Information Retrieval: Text Mining

The primary purpose of this assignment is to get familiar with some text mining algorithms: feature extraction, feature selection, classification, and clustering.

Again, you can work individually or in a two-person team.

Setup:

- Python is recommended and you will use the functions provided by scikit-learn for most of the tasks.
- Download the mini 20 newsgroups dataset. The documents are categorized into 20 directories, corresponding to the 20 news groups. Note: you are encouraged to check the related literatures to find the performance of various classification and clustering methods on this dataset.
- For plotting curves, you may consider *matplotlib*, as shown in this example. More examples can be found on the sklearn or other websites.

Part 1: Extracting features

You can use the inverted indexing code you implemented in the first project, or revise some code from the web such as https://nlpforhackers.io/building-a-simple-inverted-index-using-nltk/ for generating the features. Please implement a script feature-extract.py, so that the following command will generate the required training datasets for classification and clustering.

```
python feature-extract.py directory_of_newsgroups_data feature_definition_file class_defini
```

Specifically, the command will generate the following files

- feature_definition_file contains the list of terms used as features. Each line contains a pair (feature id, term) sorted by index. The feature id is used in the training_data file.
- class_definition_file contains of the class mapping. The 20 newsgroups are grouped into 6 classes: (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). Each row in the class definition file is a mapping (newsgroup, class_label). You can hard-code the mappings in your program and output them.

- A training_data_file is in the libsvm format, i.e., the sparse representation of the document-term matrix. Each row represents one document and takes the form `< class label> <feature-id>:<feature-value> <feature-id>: <feature-value> ...` This format is especially suitable for sparse datasets - all the zero-value features are skipped. According to the newsgroup directory that the document belongs to, you can find the class label from the mapping defined in the class_definition_file. Each feature-id refers to the term defined in feature_definition_file. Each feature value is the corresponding TFIDF value (or TF, or IDF for different types of training data) of the term in the document. When you parse the documents, you only need to look at the subject and body. The subject line is indicated by the keyword "Subject:", while the number of lines in the body is indicated by "Lines: xx", which are

the last xx lines of the document. Some files may miss the "Lines: xx" or have other exceptions. Please manually add the "Lines: xx" line or appropriately handle the exceptions. You will generate three training data files: e.g., training_data_file.TF, training_data_file.IDF, and training_data_file.TFIDF. Please carefully test the feature values you generated. Your experiments on classification and clustering may select different training (for example, training Bernoulli Naive Bayes classifier may need IDF information).

Once you have the training data in the libsvm format, you can easily load it into sklearn as follows.

```
from sklearn.datasets import load_svmlight_file
feature_vectors, targets = load_svmlight_file("/path/to/train_dataset.txt")
```

Data pre-processing has been one of the most time-consuming and critical steps in data mining. Make sure you thoroughly test your code.

Part 2: Classification

We will experiment with several algorithms mentioned in the class: Multinominal Naive Bayes, Bernoulli Naive Bayes, k Nearest Neighbor, and SVM. The corresponding usages are as follows.

- The usage of Multinominal NB:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
```
- The usage of Bernoulli NB:
```
from sklearn.naive_bayes import BernoulliNB
clf = BernoulliNB()
```
- The usage of kNN:
```
from sklearn.neighbors  import KNeighborsClassifier
clf = KNeighborsClassifier()
```
- The usage of SVM:
```
from sklearn.svm  import SVC
clf = SVC()
```

You can check the corresponding links to the sklearn documentation pages for the parameters. To simplify the task, just use the default parameters, i.e., no parameter setting for the classifiers.

Classifier Evaluation. For more realistic evaluations, you will need to partition the data by date and train with older data and test on newer data (i.e., the heldout data). This project does not require you to follow this procedure. However, we still want to use cross validation for the whole dataset to get more reliable estimation of the classifier performance - you can get both the mean and standard deviation of a selected metric. The following code snippet shows how to use cross validation in evaluation.

```
from sklearn.model_selection import cross_val_score
scores = cross_val_score(clf, feature_vectors, targets, cv=5, scoring='f1_macro')
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

This example uses 5-fold cross-validation (cv=5) and the metric is f1_macro, which is the macro-averaging of the F1 scores for different classes (check the lecture slides for the detail).

For each classifier, please report the mean and 2*std of 5-fold f1_macro, precision_macro, and recall_macro, respectively. Hint: a reasonble classifier

typically has F1 score in the range >0.5. Discuss what you have observed. Put the related code in "classification.py".

Part 3: Feature Selection

We will evaluate the two feature selection methods discussed in the class: the chi-squared method and the mutual information method, to find out how they performs for this dataset. The following example shows how to use them in sklearn.

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2, mutual_info_classif
X = feature_vectors
y = targets
X_new1 = SelectKBest(chi2, k=100).fit_transform(X, y)
X_new2 = SelectKBest(mutual_info_classif, k=100).fit_transform(X, y)
```

The selection of K depends on the actual number of features available. For this dataset, it can be in the range from a few hundreds to a few thousands. Please test with both methods for a number of Ks and the four classifiers to find out how feature selection performs for this dataset and report with figures of (x-axis: K, y-axis:f1_macro). Discuss what you have observed. Put the code in "feature_selection.py"

Part 4: Document Clustering

We will experiment with two clustering algorithms: kMeans and hierachical clustering. Suggestion: using the Part-3 feature selection methods, you can get a smaller feature set for faster and more appropriate evalution in this experiment. The following example shows the usages of kMeans and hierarchical clustering.

```
from sklearn.cluster import KMeans, AgglomerativeClustering
kmeans_model = KMeans(n_clusters=20).fit(feature_vectors)
single_linkage_model = AgglomerativeClustering(
        n_clusters=20, linkage='ward').fit(feature_vectors)
```

The hierarchical clustering algorithm uses the agglomerative style (botton-up) and one of the three linkage methods (ward, complete, average) for computing cluster distances.

Try a range of n_clusters (the number of clusters) in [2, 25] for each clustering algorithm and evaluate each one's clustering quality with the Sihouette Coefficient (SC) and Normalized Mutual Information (NMI) measures. SC does not depend on the existing class labels, while NMI does. The following example shows the usage, where "classification_labels" are the "targets" used in the classification modeling tasks.

```
from sklearn.cluster import KMeans
from sklearn import metrics
kmeans_model = KMeans(n_clusters=3).fit(X)
clustering_labels = kmeans_model.labels_
metrics.silhouette_score(X, clustering_labels, metric='euclidean')
metrics.normalized_mutual_info_score(classification_labels, clustering_labels)
```

Plot the figures of (x-axis: K, y-axis: the measure) for both quality measures, correspondingly. Each figure contains the curves for the two clustering methods. Discuss what you have observed. Put the code in a script file "clustering.py".

Deliverables

Turn in three files (DO NOT zip them into one zip file) to Pilot: (1) a summary in PDF file about your your implementation, experimental results, and discussions for each part of the project, (2) paste your well-commented source code in one PDF file, and (3) the zipped whole source code directory including the generated libsvm format data (but please exclude the raw newsgroup data!). You should put your team members names in the beginning of each PDF file.

Plagiarism will be penalized with an award of 0 and further action. No deadline extension will be given.

This page, first created: March, 2018, modified March, 2020