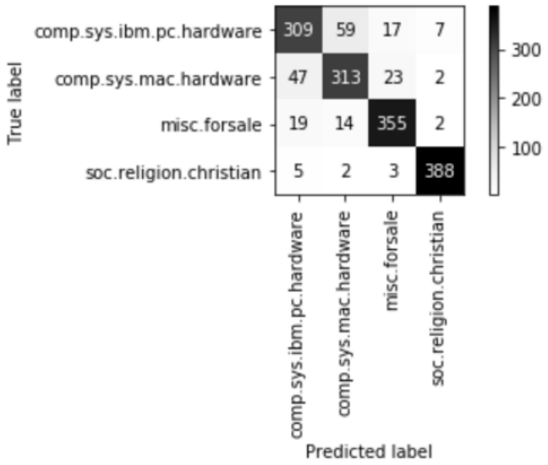

Figure 7: Confusion Matrix of SVM one VS rest.