EE239AS Project 2

Classification Analysis

Winter 2016

Introduction:

Statistical classification refers to the task of identifying a category, from a predefined set, to which a data point belongs, given a training data set with known category memberships. Classification differs from the task of clustering, which concerns grouping data points with no predefined category memberships, where the objective is to seek inherent structures in data with respect to suitable measures. Classification turns out as an essential element of data analysis, especially when dealing with a large amount of data. In this project we look into different methods for classifying textual data.

Dataset and Problem Statement:

In this project we work with "20 Newsgroups" dataset. It is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups, each corresponding to a different topic.
The objective is to, given the collection of documents with predefined class types, train a classifier to group the documents into two classes: Computer Technology and Recreational activity. These two classes include the following sub-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

We highly recommend using python as your programming language. The easiest way to load the data is to use the built-in dataset loader for 20 newsgroups from scikit-learn package. As an example, if you want to load only "comp.graphics" category, then you can use the following command:

```
01. from sklearn.datasets import fetch_20newsgroups
02.
03. categories = ['comp.graphics']
04. graphics_train = fetch_20newsgroups(subset='train', categories=categories, shuffle=True, random_state=42)
```

Alternatively, you can fetch the dataset by running the python script located here and load the data manually. Note that documents belonging to each class are placed in a separate directory. The training and test sets are also separated for each category.

a) 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).

Modeling Text Data and Feature Extraction:

The primary step in classifying a corpus of text is document representation; this representation should be succinct as not to include too much irrelevant information, which leads to computational intractability and over fitting and at the same time capture essential features of the document. The dominant representation of a document used in the context of text analysis is to reduce a document to an unordered array of terms, which is based upon the so-called “Bag of Words” assumption. Essentially, a document is represented as a histogram of terms or a vector of some more sophisticated statistics of the terms in a vocabulary. As such, a corpus of text can be summarized into a term-document matrix whose entries are some statistic that represents the significance of a word in a document.

As a simple approach, one can consider a normalized count of the vocabulary words in each document as a representation vector. Another popular numerical statistic to capture the importance of a word to a document in a corpus is the Term Frequency-Inverse Document Frequency (TFxIDF) metric. This measure takes into account not only a suitably normalized count of the word in the document but also the frequency of the word in the whole corpus. To avoid unnecessarily large feature vectors (vocabulary size), terms that appear in almost every document, or are very rare are taken out of the vocabulary. The same goes with special characters, common stop words (e. g. and), and other appearances of words that share the same stem already in the vocabulary (e. g. goes, going).

b) Now, we turn the documents in the data set into numerical feature vectors. First tokenize each document and extract all the words that appear in your documents, excluding the stop words, punctuations, and different stems of a word. Then create the TFxIDF vector representations. Report the final number of terms you extracted.

As for a list of stop words, you can refer to the list provided in scikit-learn package. You can run the following commands to take a look at this list.

```
01. from sklearn.feature_extraction import text
02. stop_words = text.ENGLISH_STOP_WORDS
```

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

$$(0.5 + 0.5\frac{f_{t,c}}{\max_{t'} f_{t',c}}) \times \log\frac{|C|}{|c \in C : \ t \in c|}$$

c) Find the 10 most significant terms in the following classes, with respect to the above measure (Note that you evaluate the measure, where C is the collection of all 20 classes):

comp.sys.ibm.pc.hardware , comp.sys.mac.hardware, misc.forsale, and soc.religion.christian.

Feature Selection:

Even after these operations, the dimensionality of the representation vectors (TFxIDF vectors) ranges in the order of thousands; however, although lying in a high dimensional space, these vectors are sparse. High dimensionality of data diminishes the performance of many learning algorithms; hence the dimension of the space wherein your data lie should be reduced. This can be done via selecting a subset of the original features, which are more relevant with respect to certain performance measure, or through transforming the features into a lower dimensional space.

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.

The LSI representation is obtained by computing eigenvectors corresponding to the largest eigenvalues of the term-document matrix. LSI is very much similar to Principal Component Analysis (PCA), except in PCA the eigenvalue decomposition is applied to the covariance matrix of the data(Note that here the covariance matrix is referred to the document-term matrix multiplied by its transpose, whose entries represent co-occurring terms in the documents.)

Eigenvectors corresponding to the dominant eigenvalues obtained by LSI are now directions related to dominant combinations of terms occurring in the corpus, which are referred to as “topics” or “semantic concepts”. Thus, the new low dimensional representations are the magnitudes of the projections of the documents into these latent topics.

Let D denote the $t \times d$ term-document matrix with rank r where each of the d columns represents a document vector of dimension t. The singular value decomposition results in

$$D = U\Sigma V^T,$$

Where Σ is an $r \times r$ diagonal matrix of singular values of D, U is a $t \times r$ matrix of left singular column vectors, and V is a $d \times r$ matrix of right singular vectors. Dropping all but k largest singular values and corresponding singular vectors gives the following truncated approximation

$$D_k = U_k \Sigma_k V_k^T.$$

The approximation is the best rank k approximation of D in the sense of minimizing the sum of squared differences between the entries of D and D_k.

The $t \times k$ matrix U_k can now be used as a projection matrix to map each $t \times 1$ document column vector $\vec{d}$ into a k-dimensional representation

$$\overrightarrow{d_k} = U_k^T \vec{d}.$$

d) Apply LSI to the TFxIDF matrix and pick k=50; so each document is mapped to a 50-dimensional vector. **Use the selected features in your learning algorithms**.

Learning Algorithms:

e) 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 and computational complexity. Depending on the package you use, an optimization problem is solved to learn the vector of feature weights, $\widehat{w}$, and the intercept, b, given the training data set. Once the weights are learned, new data points are classified by computing $W^T.\vec{x} + b$, with $\vec{x}$, being the vector representation of the new document to classify.

Basically by considering the sign of the evaluation equation $sign(W^T.\vec{x} + b)$, in other words thresholding the equation with 0, one can determine the label of the new data points. Classification accuracy is measured using the average of precision, recall and accuracy. Precision is the proportion of items placed in the category that are really in the category, and Recall is the proportion of items in the category that are actually placed in the category.

Depending on the application of interest, 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. The curve is created by plotting the true positive rate against the false positive rate at various threshold settings. In order to obtain the other points in the ROC curve you can change the threshold you use for the evaluation equation.

Use SVM method to separate the documents into Computer Technology vs Recreational Activity groups. In your submission, plot the ROC curve, report the confusion matrix and calculate the accuracy, recall and precision of your classifier.

f) An alternative approach is to use a soft margin SVM. That is, we solve the following optimization problem

$$\min \frac{1}{2} \| w \|_2^2 + \gamma \sum_{i=1}^{n} \xi_i$$

Under the constraints $y_i\left(w^T . \vec{x}_i + b\right) \geq 1 - \xi_i$, and $\xi_i \geq 0$, for all $i \in \{1, \dots, n\}$.

Note that in the objective function, each slack variable shows the amount of error that the classifier makes on a given example. Minimizing the sum of the slack variables corresponds to minimizing the loss function on the training data, while minimizing the first term corresponds to maximizing the margin between the two classes. The tradeoff parameter γ determines how much important each component is. Note that in this case, $\gamma = 0$ corresponds to the hard margin SVM.
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 $\{10^{-k} | -3 \leq k \leq 3, k \in Z\}$. Report the confusion matrix and calculate the accuracy, recall and precision of your classifier.

g) Next, we use naïve Bayes algorithm for the same classification task. The algorithm estimates the maximum likelihood probability of a class given a document with feature set $\vec{x}$, using Bayes rule, based upon the assumption that given the class, the features are statistically independent.
Train a multinomial naïve Bayes classifier and plot the ROC curve for different values of the threshold on class probabilities. You should report your ROC curve along with that of the other algorithms. Again, Report the confusion matrix and calculate the accuracy, recall and precision of your classifier.

h) Repeat the same task with the logistic regression classifier, and plot the ROC curve for different values of the threshold on class probabilities. You should report your ROC curve along with that of the other algorithms. Provide the same evaluation measures on this classifier.

Multiclass 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.

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

$$\hat{c} = \arg\min_{c \in C} p(c|\overrightarrow{x}).$$

For SVM, however, one needs to extend the binary classification techniques when there are multiple classes. A natural way to do so is to perform a one versus one classification on all $\binom{|C|}{2}$ pairs of classes, and given a document the class is assigned with the majority vote. In case there is more than one class with the highest vote, the class with the highest total classification confidence levels in the binary classifiers is picked.
An alternative strategy would be to fit one classifier per class, which reduces the number of classifiers to be learnt to $|C|$. For each classifier, the class is fitted against all the other classes. Note that in this case, the unbalanced number of documents in each class should be handled. By learning a single classifier for each class, one can get insights on the interpretation of the classes based on the features.

i) In this part, we aim to learn classifiers on the documents belonging to the classes mentioned in part b; namely

comp.sys.ibm.pc.hardware , comp.sys.mac.hardware, misc.forsale, and soc.religion.christian.

Perform Naïve 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 and precision of your classifiers.

Submission: Please submit a zip file containing your report, and your codes with a readme file on how to run your code to ee239as.winter2016@gmail.com. The zip file should be named as "Project2_UID1_UID2_..._UIDn.zip" where UIDx are student ID numbers of the team members. If you had any questions you can send an email to the same address.