**ECE 219 Large Scale Data Mining Project 1 Report**

**Group Member:**

**Zhengxiang Yi UID: 304761570**

**Pengyu Chen UID: 704732316**

**Xingquan Guan**

**Dong Wang UID: 804363328**

* Dataset Description

The 20 Newsgroups data set is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. The data is organized into 20 different newsgroups, each corresponding to a different topic. Some of the newsgroups are very closely related to each other (e.g. **comp.sys.ibm.pc.hardware / comp.sys.mac.hardware**), while others are highly unrelated (e.g **misc.forsale / soc.religion.christian**). Here is a list of the 20 newsgroups, partitioned (more or less) according to subject matter: (reference: <http://qwone.com/~jason/20Newsgroups/>)

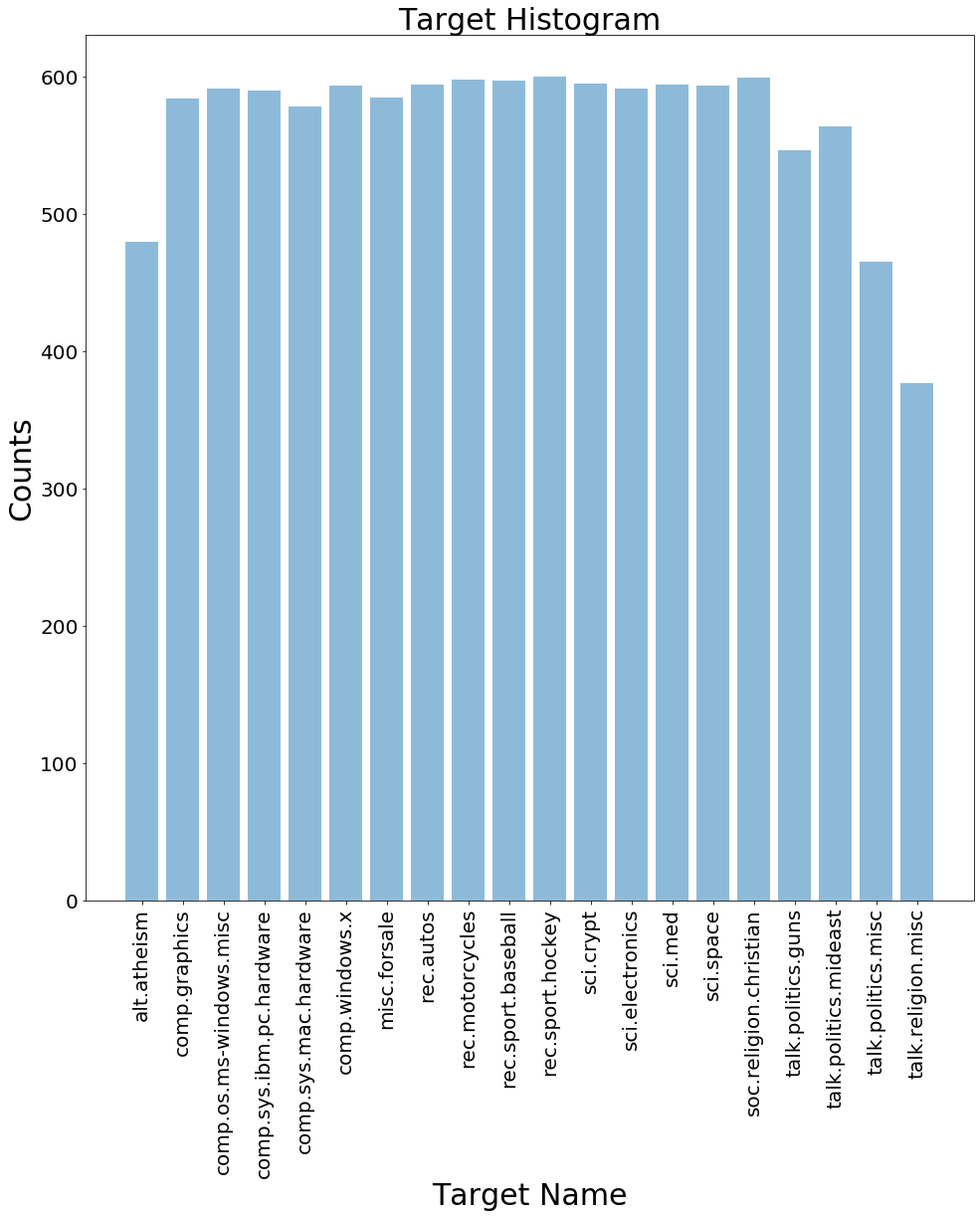
|  |  |  |
| --- | --- | --- |
| comp.graphics  comp.os.ms-windows.misc  comp.sys.ibm.pc.hardware  comp.sys.mac.hardware  comp.windows.x | rec.autos  rec.motorcycles  rec.sport.baseball  rec.sport.hockey | sci.crypt  sci.electronics  sci.med  sci.space |
| misc.forsale | talk.politics.misc  talk.politics.guns  talk.politics.mideast | talk.religion.misc  alt.atheism  soc.religion.christian |

**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.**

The code to generate the histogram and the histogram can be found as following. As shown in the histogram (fig 1), there are totally 20 topics, which includes 'alt.atheism', 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'comp.windows.x', 'misc.forsale', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey', 'sci.crypt', 'sci.electronics', 'sci.med', 'sci.space', 'soc.religion.christian', 'talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc' and 'talk.religion.misc'.

From the overall distribution, there are totally 11314 documents in the dataset. ‘res.sport.hockey’ has the largest number of data, which is 600. Topic ‘talk.religion.misc’ has the least number of data, which is 377. On average, there are 565 documents for each topic.

|  |
| --- |
| # Plot the histogram of the target category in Training set import numpy as np import matplotlib.pyplot as plt #count number and id num\_target = {} for num in newsgroups\_train.target:  if num not in num\_target:  num\_target[num] = 1  else:  num\_target[num] += 1 #match target name and counts name\_target = {} for i in range(len(newsgroups\_train.target\_names)):  name\_target[newsgroups\_train.target\_names[i]] = num\_target[i] |



* Feature Extraction

To extract the features from the document, first we need to tokenize the words. First, we need to clean the text to remove all non-letters. After cleaning the data, use ‘penn2morphy’ function to lemmaitze the words, where we transfer word forms and sometimes derivationally related forms of a word to a common base form. Then, the cleaned text can be tokenized into single words by package nltk. After that, use CountVectorizer to convert the documents to a matrix of token counts. If ‘english’ is used as stop words, a built-in stop word list for English is used to remove uninformative words. Use the vectorizer to vectorize the training and testing data, then apply tf-idf transformer to get the tf-idf matrices for training and testing subsets. The code can be found as following.

|  |
| --- |
| computer\_technology = ['comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware'] recreational\_activity = ['rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey'] train\_dataset = fetch\_20newsgroups(subset = 'train', categories = computer\_technology + recreational\_activity, shuffle = True, random\_state = 42, remove=('headers', 'footers', 'quotes')) test\_dataset = fetch\_20newsgroups(subset = 'test', categories = computer\_technology + recreational\_activity, shuffle = True, random\_state = 42, remove=('headers', 'footers', 'quotes')) #Lemmaitzer 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'  ##tokenizer def tokenizer(text):  clean\_text = re.sub(r'[^A-Za-z]', " ", text)  tokenized\_text = nltk.word\_tokenize(clean\_text)   return tokenized\_text  ##tokenizer & vectorize vectorizer = CountVectorizer(min\_df=3, tokenizer=tokenizer, lowercase=True, stop\_words='english') X\_train\_counts = vectorizer.fit\_transform(train\_dataset.data) X\_test\_counts = vectorizer.transform(test\_dataset.data)  ##tf-idf tfidf\_transformer = TfidfTransformer() X\_train\_tfidf = tfidf\_transformer.fit\_transform(X\_train\_counts) X\_test\_tfidf = tfidf\_transformer.transform(X\_test\_counts) |
|  |

**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.

**The shape of TF-IDF matrix of training set and testing set are (4732, 9429) and (3150, 9429), respectively.**

* Dimensionality Reduction

Consider the fact that the learning algorithm may perform poorly on high dimensional data, LSI and NMF are adopted here to reduce the dimension of TF-IDF matrix. LSI performs Single Value Decomposition to the TF-IDF matrix, then use as the reduced dimension data matrix consider the first k principal components. NMF basically tries to use topic matrix to summarize the documents. Find the reduced dimension matrix by minimizing the distance between TF-IDF matrix to the projection of reduced dimension matrix on the topic matrix. Use the following code to perform dimension reduction.

|  |
| --- |
| svd = TruncatedSVD(n\_components=50, random\_state=0) X\_train\_reduced = svd.fit\_transform(X\_train\_tfidf) X\_test\_reduced = svd.transform(X\_test\_tfidf) #NMF from sklearn.decomposition import NMF model = NMF(n\_components=50, init='random', random\_state=0) W\_train = model.fit\_transform(X\_train\_tfidf) H = model.components\_ W\_test = model.transform(X\_test\_tfidf) |

**QUESTION 3: Reduce the dimensionality of the data using the methods 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 in NMF or the ? Why is the case?

**By using the specified condition, the reduced dimension of training and testing set using LSI and NMF are (4732, 50) and (3150, 50), respectively. The squared residuals of LSI and NMF are 3922 and 3964, respectively. The LSI squared residual is smaller, since when computing single value decomposition, the top 50 most important principal components are kept, which contains the most informative contents of the original matrix. Also, in LSI, all principal components are perpendicular to each other, which intuitively keep the information more efficiently. Since NMF does not adopt eigenvalue computation, there would be some information loss compared to LSI after reducing the same amount of dimensions.**

* Classification Algorithms

Support Vector Machine (Hard & Soft Margin)

Basic idea of support vector machine algorithm: Introduce 'soft margin', where mis-classification is allowed, but we have to limit the number of misclassification. For the misclassification cases, we have or ​, the objective function can be written as . Use hinge loss to substitute the zero-one loss function here since zero-one loss is not differentiable. When , it is the hard margin version of SVM.

**QUESTION 4: Hard margin and soft margin linear SVMs:**

• Train two linear SVMs and compare:

– Train one SVM with γ = 1000 (hard margin), another with γ = 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?

**Hard Margin SVM**

**The accuracy, precision, recall and F1 score of hard margin SVM are 51.0%, 87.5%, 1.34% and 0.027, respectively. The confusion matrix is shown in fig 2. The prediction of hard margin SVM is poor because the original data is not strictly linear separable. A very large trade-off parameter penalizes mis-classification on single data. The algorithm tries best to linear separate the data.**

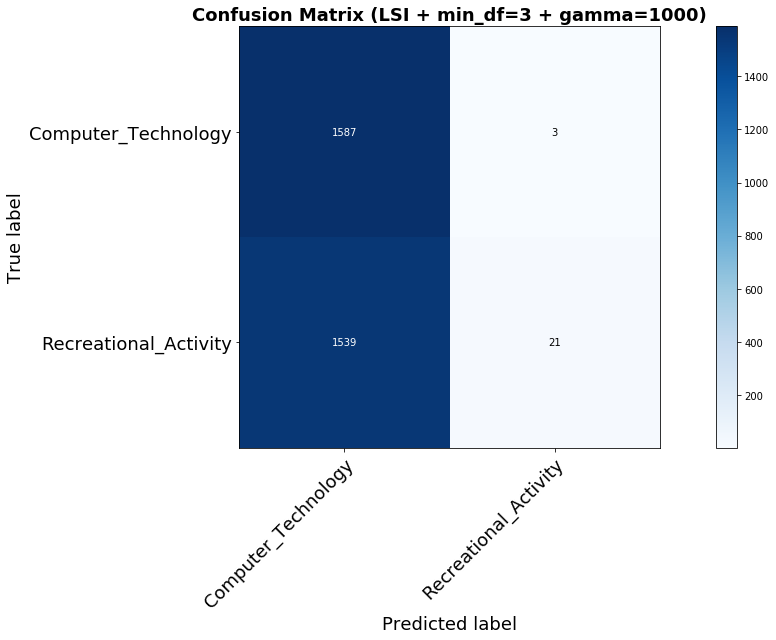


Fig 2. Confusion matrix of gamma = 1000

**Soft Margin SVM**

**The accuracy, precision, recall and F1 score of soft margin SVM are 50.4%, 0%, 0% and 0, respectively. The confusion matrix is shown in fig 3.**

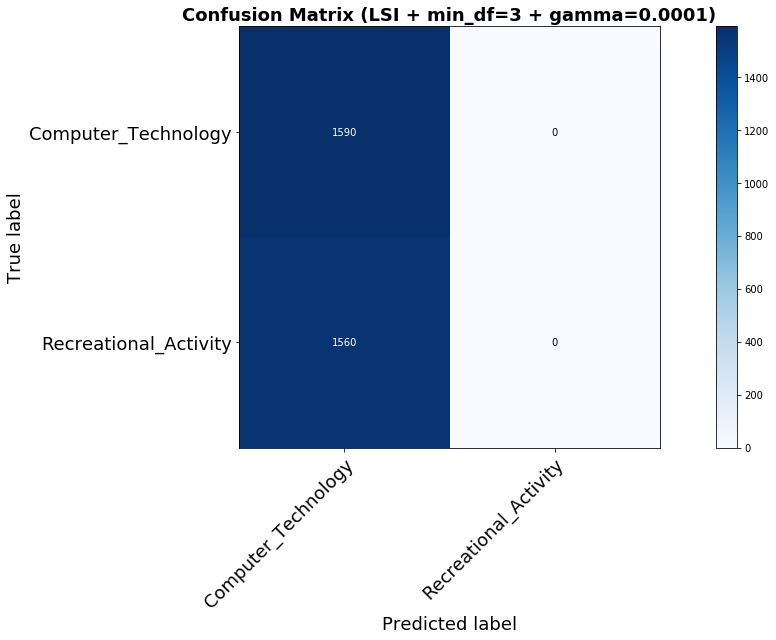


Fig 3. Confusion matrix of gamma = 0.0001

– What happens for the soft margin SVM? Why is the case?

**In the soft margin SVM, all data are classified as the first class such that the precision, recall and F1 score are all 0. The reason is trade-off parameter gamma is too small so that the misclassification of class 2 to class 1 can be totally tolerated. The algorithm would automatically classifies all class 2 to class 1.**

∗ Does the ROC curve of the soft margin SVM look good? Does this conflict with other metrics?

**The ROC curves of hard and soft margin SVM are shown in fig 4. The curves don’t look good, since both curves are close to 45 degree straight line, which represents random guess. ROC of hard margin SVM is slightly higher than soft margin SVM, which infers hard margin SVM is slightly better than soft margin SVM in this case (higher accuracy, precision, recall and F1 score than soft margin SVM). Soft margin SVM is strictly the 45 degree straight line, which is consistent with other metrics (precision, recall and F1 score are all 0). The observations in the ROC curve are not conflict with other metrics.**

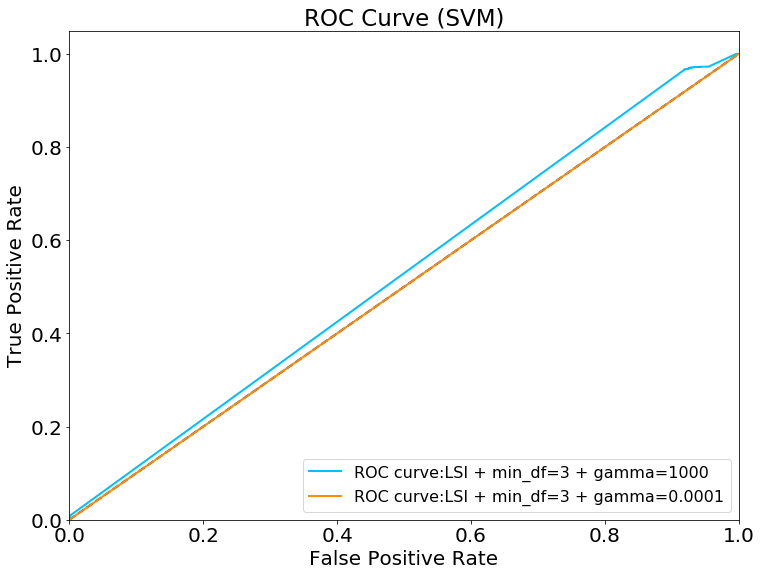


Fig 4. ROC Curve of hard and soft margin SVM

• 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 {10k|−3 ≤ k ≤ 3, k ∈ 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.

**Use the following code to select γ based on highest cross validation score. The best γ is 1.**

|  |
| --- |
| #cross validation to select best r from sklearn.model\_selection import cross\_val\_score r = [0.001,0.01,0.1,1,10,100,1000] average\_score = [] for gamma in r:  clf = svm.SVC(gamma=gamma)  scores = cross\_val\_score(clf, X\_train\_reduced, target\_train, cv=5)  average\_score.append(scores.mean()) best\_r = r[np.argmax(average\_score)] print('Best gamma is:',best\_r) |

**Best gamma is 1 by selecting the model with highest cross-validation score. The corresponding accuracy, precision, recall and F1 score are 92.5%, 97.2%, 87.4% and 0.920 respectively. The confusion matrix and ROC curve are shown in fig 5 and fig 6. Gamma = 1 is the best trade-off parameter selection, it either tolerate misclassification to a fair degree and penalize the misclassification fairly.**

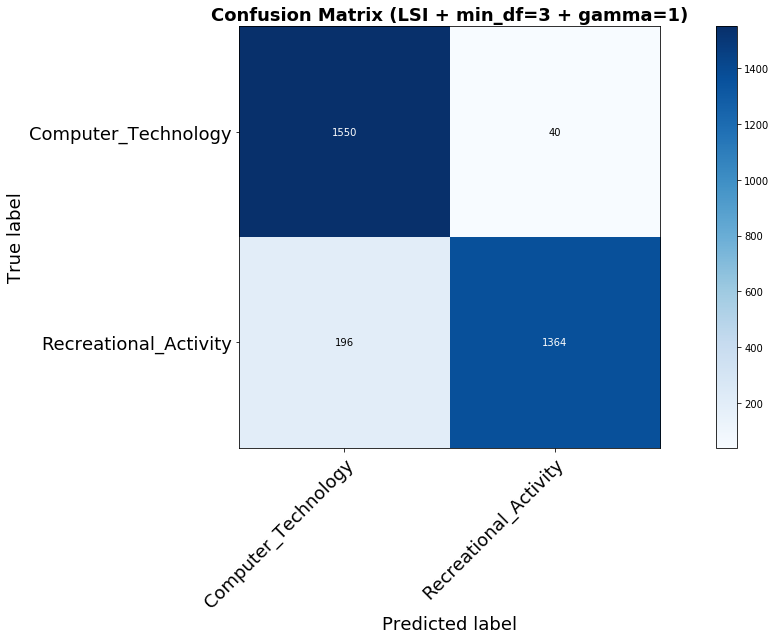


Fig 5. Confusion matrix of gamma = 1 SVM

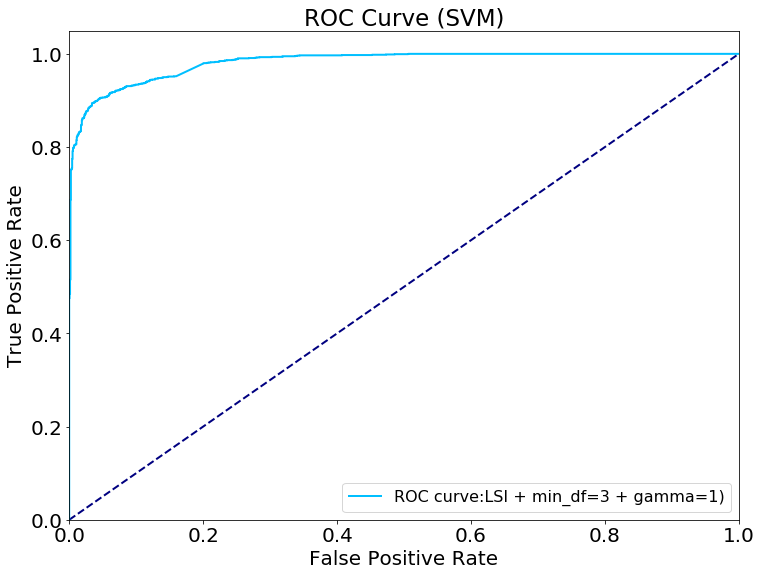


Fig 6. ROC curve of gamma = 1 SVM

**Logistic Regression**

**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.

|  |
| --- |
| #Logistic Regression without Regularization clf = LogisticRegression(C=1e-10) #remove regularization clf.fit(X\_train\_reduced, target\_train) Y\_predict = clf.predict(X\_test\_reduced) #Calculate fpr & ftr  fpr, tpr, thresholds = metrics.roc\_curve(target\_test, clf.predict\_proba(X\_test\_reduced)[:, 1]) fprs.append(fpr) tprs.append(tpr) print("-"\*20+ " Logistic Regression without regularization "+"-"\*20) print("accuracy:",metrics.accuracy\_score(target\_test, Y\_predict)) print("precision:",metrics.precision\_score(target\_test, Y\_predict)) print("recall:",metrics.recall\_score(target\_test, Y\_predict)) print("F-1 score:",f1\_score(target\_test, Y\_predict)) conf\_mat = metrics.confusion\_matrix(target\_test, Y\_predict) print("confusion matrix:",conf\_mat) |

**The regularization term in Logistic Regression package in sklearn is set to be e-10 to remove penalization. The accuracy, precision, recall and F1 score of non-regularized Logistic Regression are 57.7%, 100%, 14.5% and 0.254, respectively. This informs that without regularization, Logistic Regression is overfitting since it has 100% precision but extremely low recall and F1 score.**

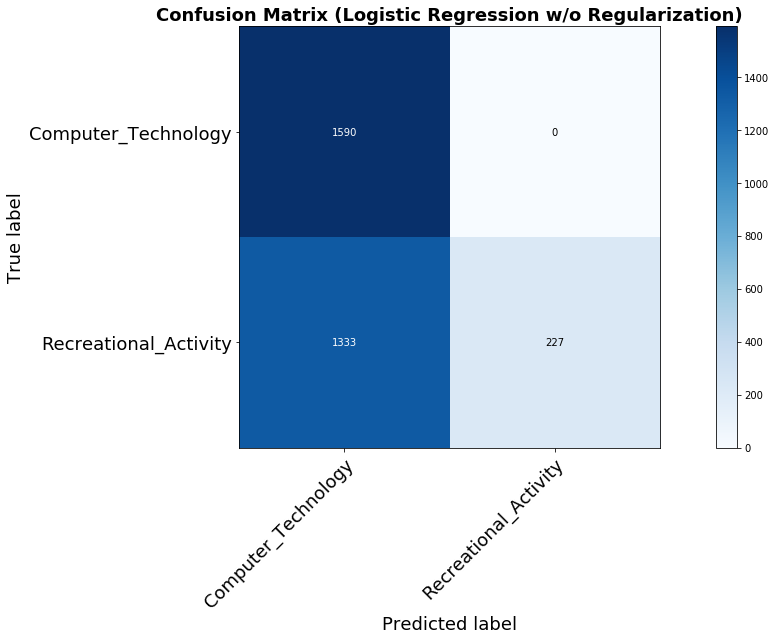


Fig 7. Confusion matrix of Logistic Regression without Penalty

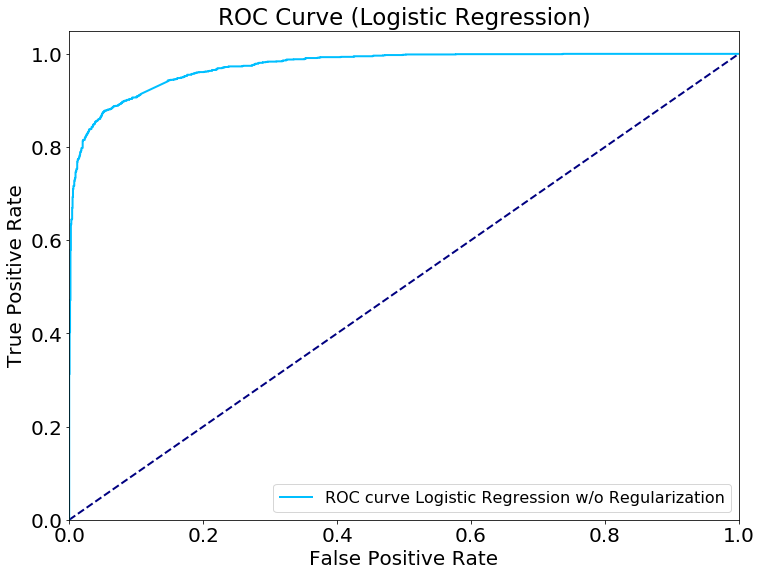


Fig 8. ROC curve of Logistic Regression without Penalty

• Regularization:

– Using 5-fold cross-validation on the dimension-reduced-by-svd training data, find the best regularization strength in the range {10k | − 3 ≤ k ≤ 3, k ∈ Z} for logistic regression with L1 regularization and logistic regression L2 regularization, respectively.

**Use the following code to perform cross validation on penalty term selection. For l1 regularization, the best C is 10; for l2 regularization, the best C is 100.**

|  |
| --- |
| #CV to find best L1 regularization c = [0.001,0.01,0.1,1,10,100,1000] average\_score = [] for C in c:  clf = LogisticRegression(penalty='l1',C=C)  scores = cross\_val\_score(clf, X\_train\_reduced, target\_train, cv=5)  average\_score.append(scores.mean()) best\_C\_l1 = c[np.argmax(average\_score)] print('Best regularization of L1 is:',best\_C\_l1)  #CV to find best L2 regularization c = [0.001,0.01,0.1,1,10,100,1000] average\_score = [] for C in c:  clf = LogisticRegression(penalty='l2',C=C)  scores = cross\_val\_score(clf, X\_train\_reduced, target\_train, cv=5)  average\_score.append(scores.mean()) best\_C\_l2 = c[np.argmax(average\_score)] print('Best regularization of L2 is:',best\_C\_l2) |

– Compare the performance (accuracy, precision, recall and F-1 score) of 3 logistic classi- fiers: w/o regularization, w/ L1 regularization and w/ L2 regularization (with the best parameters you found from the part above), using test data.

**The comparison of performance can be found in table 1. As shown in the table, l1 and l2 regularization has very close performance. L1 performs slightly better in accuracy, recall and F1 score. With regularization logistic regression is much better than without regularization method, since it penalize the magnitude of the coefficients to avoid overfitting.**

**Table 1. Performance comparison of different logistic classifiers**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Without Regularization | W/ L1 | W/ L2 |
| Accuracy | 57.7% | 93.8% | 93.7% |
| Precision | 100% | 96.08% | 96.1% |
| Recall | 14.6% | 91.1% | 90.96% |
| F1 Score | 0.254 | 0.935 | 0.934 |

– 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 l1 and l2 regularization term penalizes coefficients with large magnitude (both positive and negative). If the coefficients’ magnitudes are high, the model is complex, fits the training set better and leads to overfitting. After regularization, the magnitude of coefficients decrease. In such case, the model loses generalization ability and performs badly in testing data. L1 shrinks coefficients that are close to zero to be zero such that one can use it to select parameters considered in the model. L2 shrinks all coefficients by a same amount. L1 regularization is more suitable for dealing with matrix with sparsity to reduce the redundant features. L2 regularization suits generally for penalizing complex models.**

– 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?

**Logistic regression uses generalized linear model to fit the probability falling in a specific class. When the probability is greater than 0.5, it classifies the data into label 1 and when the probability is less than 0.5 it classifies into label 0. In such case, the hyperplane selected by logistic regression is , which corresponds probability equals to 0.5 case. Logistic regression fits the data by maximum likelihood method. Logistic regression penalizes model by regularizing coefficients with high magnitude.**

**Support vector machine tries to find the support vector and finds the hyperplane that maximizes the distance between positive/negative labels to the hyperplane. In SVM, only support vectors play an important role in specifying the hyperplane, and all other data does not contribute to the classification. The hyperplane is specified by maximizing the distance under the constraint . SVM fits the data by coordinate gradient descent method. SVM penalizes model by allowing some level of misclassification.**

**In practice, normal SVM and logistic regression are supposed to have close performance since both of them trying to find a linear hyperplane to separate the data. However, kernel trick can be applied to SVM to project the data into infinite dimension such that the data set may be linear separable.**

**Naive Bayes**

QUESTION 6: Na ̈ıve 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.

**The accuracy, precision, recall and F1 score of GaussianNB classifier are 77.4%, 91.6%, 59.8% and 0.724, respectively. The confusion matrix and ROC curve can be found in fig 9 and 10.**

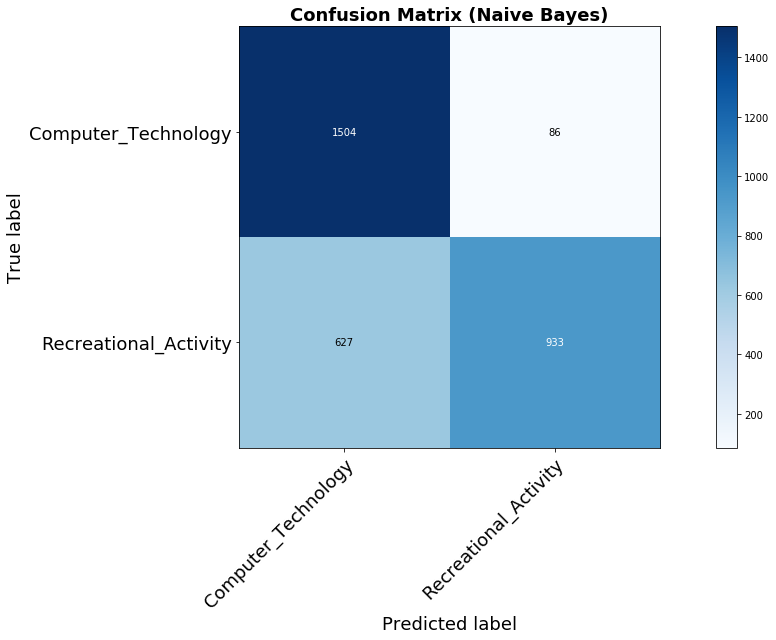


Fig 9. Confusion matrix of Logistic Regression without Penalty

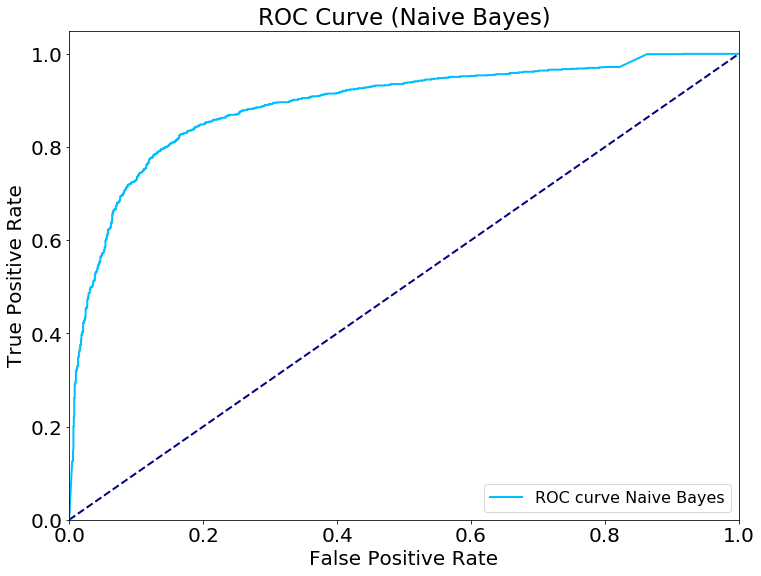


Fig 10. ROC curve of Logistic Regression without Penalty

* Grid Search for Parameters

QUESTION 7: Grid search of parameters:

• Construct a Pipeline that performs feature extraction, dimensionality reduction and classifi-

cation;

• Do grid search with 5-fold cross-validation to compare the following (use test accuracy as the score to compare):

• What is the best combination?

**4 different pipelines are constructed based on removing and keeping lemmatization, removing and keeping header and foot. Training the model using the specified grid pUse the pipeline settings as following code.**

|  |
| --- |
| start = time.time()  cachedir = mkdtemp() memory = Memory(cachedir=cachedir, verbose=10) print("Building Pipeline") #remove\_lemmatized pipeline1 = Pipeline([  ('vect', CountVectorizer(stop\_words='english', tokenizer=lemmatized\_tokenizer)),  ('tfidf', TfidfTransformer()),  ('reduce\_dim', TruncatedSVD(n\_components=50,random\_state=0)),  ('clf', GaussianNB()), ]) #remove\_tokenizer pipeline2 = Pipeline([  ('vect', CountVectorizer(stop\_words='english', tokenizer=tokenizer)),  ('tfidf', TfidfTransformer()),  ('reduce\_dim', TruncatedSVD(n\_components=50,random\_state=0)),  ('clf', GaussianNB()), ]) #not-remove\_lemmatized pipeline3 = Pipeline([  ('vect', CountVectorizer(stop\_words='english', tokenizer=lemmatized\_tokenizer)),  ('tfidf', TfidfTransformer()),  ('reduce\_dim', TruncatedSVD(n\_components=50,random\_state=0)),  ('clf', GaussianNB()), ]) #not-remove\_tokenizer pipeline4 = Pipeline([  ('vect', CountVectorizer(stop\_words='english', tokenizer=tokenizer)),  ('tfidf', TfidfTransformer()),  ('reduce\_dim', TruncatedSVD(n\_components=50,random\_state=0)),  ('clf', GaussianNB()), ]) param\_grid = [  {  'vect\_\_min\_df': min\_df,  'reduce\_dim': [svd, nmf],  'clf': [svm\_clf, logistic\_l1\_clf, logistic\_l2\_clf, GaussianNB\_clf]  } ] |

**The best combination of the four pipelines are reported in table 2 separately. By comparing the mean test score, the best option is keeping headers and footers, adopt lemmatization, logistic regression with l2 penalty with C = 100, using Truncated SVD (LSI) as dimension reduction algorithm with parameter min\_df = 3. This combination gives the highest testing score 98.3%.**

**Table 2. Best Choice for 4 pipelines**

|  |  |  |  |
| --- | --- | --- | --- |
| Pipeline | param\_clf | param\_reduce\_dim | Mean  test\_score |
| Remove headers and footers, lemmatized | LogisticRegression(C=100, class\_weight=None, d... | TruncatedSVD(algorithm='randomized', n\_compone... | 0.935518 |
| Remove headers and footers, not lemmatized | LogisticRegression(C=100, class\_weight=None, d... | TruncatedSVD(algorithm='randomized', n\_compone... | 0.930233 |
| LogisticRegression(C=100, class\_weight=None, d... | TruncatedSVD(algorithm='randomized', n\_compone... | 0.934461 |
| Keep headers and footers, lemmatized | LogisticRegression(C=100, class\_weight=None, d... | TruncatedSVD(algorithm='randomized', n\_compone... | 0.983087 |
| Keep headers and footers, not lemmatized | LogisticRegression(C=100, class\_weight=None, d... | TruncatedSVD(algorithm='randomized', n\_compone... | 0.980973 |

* 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 Na ̈ıve 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.

**Since it’s a multiclass classification problem, the macro precision, recall and F1 score are reported in table 3. The corresponding confusion matrix are shown in fig 11, 12 and 13.**

**Table 3. Multiclass classification performance (macro performance)**

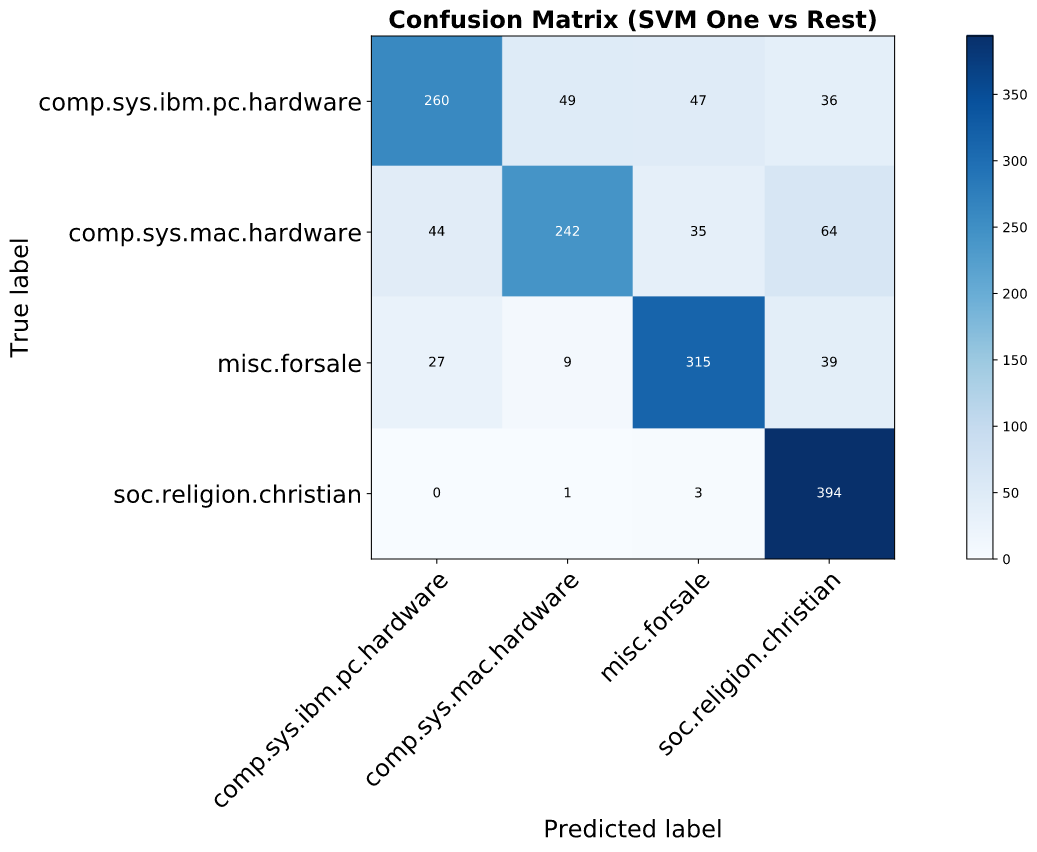
|  |  |  |  |
| --- | --- | --- | --- |
|  | Naive Bayes | SVM (One vs One) | SVM (One vs Rest) |
| Accuracy | 65.6% | 63.1% | 77.4% |
| Precision | 65.4% | 79.8% | 77.9% |
| Recall | 65.4% | 62.8% | 77.2% |
| F1 Score | 0.64 | 0.58 | 0.77 |



**Fig 11. Confusion matrix of Naive Bayes multiclass classifier**



**Fig 12. Confusion matrix of SVM One vs One multiclass classifier**



**Fig 13. Confusion matrix of SVM One vs Rest multiclass classifier**