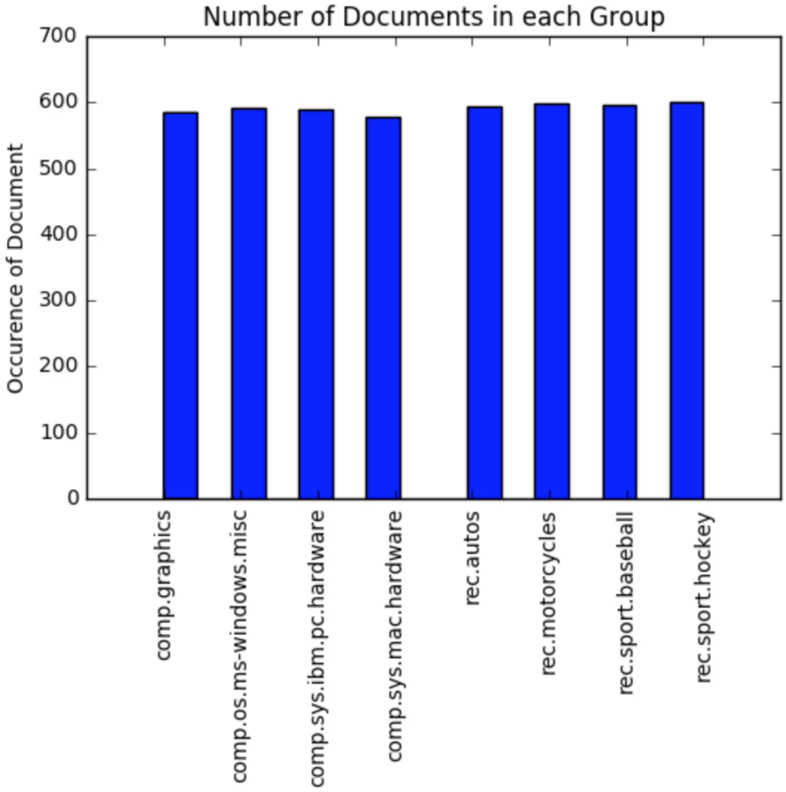
EE219 Project 2

304743326 Andrew Lin, 004587761 Wei-Ting Chen

a) Examine the data to check if it's evenly distributed. The histogram shows the numbers of documents of each subgroup in groups Computer Technology and Recreational Activity, and we can see that it's evenly distributed. The numbers of documents in each group are 2343 and 2389 respectively.



b) Turn the documents into numerical feature vectors.
   i) Number of terms extracted excluding punctuations and stop words: 78910
   ii) Number of terms extracted with additional stemming: 43614

c) TFICF: we calculate TFICF by transforming the feature vectors into classes by terms instead of documents by terms, which is used in calculating TFIDF. Some terms might look a little weird because of stemming, though this improves the accuracy of the model. The followings are the top 10 significant terms in each subgroup:

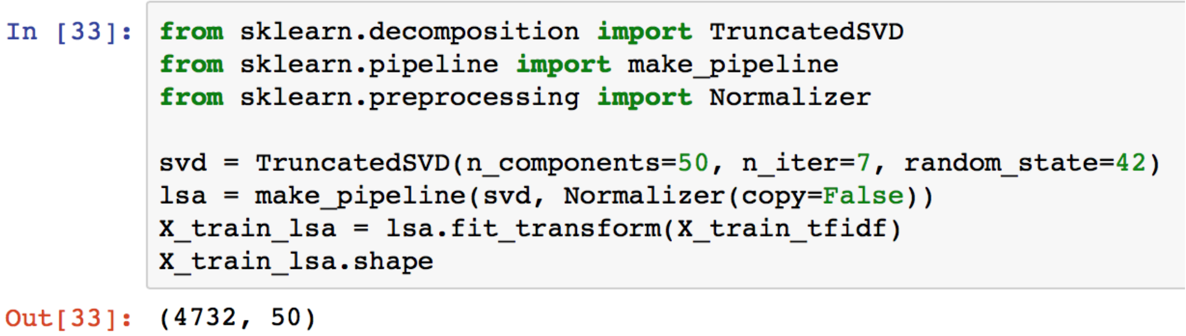
comp.sys.ibm.pc.hardware				
tsuda	thermodynamik	perrier	lorpu86	anyday
ln63sdm	zener	quran	rolex	680040

comp.sys.mac.hardware				
attribut	7b8	1qmnp8inn31v	intent	gh
1094	2dim	ig4	xvoid	287ca4

misc.forsale				
7oo5kjz	hinder	tvar	yesss	nism
6310	dmaluso	mackermit	decmoract	forthwith

soc.religion.christian				
rodal	mudvil	difficuti	f4d	arp0150
sinusoid	censorship	track	perimet	polyn

d) Map the TFIDF of each document to a 50-dimensional vector by applying latent semantic indexing.

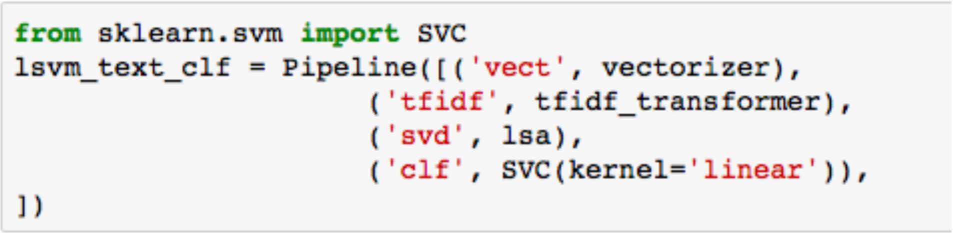


```
In [33]: from sklearn.decomposition import TruncatedSVD
         from sklearn.pipeline import make_pipeline
         from sklearn.preprocessing import Normalizer

         svd = TruncatedSVD(n_components=50, n_iter=7, random_state=42)
         lsa = make_pipeline(svd, Normalizer(copy=False))
         X_train_lsa = lsa.fit_transform(X_train_tfidf)
         X_train_lsa.shape

Out[33]: (4732, 50)
```

Learning Algorithms

We built a data pipeline for vectorizer, transformer, feature selection, and classifier, which makes it easier to run different learning algorithm.

For example, the Pipeline for linear SVM would be:

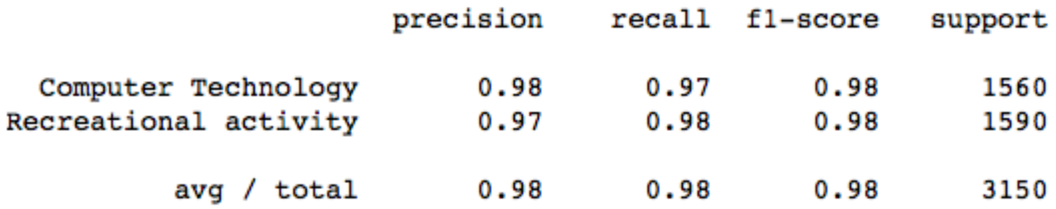


```
from sklearn.svm import SVC
lsvm_text_clf = Pipeline([('vect', vectorizer),
                      ('tfidf', tfidf_transformer),
                      ('svd', lsa),
                      ('clf', SVC(kernel='linear')),
])
```

e) Linear SVM

i) Accuracy: 0.9765

ii) Precision, Recall:

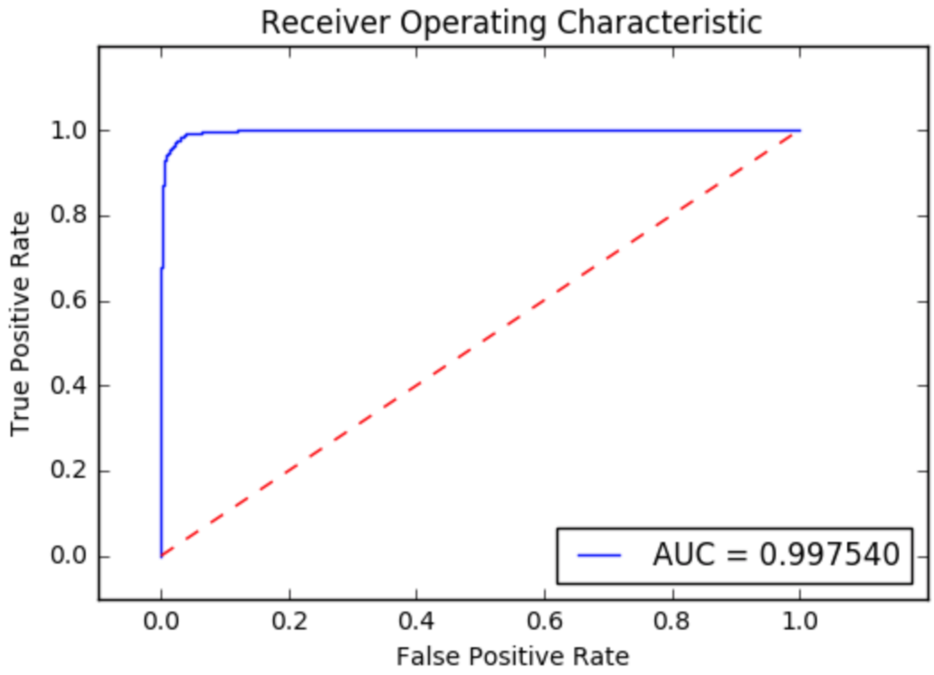


```
                       precision    recall  f1-score   support

  Computer Technology       0.98      0.97      0.98      1560
Recreational activity       0.97      0.98      0.98      1590

          avg / total       0.98      0.98      0.98      3150
```

iii) Confusion Matrix:

Confusion Matrix		Predicted	
		Computer Technology	Recreational Activity
Actual	Computer Technology	1512	48
	Recreational Activity	26	1564

iv) ROC Curve:



AUC in the graph denotes the area under the ROC curve, and the diagonal dashed line being the curve with AUC = 0.5. The plotted ROC curve indicates that the classifier has a low false positive rate while maximizing the true positive rate, which makes it a pretty good classifier.

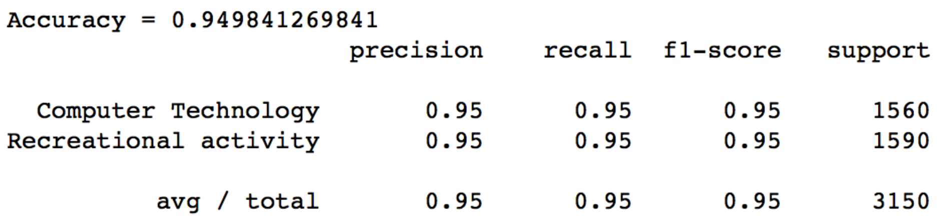
f) Soft Margin SVM

i) Cross Validation: For each □ in range $\{10^{-k} | -3 \leq k \leq 3, k \in Z\}$, we run 5-fold cross validation and pick the model with the best validation accuracy. The following table show the validation accuracy of each fold of each □.

k = -3	k = -2	k = -1	k = 0	k = 1	k = 2	k = 3
0.97571	0.97676	0.97571	0.97465	0.97465	0.96832	0.45828
0.97993	0.97993	0.97888	0.98099	0.97993	0.97254	0.46568
0.97674	0.97674	0.97780	0.97780	0.97145	0.96617	0.51057
0.97674	0.97674	0.97780	0.97991	0.98097	0.97780	0.50845
0.97885	0.97780	0.97780	0.97357	0.97674	0.96828	0.51268

ii) Accuracy: 0.9498

iii) Precision, Recall:

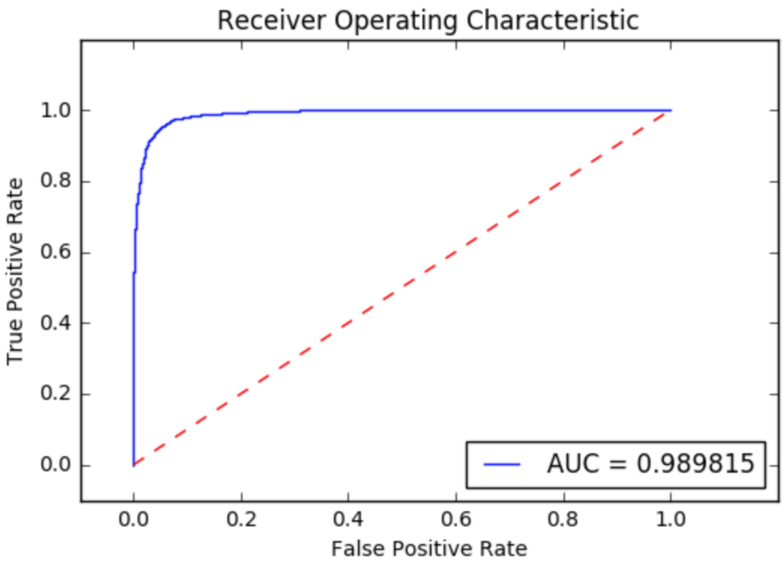


```
Accuracy = 0.949841269841
                           precision    recall  f1-score   support

  Computer Technology         0.95      0.95      0.95      1560
Recreational activity         0.95      0.95      0.95      1590

          avg / total         0.95      0.95      0.95      3150
```

iv) Confusion Matrix:

Confusion Matrix		Predicted	
		Computer Technology	Recreational Activity
Actual	Computer Technology	1477	83
	Recreational Activity	75	1515

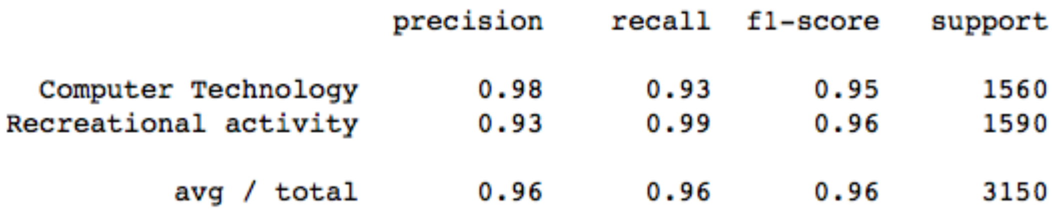
v) ROC Curve:



g) Naive Bayes

i) Accuracy: 0.9556

ii) Precision, Recall:

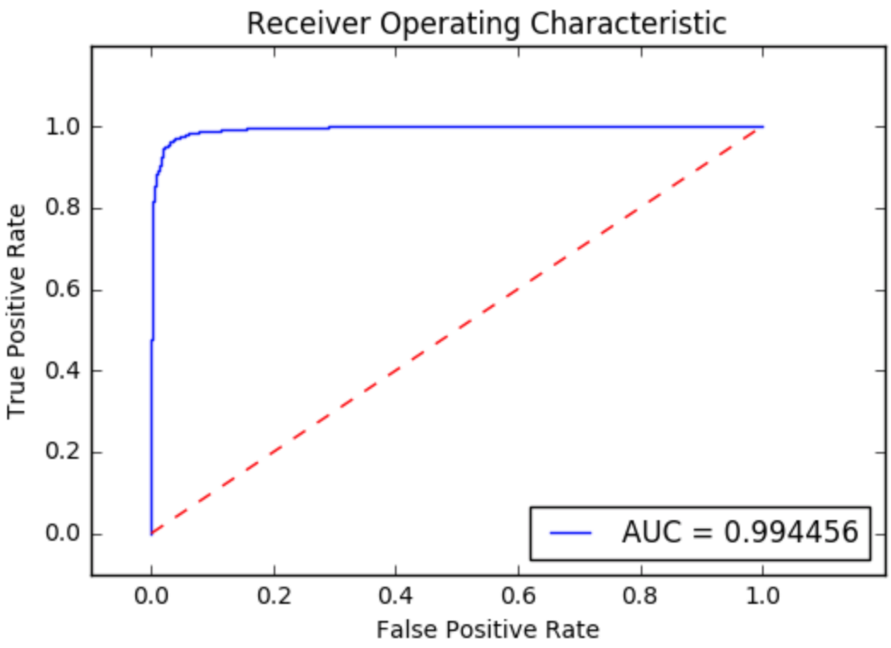


```
                        precision    recall  f1-score   support

   Computer Technology       0.98      0.93      0.95      1560
 Recreational activity       0.93      0.99      0.96      1590

           avg / total       0.96      0.96      0.96      3150
```

iii) Confusion Matrix:

Confusion Matrix		Predicted	
		Computer Technology	Recreational Activity
Actual	Computer Technology	1443	117
	Recreational Activity	23	1567

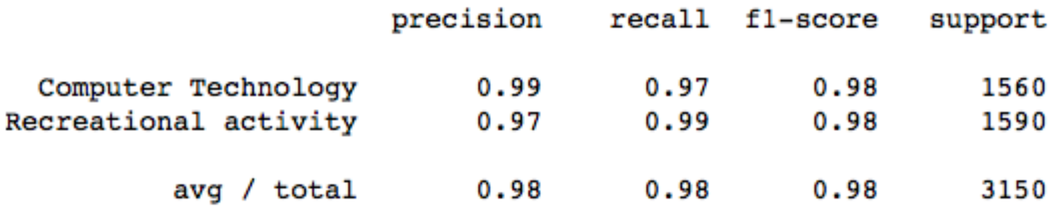
iv) ROC Curve:



h) Logistic Regression

i) Accuracy: 0.9771

ii) Precision, Recall:

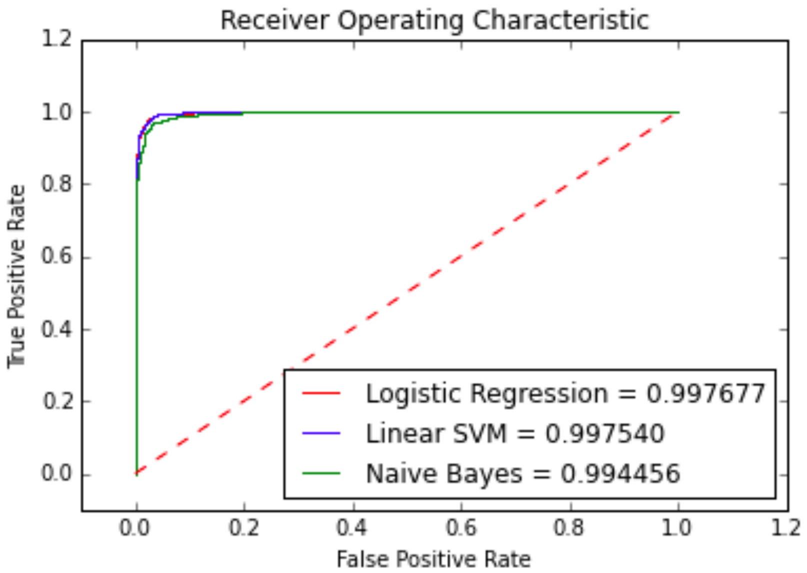


```
                       precision    recall  f1-score   support

   Computer Technology      0.99      0.97      0.98      1560
 Recreational activity      0.97      0.99      0.98      1590

           avg / total      0.98      0.98      0.98      3150
```

iii) Confusion Matrix:

Confusion Matrix		Predicted	
		Computer Technology	Recreational Activity
Actual	Computer Technology	1511	49
	Recreational Activity	23	1567

iv) ROC Curve:

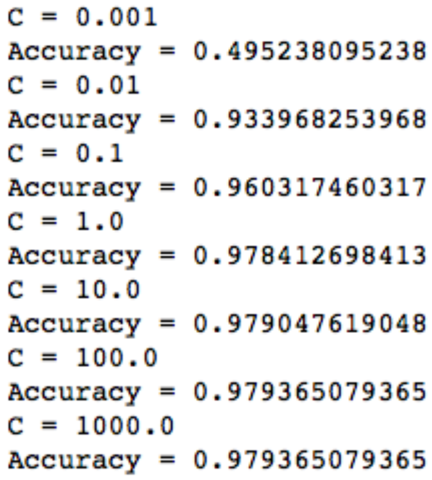


The graph above shows the ROC curve for linear SVM, naive bayes, and logistic regression. We can see from the visualization that these classifiers all have large area under the curve, which indicates that they perform pretty well.

i) Regularized Logistic Regression: For regularized logistic regression, we selected the coefficients within the range of { 10^{-k} | -3 ≤ k ≤ 3, k ∈ Z }.

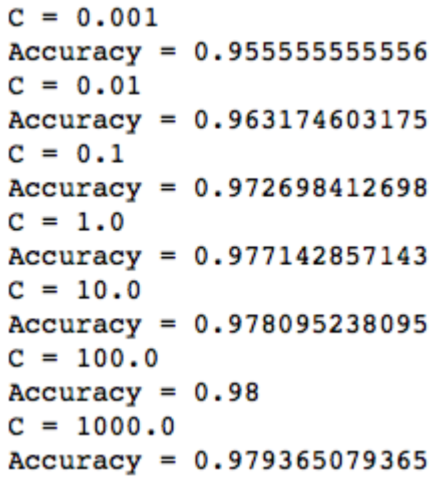
The accuracy with respect to the chosen coefficients C are as follows:

L1 Regularization



```
C = 0.001
Accuracy = 0.495238095238
C = 0.01
Accuracy = 0.933968253968
C = 0.1
Accuracy = 0.960317460317
C = 1.0
Accuracy = 0.978412698413
C = 10.0
Accuracy = 0.979047619048
C = 100.0
Accuracy = 0.979365079365
C = 1000.0
Accuracy = 0.979365079365
```

L2 Regularization

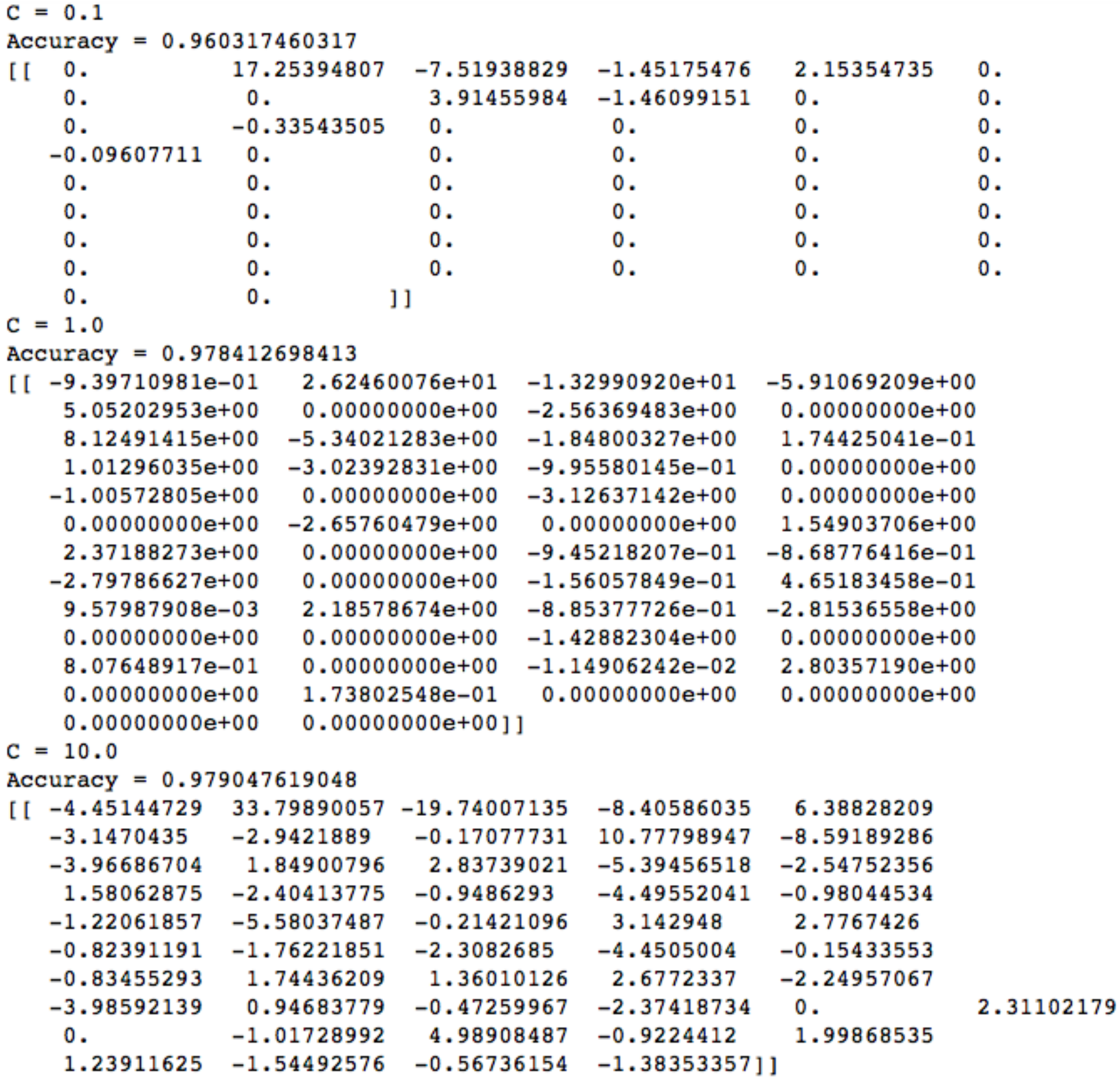


```
C = 0.001
Accuracy = 0.955555555556
C = 0.01
Accuracy = 0.963174603175
C = 0.1
Accuracy = 0.972698412698
C = 1.0
Accuracy = 0.977142857143
C = 10.0
Accuracy = 0.978095238095
C = 100.0
Accuracy = 0.98
C = 1000.0
Accuracy = 0.979365079365
```

We can see that classification accuracy increase when C becomes larger. In Scikit-learn, C is the inverse of regularization strength, where smaller values specify stronger regularization. The significantly low accuracy in L1 regularization for C=0.001 is probably caused by underfitting. Since strong regularization might end up penalizing all the parameters, so all the parameters end up being close to zero.

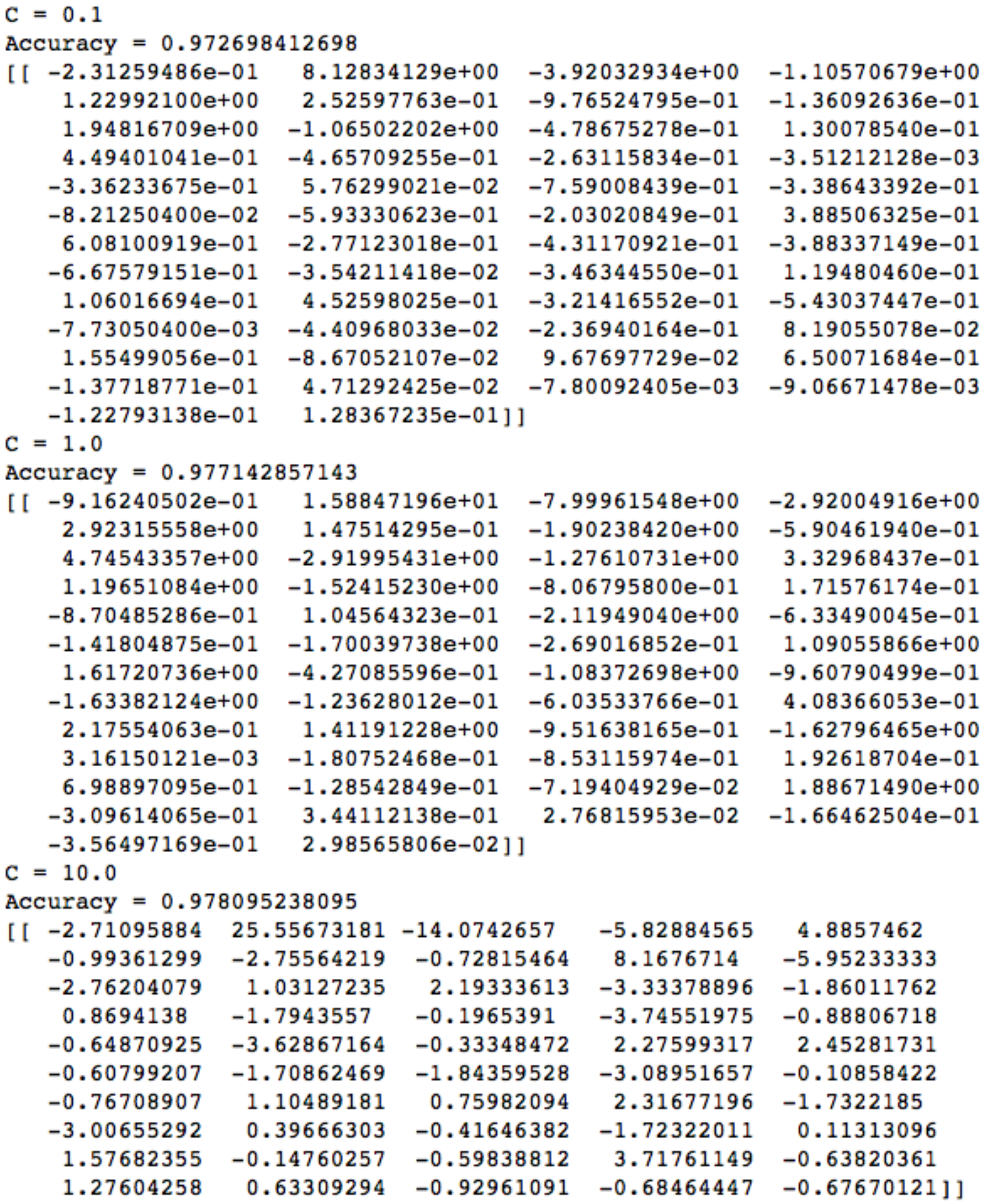
The coefficients of the fitted hyperplane with respect to the chosen coefficients C:

L1 Regularization



```
C = 0.1
Accuracy = 0.960317460317
[[  0.           17.25394807  -7.51938829  -1.45175476   2.15354735   0.
    0.            0.           3.91455984  -1.46099151   0.           0.
    0.           -0.33543505   0.            0.           0.           0.
   -0.09607711    0.           0.            0.           0.           0.
    0.            0.           0.            0.           0.           0.
    0.            0.           0.            0.           0.           0.
    0.            0.           0.            0.           0.           0.
    0.            0.           0.            0.           0.           0.
    0.            0.         ]]
C = 1.0
Accuracy = 0.978412698413
[[ -9.39710981e-01   2.62460076e+01  -1.32990920e+01  -5.91069209e+00
    5.05202953e+00   0.00000000e+00  -2.56369483e+00   0.00000000e+00
    8.12491415e+00  -5.34021283e+00  -1.84800327e+00   1.74425041e-01
    1.01296035e+00  -3.02392831e+00  -9.95580145e-01   0.00000000e+00
   -1.00572805e+00   0.00000000e+00  -3.12637142e+00   0.00000000e+00
    0.00000000e+00  -2.65760479e+00   0.00000000e+00   1.54903706e+00
    2.37188273e+00   0.00000000e+00  -9.45218207e-01  -8.68776416e-01
   -2.79786627e+00   0.00000000e+00  -1.56057849e-01   4.65183458e-01
    9.57987908e-03   2.18578674e+00  -8.85377726e-01  -2.81536558e+00
    0.00000000e+00   0.00000000e+00  -1.42882304e+00   0.00000000e+00
    8.07648917e-01   0.00000000e+00  -1.14906242e-02   2.80357190e+00
    0.00000000e+00   1.73802548e-01   0.00000000e+00   0.00000000e+00
    0.00000000e+00   0.00000000e+00]]
C = 10.0
Accuracy = 0.979047619048
[[ -4.45144729  33.79890057 -19.74007135  -8.40586035   6.38828209
   -3.1470435   -2.9421889   -0.17077731  10.77798947  -8.59189286
   -3.96686704   1.84900796   2.83739021  -5.39456518  -2.54752356
    1.58062875  -2.40413775  -0.9486293   -4.49552041  -0.98044534
   -1.22061857  -5.58037487  -0.21421096   3.142948     2.7767426
   -0.82391191  -1.76221851  -2.3082685   -4.4505004   -0.15433553
   -0.83455293   1.74436209   1.36010126   2.6772337   -2.24957067
   -3.98592139   0.94683779  -0.47259967  -2.37418734   0.           2.31102179
    0.          -1.01728992   4.98908487  -0.9224412    1.99868535
    1.23911625  -1.54492576  -0.56736154  -1.38353357]]
```

L2 Regularization



```
C = 0.1
Accuracy = 0.972698412698
[[ -2.31259486e-01   8.12834129e+00  -3.92032934e+00  -1.10570679e+00
    1.22992100e+00   2.52597763e-01  -9.76524795e-01  -1.36092636e-01
    1.94816709e+00  -1.06502202e+00  -4.78675278e-01   1.30078540e-01
    4.49401041e-01  -4.65709255e-01  -2.63115834e-01  -3.51212128e-03
   -3.36233675e-01   5.76299021e-02  -7.59008439e-01  -3.38643392e-01
   -8.21250400e-02  -5.93330623e-01  -2.03020849e-01   3.88506325e-01
    6.08100919e-01  -2.77123018e-01  -4.31170921e-01  -3.88337149e-01
   -6.67579151e-01  -3.54211418e-02  -3.46344550e-01   1.19480460e-01
    1.06016694e-01   4.52598025e-01  -3.21416552e-01  -5.43037447e-01
   -7.73050400e-03  -4.40968033e-02  -2.36940164e-01   8.19055078e-02
    1.55499056e-01  -8.67052107e-02   9.67697729e-02   6.50071684e-01
   -1.37718771e-01   4.71292425e-02  -7.80092405e-03  -9.06671478e-03
   -1.22793138e-01   1.28367235e-01]]
C = 1.0
Accuracy = 0.977142857143
[[ -9.16240502e-01   1.58847196e+01  -7.99961548e+00  -2.92004916e+00
    2.92315558e+00   1.47514295e-01  -1.90238420e+00  -5.90461940e-01
    4.74543357e+00  -2.91995431e+00  -1.27610731e+00   3.32968437e-01
    1.19651084e+00  -1.52415230e+00  -8.06795800e-01   1.71576174e-01
   -8.70485286e-01   1.04564323e-01  -2.11949040e+00  -6.33490045e-01
   -1.41804875e-01  -1.70039738e+00  -2.69016852e-01   1.09055866e+00
    1.61720736e+00  -4.27085596e-01  -1.08372698e+00  -9.60790499e-01
   -1.63382124e+00  -1.23628012e-01  -6.03533766e-01   4.08366053e-01
    2.17554063e-01   1.41191228e+00  -9.51638165e-01  -1.62796465e+00
    3.16150121e-03  -1.80752468e-01  -8.53115974e-01   1.92618704e-01
    6.98897095e-01  -1.28542849e-01  -7.19404929e-02   1.88671490e+00
   -3.09614065e-01   3.44112138e-01   2.76815953e-02  -1.66462504e-01
   -3.56497169e-01   2.98565806e-02]]
C = 10.0
Accuracy = 0.978095238095
[[ -2.71095884  25.55673181 -14.0742657   -5.82884565   4.8857462
   -0.99361299  -2.75564219  -0.72815464   8.1676714   -5.95233333
   -2.76204079   1.03127235   2.19333613  -3.33378896  -1.86011762
    0.8694138   -1.7943557   -0.1965391   -3.74551975  -0.88806718
   -0.64870925  -3.62867164  -0.33348472   2.27599317   2.45281731
   -0.60799207  -1.70862469  -1.84359528  -3.08951657  -0.10858422
   -0.76708907   1.10489181   0.75982094   2.31677196  -1.7322185
   -3.00655292   0.39666303  -0.41646382  -1.72322011   0.11313096
    1.57682355  -0.14760257  -0.59838812   3.71761149  -0.63820361
    1.27604258   0.63309294  -0.92961091  -0.68464447  -0.67670121]]
```

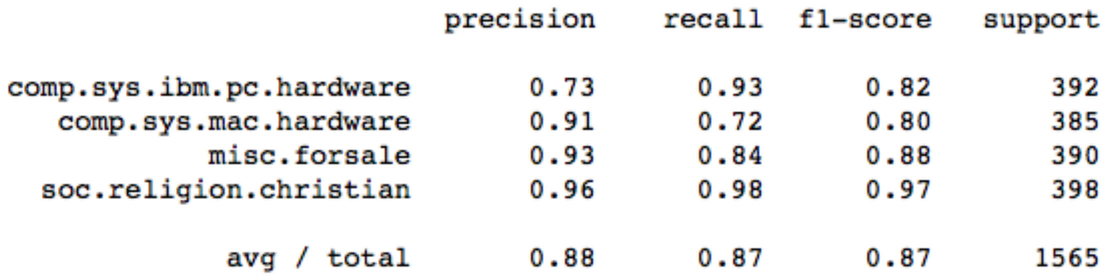
Since stronger regularization results in penalizing more parameters, the coefficients of the fitted hyperplane would have more parameters that are close to zero.

L1 regularization might help perform feature selection in sparse feature spaces, since it has the property of producing many coefficients with values close to zero with few large coefficients. L2 regularization has unique solution while L1 regularization does not, which allows the L2 regularization solutions to be calculated computationally efficiently.

j) Multiclass Naive Bayes

i) Accuracy: 0.8696

ii) Precision, Recall:



```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.73      0.93      0.82       392
   comp.sys.mac.hardware       0.91      0.72      0.80       385
            misc.forsale       0.93      0.84      0.88       390
  soc.religion.christian       0.96      0.98      0.97       398

             avg / total       0.88      0.87      0.87      1565
```

iii) Confusion Matrix:

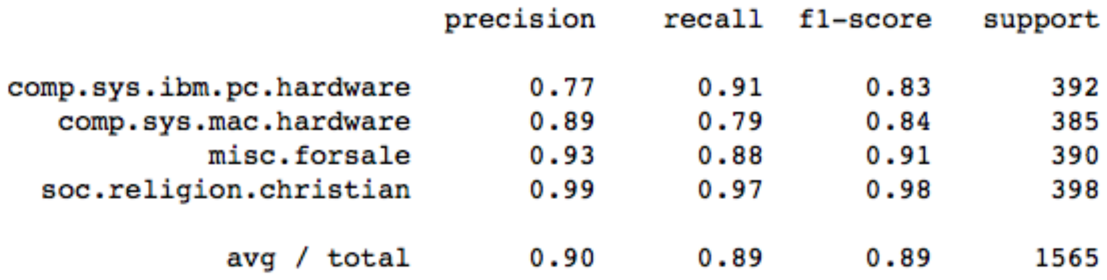
Class1: comp.sys.ibm.pc.hardware
Class2: comp.sys.mac.hardware
Class3: misc.forsale
Class4: soc.religion.christian

Confusion Matrix		Predicted			
		Class1	Class2	Class3	Class4
Actual	Class1	366	15	8	3
	Class2	87	278	13	7
	Class3	43	11	328	8
	Class4	4	2	3	389

k) Multiclass SVM - One vs One

i) Accuracy: 0.8888

ii) Precision, Recall:



```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.77      0.91      0.83       392
   comp.sys.mac.hardware       0.89      0.79      0.84       385
            misc.forsale       0.93      0.88      0.91       390
  soc.religion.christian       0.99      0.97      0.98       398

             avg / total       0.90      0.89      0.89      1565
```

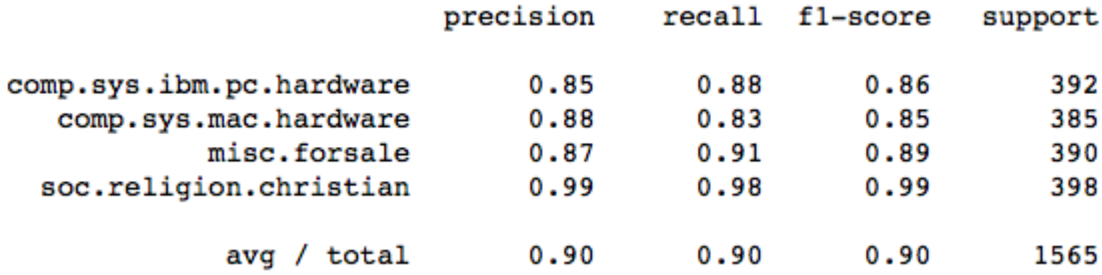
iii) Confusion Matrix:

Class1: comp.sys.ibm.pc.hardware
Class2: comp.sys.mac.hardware
Class3: misc.forsale

Class4: soc.religion.christian

Confusion Matrix		Predicted			
		Class1	Class2	Class3	Class4
Actual	Class1	356	27	9	0
	Class2	67	305	12	1
	Class3	35	11	343	1
	Class4	7	0	4	387

I) Multiclass SVM - One vs Rest
   i) Accuracy: 0.8990
   ii) Precision, Recall:



```
                          precision    recall  f1-score   support

comp.sys.ibm.pc.hardware       0.85      0.88      0.86       392
   comp.sys.mac.hardware       0.88      0.83      0.85       385
            misc.forsale       0.87      0.91      0.89       390
  soc.religion.christian       0.99      0.98      0.99       398

             avg / total       0.90      0.90      0.90      1565
```

   iii) Confusion Matrix:
      Class1: comp.sys.ibm.pc.hardware
      Class2: comp.sys.mac.hardware
      Class3: misc.forsale
      Class4: soc.religion.christian

Confusion Matrix		Predicted			
		Class1	Class2	Class3	Class4
Actual	Class1	343	29	19	1
	Class2	39	318	27	1
	Class3	21	13	355	1
	Class4	1	0	6	391