**EE 219**

**Graphs and Network Flows**

Homework 1 Report

Pingyuan Yue: 504737715

Ke Xu: 604761427

Yuanyi Ding: 404773978

2/12/2017

Table of Contents

[**1. Introduction** 3](#_Toc474858186)

[**2. Dataset and Problem Statement** 3](#_Toc474858187)

[(a) Loading data and histogram plotting 3](#_Toc474858188)

[**3. Modeling Text Data and Feature Extraction** 4](#_Toc474858189)

[(b) Creating Bags of Words and TF-IDF Representation 4](#_Toc474858190)

[(c) 10 Most Significant Features Based on TF-ICF 5](#_Toc474858191)

[**4. Feature Selection** 7](#_Toc474858192)

[(d) LSI Transform 7](#_Toc474858193)

[**5. Learning Algorithms** 8](#_Toc474858194)

[(e) Hard Margin SVM classification 8](#_Toc474858195)

[(f) Soft Margin SVM Classification 10](#_Toc474858196)

[(g) Naive Bayes Algorithm 11](#_Toc474858197)

[(h) Logistic Regression Classification 13](#_Toc474858198)

[(i) Logistic Regression with L-1 and L-2 Norm 15](#_Toc474858199)

[**6. Multiclass Classification** 17](#_Toc474858200)

[(j) Multiclass Classification with Naive Bayes Algorithm and SVM Classification 17](#_Toc474858201)

# Introduction

Classification is a general process related to categorization, the process in which ideas and objects are recognized, differentiated, and understood. In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. An example would be assigning a given email into "spam" or "non-spam" classes or assigning a diagnosis to a given patient as described by observed characteristics of the patient (gender, blood pressure, presence or absence of certain symptoms, etc.). Classification is an example of pattern recognition.

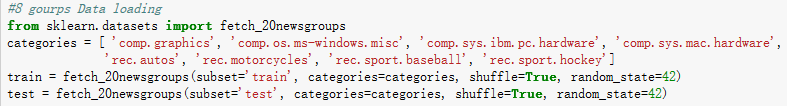
This project mainly works on a practical classification problem on basis of a set of data partitioned across 20 different news group. The data is loaded, reformatted into bags of words. Then features are extracted from the bags of words before it could be trained by different classifiers. In this project, Python is used as primary language. The performance and characteristics are analyzed and compared across different classifiers and coefficients.

# Dataset and Problem Statement

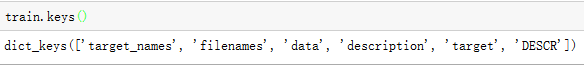
## (a) Loading data and histogram plotting

In a classification problem one should make sure to handle unbalanced datasets properly. To do so, either modify the penalty function or simply sample the majority class randomly, to have the same number of instances as your minority class. To get started, plot a histogram of the number of documents per topic to make sure they are evenly distributed. Then report the number of documents in the two groups above (Computer Technology and Recreational Activity).

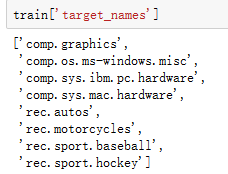
There is a simple built-in function in Python to load the 20 news group data from scikit-learn package. In this case, both training data and testing data in 8 different groups are loaded using the following command:



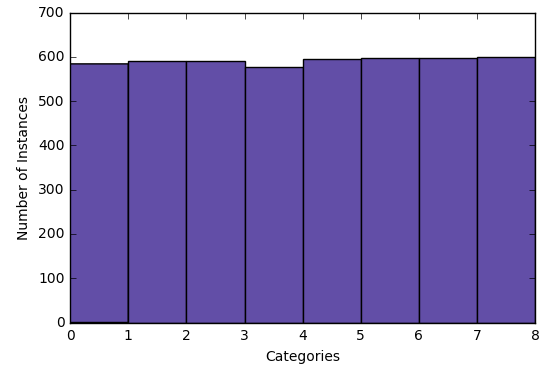
The data is loaded as dictionary. The keys of dictionary could be returned by calling .keys() function:



The target names of the data loaded are the same with the names defined in categories.



After loading the dataset, the histogram of eight different targets could be simply plotted by ‘hist’ function:



From the histogram above, it could be concluded that the 8 categories are almost evenly distributed.

The number of documents of the two classes could be counted using simple iteration:

|  |  |  |
| --- | --- | --- |
| Class | Computer Technology | Recreational Activity |
| # of Documents | 2343 | 2389 |

The documents in two superclasses are also evenly distributed based on the table above.

# Modeling Text Data and Feature Extraction

## (b) Creating Bags of Words and TF-IDF Representation

In the problem b, we have to extract the words in the documents without different stems of a word; then turn the documents in the dataset into numerical feature vectors (a matrix).

The first thing we need to do is to tokenize each document and extract all the words that appear in your documents, excluding the stop words, punctuations, and different stems of a word. Excluding stop words and punctuations could be done by simply using CounterVectorizer() function. For word tokenizing and stemming words excluding, we could implement the word\_tokenize() function and SnowballStemmer in Natural Language Toolkit (nltk) library.

The word\_tokenize() function could help us tokenize the document into separate words. For each word in every document, we call the function in the SnowballStemmer to stem very single word before concatenating all the words into a new document where stemming words are excluded. After excluding stemming word, we could simply call CounterVectorizer() function setting the parameter stop\_words=’english’. The function would automatically exclude all English stop words and punctuations before we could get the bags of word. This could be separated into two different steps with the following code:

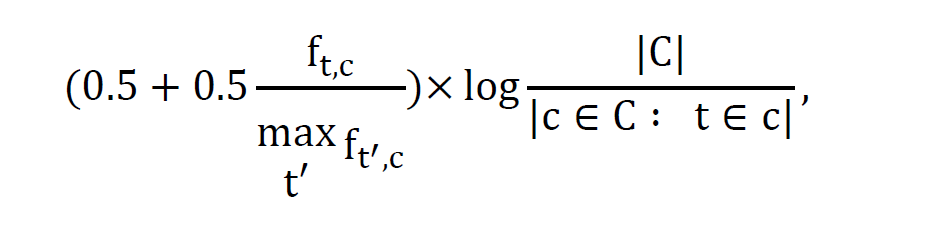


The counter vectorized bags of words have a shape of 4732 documents and 71453 features (i.e. different words in all 4732 documents). The output is a term-document matrix, which represents the occurrence of the word in the documents and also reflects the significance of a word in the documents.

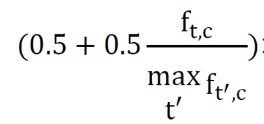
The number of terms of 8 classes we extracted is 71453 and the 8 subclasses are 'comp.graphics', 'comp.os.ms-windows.misc', 'comp.sys.ibm.pc.hardware', 'comp.sys.mac.hardware', 'rec.autos', 'rec.motorcycles', 'rec.sport.baseball', 'rec.sport.hockey.

## (c) 10 Most Significant Features Based on TF-ICF

In order to quantify how significant a word is to a class, we can define a TFxIDF like measure, that we call TFxICF, with a similar definition except that a class sits in place of a document; that is for a term t and a class c, the measure is computed as

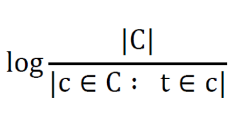


We can divide the above formula into two parts, the first part is



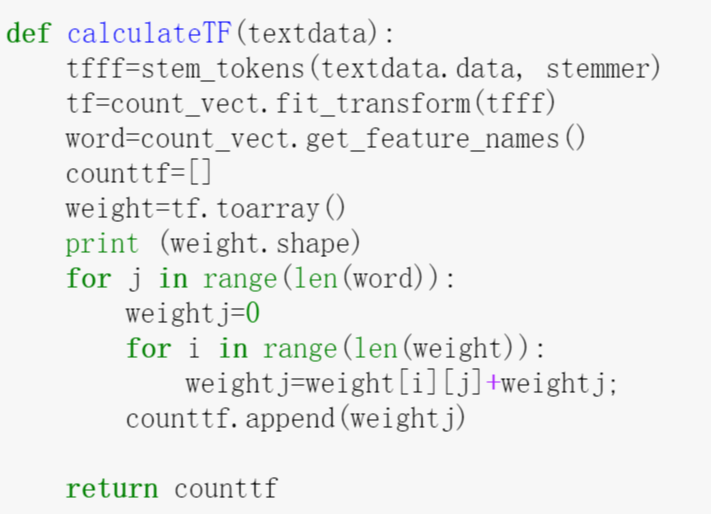
TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document). Based on the assumption that the weight of a term that occurs in a document is simply proportional to the term frequency. the number of times a term occurs in a document is called its term frequency. We might count the number of times each term occurs in each document and sum them all together. which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (the total number of terms in the document) as a way of normalization.

The second part is

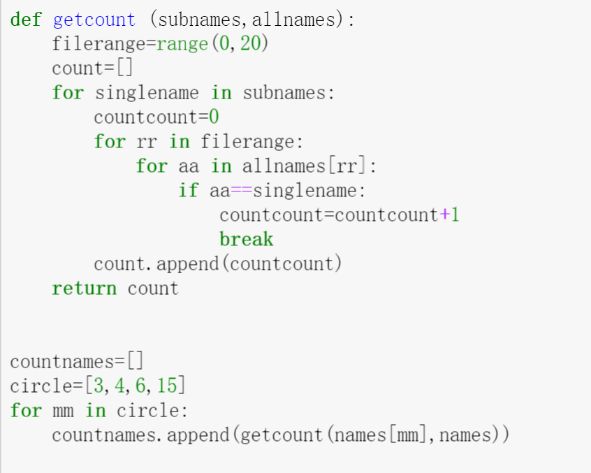


Inverse Class Frequency, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones. If the keyword appears in all classes it is less important.

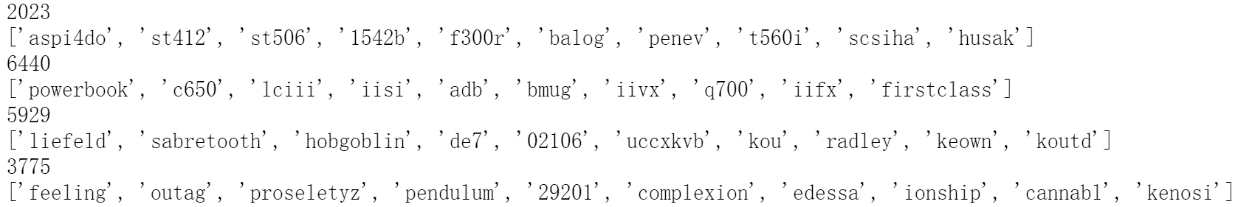
ICF(t) = log\_e(Total number of classes/ Number of classes with term t in it). To calculate the first part, we add up the weight of each word in all the documents in a class then get the term frequency of each word in a class which is the TF.



Then we count the second part. Firstly we get the names of all the words in each class, for every loop, we compare this word with all the word in another class and we add up the times that the other classes also have this word to get the whole times of this word that appear.



By calculating the whole formula we can get the 10 most significant words of each class:



For **comp.sys.ibm.pc.hardware:**

'aspi4do', 'st412', 'st506', '1542b', 'f300r', 'balog', 'penev', 't560i', 'scsiha', 'husak'

For **comp.sys.mac.hardware:**

'powerbook', 'c650', 'lciii', 'iisi', 'adb', 'bmug', 'iivx', 'q700', 'iifx', 'firstclass'

For **misc.forsale:**

'liefeld', 'sabretooth', 'hobgoblin', 'de7', '02106', 'uccxkvb', 'kou', 'radley', 'keown', 'koutd'

For **soc.religion.christian:**

'feeling', 'outag', 'proseletyz', 'pendulum', '29201', 'complexion', 'edessa', 'ionship', 'cannabl', 'kenosi'

# Feature Selection

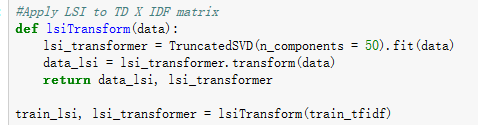
## (d) LSI Transform

Latent semantic analysis (LSA, also known as latent semantic indexing, LSI) is a technique in natural language processing, in particular distributional semantics, of analyzing relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. LSA assumes that words that are close in meaning will occur in similar pieces of text. A matrix containing word counts per paragraph (rows represent unique words and columns represent each paragraph) is constructed from a large piece of text and a mathematical technique called singular value decomposition (SVD) is used to reduce the number of rows while preserving the similarity structure among columns. Words are then compared by taking the cosine of the angle between the two vectors (or the dot product between the normalizations of the two vectors) formed by any two rows. Values close to 1 represent very similar words while values close to 0 represent very dissimilar words.

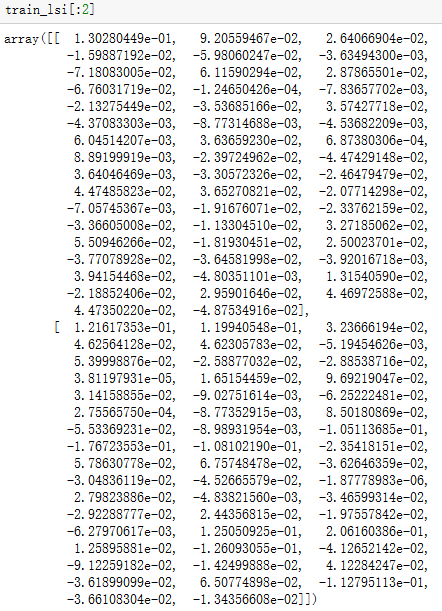
In this project, we use Latent Semantic Indexing (LSI), a dimension reducing transform that finds the optimal representation of the data in a lower dimensional space in the mean squared error sense. Here we represent the data in the term-document matrix, whose columns corresponds to TFxIDF representation of the documents.

In scikit-learn library, a built-in function TruncatedSVD could help us perfectly. TruncatedSVD implements a variant of singular value decomposition (SVD) that only computes the k largest singular values, where k is a user-specified parameter. When truncated SVD is applied to term-document matrices (as returned by CountVectorizer or TfidfVectorizer), this transformation is known as latent semantic analysis (LSA), because it transforms such matrices to a “semantic” space of low dimensionality. In particular, LSA is known to combat the effects of synonymy and polysemy (both of which roughly mean there are multiple meanings per word), which cause term-document matrices to be overly sparse and exhibit poor similarity under measures such as cosine similarity.

In this project, a function lsiTransform() and transmitted the original dataset's TfidfVectorizer into it to achieve the Latent Semantic Indexing by the algorithm of Singular Value Decomposition (SVD) and finally had the lower dimensional features(n\_components = 50) which had latent semantics for the text but not in terms of words. The code is shown as follows:



The input data must already be transformed into TF-IDF representation. Both the 50-dimensional extracted dataset and fitted transformer are returned. The extracted features of the first 5 element are shown below:

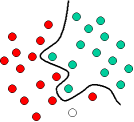


# Learning Algorithms

## (e) Hard Margin SVM classification

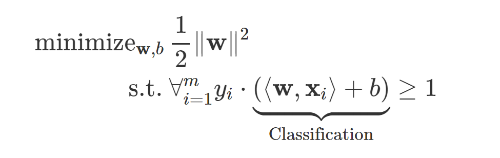
Support Vector Machines (SVMs) is a group of powerful classifiers shows efficiency when dealing with the sparse high dimensional datasets.

Illustration (a) is a linear classifier that separates a set of objects into their respective groups with a line.



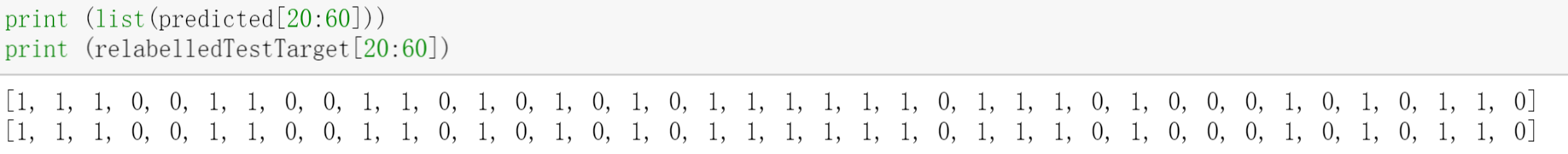
However, many other classification examples are often more complex and are needed in order to make an optimal separation, i.e., correctly classify test cases based on the examples that are available. This situation is the illustration (b). It is clear that a separation of the objects would require a curve not a line. Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks. We often use SVMs with scikit-learn to deal with the case.

In Support Vector Machines, we want to map the data into the points in space, then it can be widely divided and easier to be classified. We want the gap between groups of points as clear as possible. If data is linearly separable, it can be separated by a hyperplane. There is one hyperplane which maximizes the distance to the next data points (support vectors). This hyperplane should be:



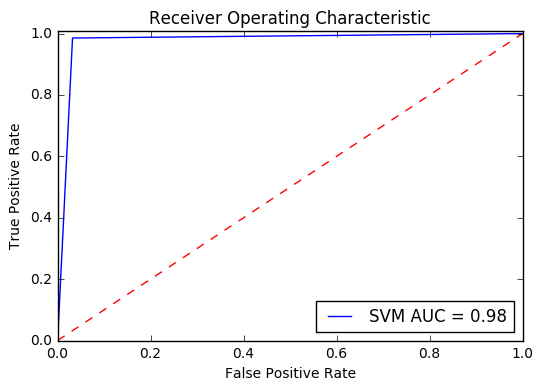
So we use the LSI representation of data as X, relabel the target of 8 subclasses (0-7) to 2 classes (0&1) and use the relabeled target as Y. Though the dimension of X is 50, much less than the dimension of Y, since the advantages of support vector machines is that it is still effective in cases where number of dimensions is greater than the number of samples, we can use SVM to fit the two datasets.

Then we use the trained classifier to predict the LSI representation of test dataset and we can see the prediction is almost the same as the true target.



And now we want to evaluate the classifier using ROC curve, confusion matrix and calculate the accuracy, recall and precision.

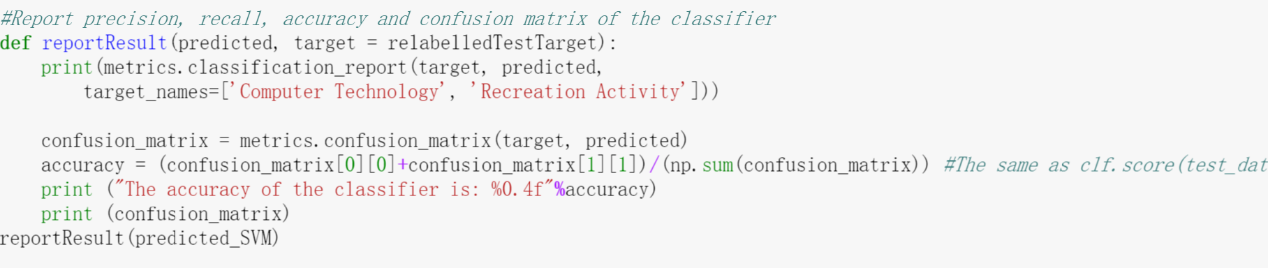
In statistics, a receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. The curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The ROC of the SVM classifier is shown below:



\*(The ROC along with other classifiers would be shown in problem (h))

From the ROC plotted above, we can see the turning point of the ROC is very close to the TPR axis indicating that the classifier works pretty well in this case.

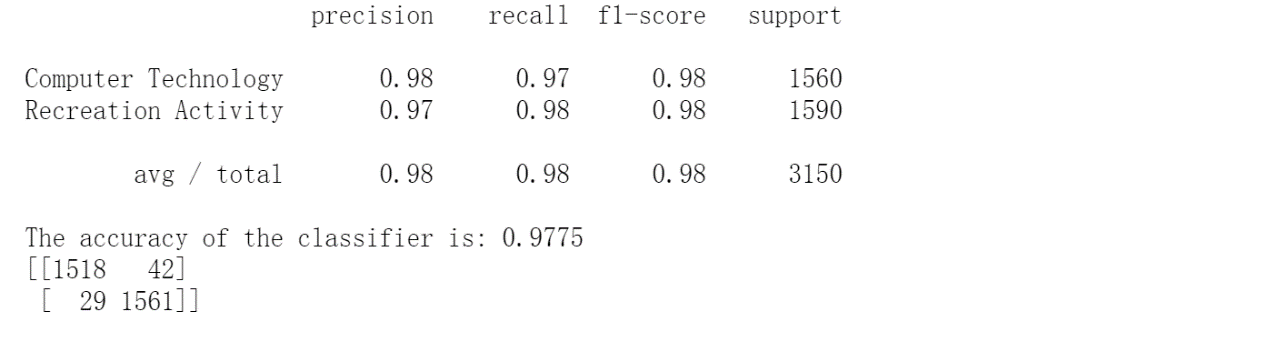
Then we report the confusion matrix and other evaluation of the classifier.



From the confusion matrix below, we got a very high TPR, TNR and low FNR, FPR, which is a very good classification.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Total population | | Predicted positive | | Predicted negative | |
| True  condition | | **condition**  **positive** | | 1510 | | 50 | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | condition  negative | 23 | 1567 |



From the result above, it could be concluded that the SVM classifier works well in this problem.

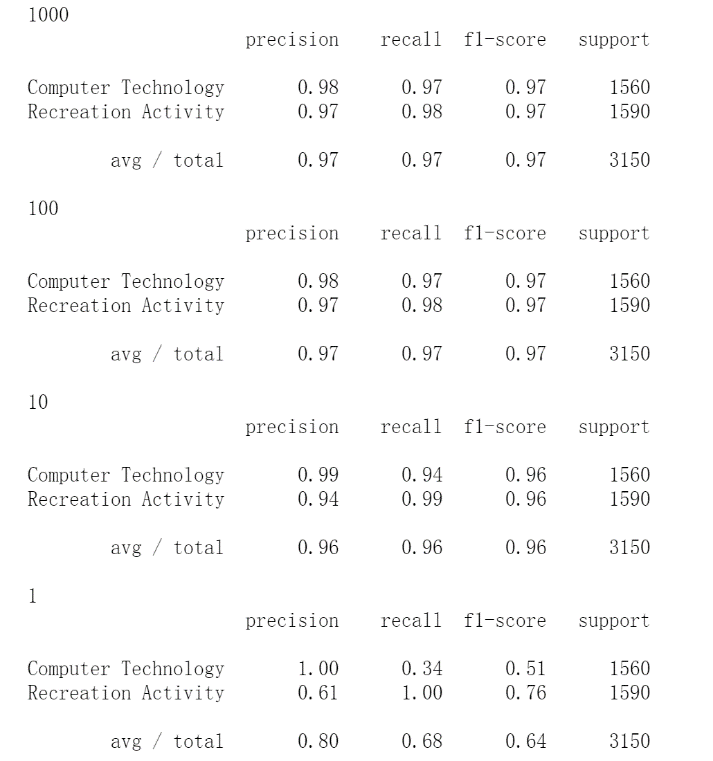
## (f) Soft Margin SVM Classification

We would expect soft-margin SVM to be better even when training dataset is linearly separable. The reason is that in a hard-margin SVM, a single outlier can determine the boundary, which makes the classifier overly sensitive to noise in the data. The allowance of softness in margins (i.e. a low cost setting) allows for errors to be made while fitting the model (support vectors) to the training/discovery data set. Conversely, hard margins will result in fitting of a model that allows zero errors. Sometimes it can be helpful to allow for errors in the training set, because it may produce a more generalizable model when applied to new datasets. Forcing rigid margins can result in a model that performs perfectly in the training set, but is possibly over-fit or less generalizable when applied to a new dataset. Identifying the best settings for 'cost' is probably related to the specific data set you are working with.

Here, we repeat the pervious part with the soft margin SVM and, using a 5-fold cross-validation, find the best value of the parameter 𝛾 in the range {10−𝑘|−3≤𝑘≤3,𝑘∈𝑍}. Report the confusion matrix and calculate the accuracy, recall and precision of your classifier.

Cross-validation, sometimes called rotation estimation, is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. It is mainly used in settings where the goal is prediction, and one wants to estimate how accurately a predictive model will perform in practice. In a prediction problem, a model is usually given a dataset of known data on which training is run (training dataset), and a dataset of unknown data (or first seen data) against which the model is tested (testing dataset)

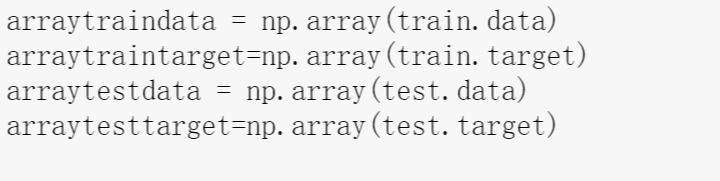
Separately, we do the cross validation using cross\_val and kFold. The simplest way to use cross-validation is to call the cross\_val\_score helper function on the estimator . The following example demonstrates the accuracy of support vector machine on dataset by splitting the data, fitting a model and computing the score 5 consecutive times (with different splits each time):

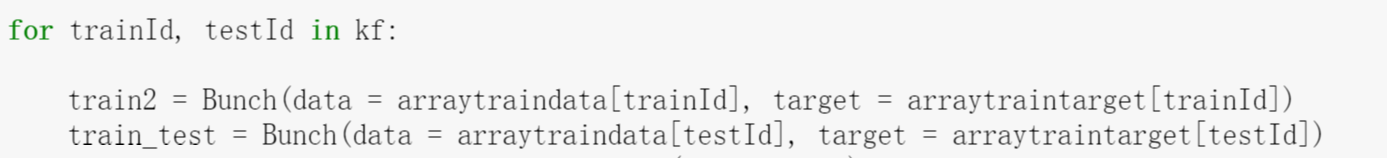


As we can see, the accuracy is lowering when C is getting smaller.

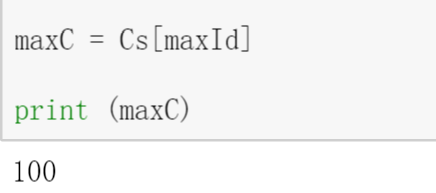
Then we split the data using K-Folds cross-validator which provides train/test indices to split data in train/test sets. Split dataset into k consecutive folds (without shuffling by default). Each fold is then used once as a validation while the k - 1 remaining folds form the training set.

Firstly, we transform the data and target to array and for each circle we extract the index of train and test:





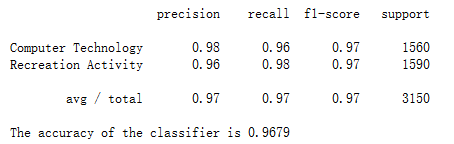
And we get the best 𝛾 is when K=-2, 𝛾 is 100.



Report of the confusion matrix and calculate the accuracy, recall and precision of your classifier is as below:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Total population | | Predicted positive | | Predicted negative | |
| True  condition | | **condition**  **positive** | | 1495 | | 65 | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | condition  negative | 36 | 1554 |

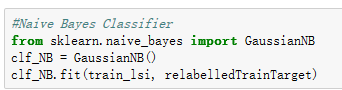


From the result above, we can conclude that the performance of soft margin SVM classifier is good enough, but slightly worse than hard margin SVM we talked about in problem (e).

## (g) Naive Bayes Algorithm

Naïve Bayes algorithm is a classification technique based on Bayes’ Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as ‘Naive’.

Naïve Bayes classifiers are a family of probabilistic classifiers using the theorem of Bayes with strong independence assumption between features. And it is very popular dealing with the text categorization, which concerns judging documents as belonging to different categories with the terms frequencies as the features. Thus it is very suitable for the case in this problem. The given vector set (feature) is statistically independent, though it is the map of the high-dimensional term frequency to the low-dimensional space.



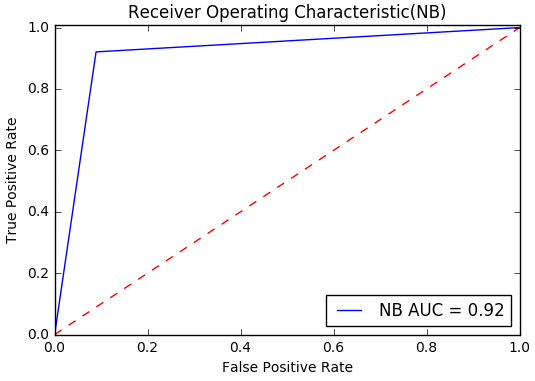
Naïve Bayes Classifier is easy to build in Python using the function MultinomialNB and GaussianNB. However, MultinomialNB function could only process input data with non-negative value. In this case, GaussianNB is chosen to create Naïve Bates classifier.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). Look at the equation below:



where P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes); P(c) is the prior probability of class; P(x|c) is the likelihood which is the probability of predictor given class; P(x) is the prior probability of predictor.

In this problem, it is assumed that the features follow a normal distribution and Naïve Bayes Classifier is created using GaussianNB(). The True Positive Rate(TPR) and False Positive Rate(FPR) could be calculated before obtaining the receiver operating curve(ROC). The ROC got by Naïve Bayes Classifier is shown below(the ROC along with other algorithms is shown in problem h):

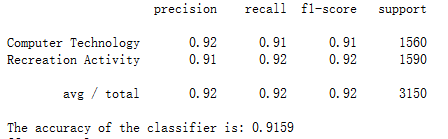


From the above figure, it is very clear that the turning point is not so close to the TPR axis comparing with SVM classifier, which indicates the distinction between the two classes trained with Naïve Bayes Classifier are not very clear and they are not so widely divided. On that regard, the Naïve Bayes classifier does not work very well on this dataset. To justify this, the confusion matrix is provided with calculated precision, recall and accuracy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Total population | Predicted positive | Predicted negative | |
| True  condition | **condition**  **positive** | 1421 | | 139 |
| **condition**  **negative** | 126 | | 1464 |

From the confusion matrix above, it could be concluded that both the True Positive Rate and the False Positive Rate is a little bit lower. That means that Naïve Bayes is not a good classifier in this problem which coordinates with what is predicted from ROC above.

The precision, recall and accuracy could also be obtained as follows:



All the three measurements are lower than the SVM classifier which is exactly the same with the outcome of the confusion matrix. In this case, SVM classifier works better than Naïve Bayes model.

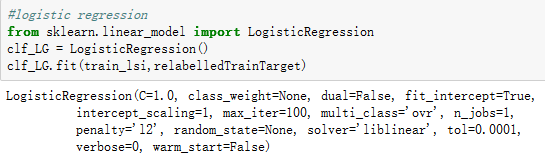
## (h) Logistic Regression Classification

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum-entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

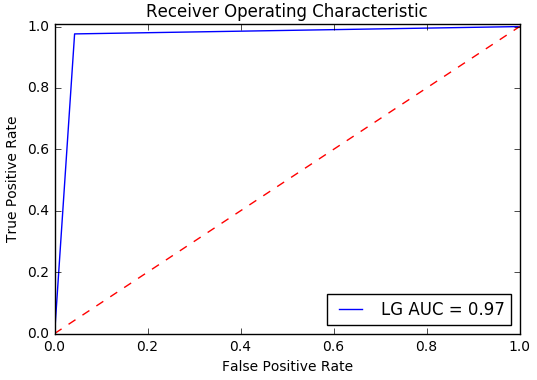
Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. In the latent variable interpretations of these two methods, Logistic regression assumes a standard logistic distribution of errors.

Logistic regression has two features which make it quite different from other linear regression. First, the conditional distribution y|x is a Bernoulli distribution rather than a Gaussian distribution, because the dependent variable is binary. Second, the predicted values are probabilities and are therefore restricted to (0,1) through the logistic distribution function because logistic regression predicts the probability of particular outcomes.

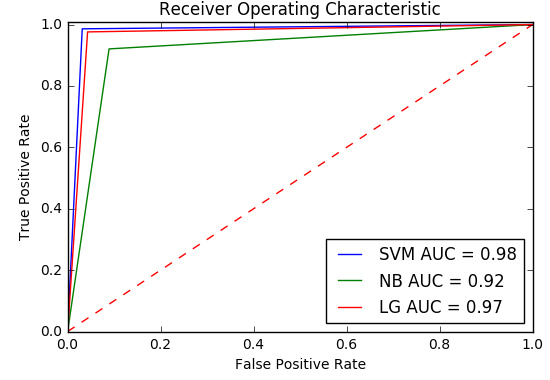
In Python, logistic regression could be simply created by using the built-in function LogisticRegression (). And the characteristics of this classifier are analyzed using defined function reportResult ().



Then the true positive rate and false positive rate are calculated to plot the receiver operating curve of the logistic regression model:



The comparison among ROC of the three different models is shown as follows:

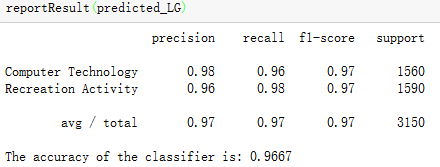


From the ROC plotted above, we can see that the logistic regression model works well comparing with the SVM model and Naive Bayes classifier. The confusion matrix and evaluations of the matrix are shown below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Total population | Predicted positive | Predicted negative |

|  |  |  |  |
| --- | --- | --- | --- |
| True  condition | **condition**  **positive** | 1491 | 69 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | condition  negative | 37 | 1553 |



It could be concluded from the measurement above that Logistic Regression Classifier works a little bit worse than SVM classifier, but better than the Gaussian Naive Bayes model. To summarize, at the same False Positive Rate, soft-margin linear SVM has slighter bigger True Positive Rate than logistic regression and the smallest True Positive Rate belongs to the Naive Bayes model. In conclusion, for this problem, the performance of the three different classifiers from best to worst is:

SVM Classifier > Logistic Regression Classifier > Naive Bayes Classifier

## (i) Logistic Regression with L-1 and L-2 Norm

The implementation of logistic regression in scikit-learn can be accessed from class LogisticRegression. This implementation can fit binary, OnevsRest, or multinomial logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

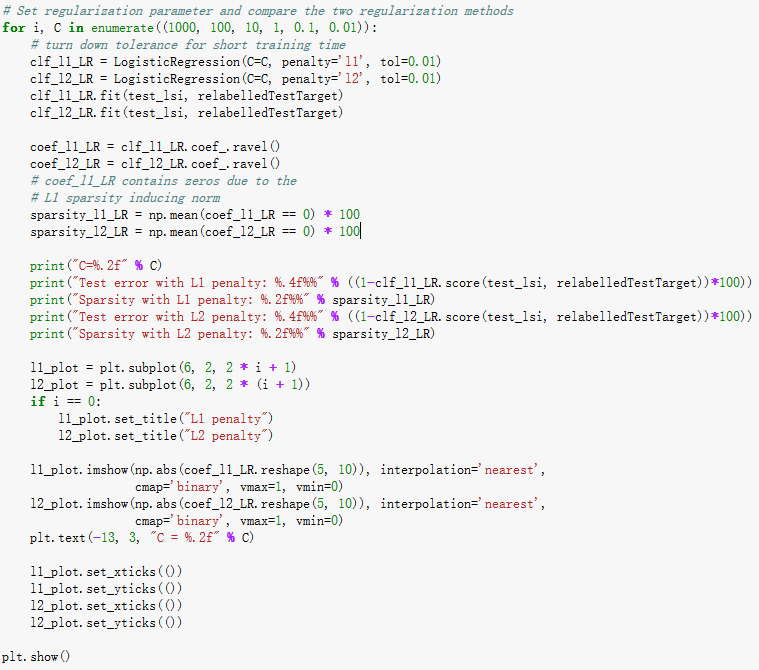


Similarly, L1 regularized logistic regression solves the following optimization problem



By simply changing the ‘penalty’ parameter to ‘l1’ or ‘l2’, we could specify L-1 regularization or L-2 regularization for the classifier respectively.

To compare the different behaviors of the classifier, we try both L-1 and L-2 regularization and sweep through different regularization coefficients, ranging from 0.01 to 1000. This could be implemented using the following code:



Comparison of the sparsity (percentage of zero coefficients) of solutions when L1 and L2 penalty are used for different values of C. We can see that large values of C give more freedom to the model. Conversely, smaller values of C constrain the model more. In the L1 penalty case, this leads to sparser solutions. The results of different behaviors under different regularizations and coefficients are listed below:

C=1000.00 C=100.00

Test error with L1 penalty: 1.8730% Test error with L1 penalty: 2.0317%

Sparsity with L1 penalty: 0.00% Sparsity with L1 penalty: 0.00%

Test error with L2 penalty: 1.9048% Test error with L2 penalty: 2.0952%

Sparsity with L2 penalty: 0.00% Sparsity with L2 penalty: 0.00%

C=10.00 C=1.00

Test error with L1 penalty: 2.0952% Test error with L1 penalty: 2.8571%

Sparsity with L1 penalty: 26.00% Sparsity with L1 penalty: 78.00%

Test error with L2 penalty: 2.3175% Test error with L2 penalty: 2.9524%

Sparsity with L2 penalty: 0.00% Sparsity with L2 penalty: 0.00%

C=0.10 C=0.01

Test error with L1 penalty: 4.3175% Test error with L1 penalty: 50.4762%

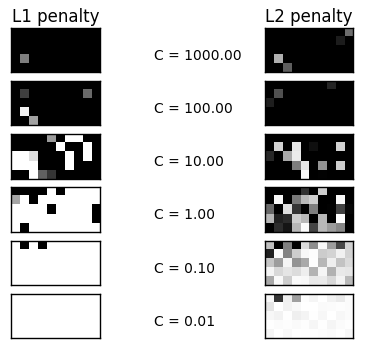
Sparsity with L1 penalty: 96.00% Sparsity with L1 penalty: 100.00%

Test error with L2 penalty: 3.5873% Test error with L2 penalty: 8.2222%

Sparsity with L2 penalty: 0.00% Sparsity with L2 penalty: 0.00%

From the result above, we can conclude that on the regard of test error, both regularization works well on low coefficients. However, L-1 penalty gets worse quickly as C grows while the accuracy of L-2 penalty is still acceptable. On the other hand, the sparsity of L-1 penalty gets worse when C is large while the sparsity of L-2 penalty is still low indicating a better accuracy.

We classify 5x10 images of digits into two classes: 0-4 against 5-9. The visualization shows coefficients of the models for varying C.



As shown in the figure above, as coefficients get smaller, the sparsity of the fitted hyperplane gets sparser. L-2 penalty works better than L-1 penalty. As we can see from the result obtained above, both two penalties could apply to different situation. When C is large, both regularization works fine and the accuracy of L-1 penalty is slightly better than the accuracy of L-2 penalty. When C get smaller, L-1 penalty is not acceptable in this situation.

# Multiclass Classification

## (j) Multiclass Classification with Naive Bayes Algorithm and SVM Classification

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

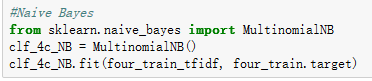
Multiclass classification means a classification task with more than two classes; e.g., classify a set of images of fruits which may be oranges, apples, or pears. Multiclass classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time.

**Naïve Bayes algorithm:**

Some classifiers perform the multiclass classification inherently. As such, Naïve 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



The Naïve Bayes multiclass classification could be implemented as follows:



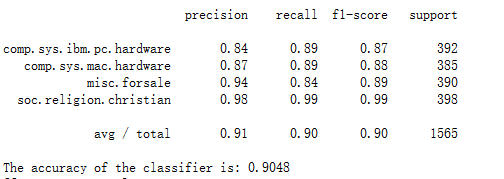
The confusion matrix of the classifier is shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Predicted 0 | | Predicted 1 | | Predicted 2 | | Predicted 3 | |
| Condition 0 | | 350 | | 31 | | 10 | | 1 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition 1 | 28 | 342 | 11 | 4 |
| Condition 2 | 37 | 21 | 329 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition 3 | 2 | 1 | 0 | 395 |

The precision, recall and accuracy of the classifier is:



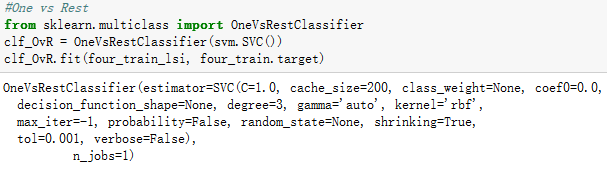
From the confusion matrix shown above, 0 stands for the comp.sys.ibm.pc.hardware, 1 stands for comp.sys.mac.hardware, and 2 stands for misc.forsale, 3 stands for soc.religion.christian. We can conclude from the measurement above that the accuracy of multiclass classification is slightly lower than binary classification. With an average accuracy of 0.90, the classifier works well in this problem, especially for the soc.religion.christian class, since the TPR is very high. The classifier works not so well on the other 3 classes which is reasonable, since comp.sys.ibm.pc.hardware and class comp.sys.ibm.pc.hardware are both classes about computer system hardware. Thus these two classes have similarity intrinsically, thus it’s acceptable to be misclassified from time to time.

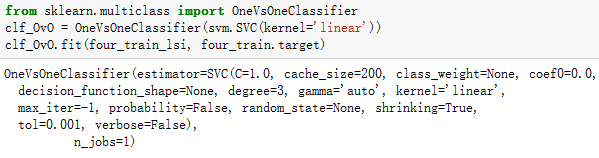
**Multiclass SVM Classification:**

OneVsRest strategy, also known as one-vs-all, is implemented in OneVsRestClassifier. The strategy consists in fitting one classifier per class. For each classifier, the class is fitted against all the other classes. In addition to its computational efficiency (only n\_classes classifiers are needed), one advantage of this approach is its interpretability. Since each class is represented by one and only one classifier, it is possible to gain knowledge about the class by inspecting its corresponding classifier. This is the most commonly used strategy and is a fair default choice.

OneVsOneClassifier constructs one classifier per pair of classes. At prediction time, the class which received the most votes is selected. In the event of a tie (among two classes with an equal number of votes), it selects the class with the highest aggregate classification confidence by summing over the pair-wise classification confidence levels computed by the underlying binary classifiers. Since it requires to fit n\_classes \* (n\_classes - 1) / 2 classifiers, this method is usually slower than one-vs-the-rest, due to its O(n\_classes^2) complexity. However, this method may be advantageous for algorithms such as kernel algorithms which don’t scale well with n\_samples. This is because each individual learning problem only involves a small subset of the data whereas, with one-vs-the-rest, the complete dataset is used n\_classes times.

The OneVsRest and OneVsOne classifier could be implemented respectively as follows:



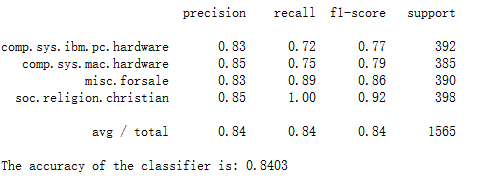


The confusion matrix and other evaluation of the **OneVsRest** classifier is shown below:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Predicted 0 | | Predicted 1 | | Predicted 2 | | Predicted 3 | |
| Condition 0 | | 282 | | 43 | | 40 | | 27 | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition 1 | 40 | 287 | 29 | 29 |

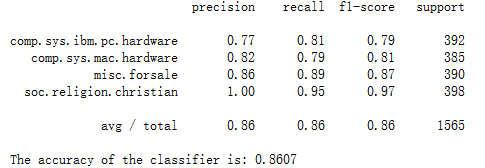
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Condition 2 | 19 | 8 | 349 | 14 |
| Condition 3 | 0 | 0 | 1 | 397 |



From the result above, we can conclude that the accuracy of the OneVsRest classifier is lower than Naive Bayes Classifier. However, the OneVsRest classifier works fine for soc.religion.christian category, since the recall of condition 3 is very high indicating that nearly all entries belong to condition 3 are correctly predicted.

The confusion matrix and other evaluation of the **OneVsOne** classifier is shown below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Predicted 0 | Predicted 1 | Predicted 2 | Predicted 3 |
| Condition 0 | 318 | 51 | 23 | 0 |
| Condition 1 | 56 | 305 | 24 | 0 |
| Condition 2 | 28 | 14 | 347 | 1 |
| Condition 3 | 9 | 2 | 10 | 377 |



The accuracy of the OneVsOne classifier is similar with OneVsRest classifier. The difference is that for OneVsRest classifier, the recall of condition 3, i.e. soc.religion.christian is very high, while in this case, the precision of condition 3 is high indicating that every entry predicted to belong to condition 3 is predicted correctly. To summarize the two SVM multiclass classification methods, it could be found that these algorithms may not be accurate enough for comp.sys.mac.hardware and comp.sys.ibm.pc.hardware and misc.forsale, but both work well for soc.religion.christian category. In general, the accuracy is lower than Naive Bayes multiclass classifier, but the accuracy is still acceptable for around 85%.

There’re certainly some advantages and drawbacks for these two methods. For OneVsRest method, it reduces the classifiers we use, but need to a cement score, which may be utilized for picking up the final outcome. For OneVsOne method, it may cost a large amount of classifiers, but each classifier only need to give a discrete outcome (the dataset is belonging to class A or B). What’s more, for OneVsRest method, we also need to pay attention to the unbalanced number of documents in each class. Thus method we tried in problem a) could be utilized here to solve the problem.