COMP10200: The Reuters-21578 Corpus

1. Introducing the Corpus

The Reuters-21578 corpus is a set of 21578 news stories from the Reuters newswire that were collected in 1987. Since its release in 1990, it has become the most widely used corpus for research in text classification.

Documents in the Reuters-21578 corpus are marked up in SGML (Standardized General Markup Language).

Here's an example:

```
<REUTERS TOPICS="YES" LEWISSPLIT="TRAIN" CGISPLIT="TRAINING-SET"
OLDID="5544" NEWID="1">
   <DATE>26-FEB-1987 15:01:01.79</DATE>
   <TOPICS><D>cocoa</D></TOPICS>
   <PLACES><D>el-salvador</D><D>usa</D><D>uruguay</D></PLACES>
   <PEOPLE></PEOPLE>
   <ORGS></ORGS>
   <EXCHANGES></EXCHANGES>
   <TEXT>2;
      <TITLE>BAHIA COCOA REVIEW</TITLE>
      <DATELINE>    SALVADOR, Feb 26 - </DATELINE>
      <BODY>
            Showers continued throughout the week in the Bahia cocoa
            zone, alleviating the drought since early January and
            improving prospects for the coming temporao, although
            normal humidity levels have not been restored, Comissaria
            Smith said in its weekly review...
      </BODY>
   </TEXT>
</REUTERS>
```

Each document can be seen as a collection of `<TEXT>` (including `<TITLE>`, `<DATELINE>`, and `<BODY>`) with labels attached for classification. These labels include `<TOPICS>`, `<PLACES>`, `<PEOPLE>`, `<ORGS>`, and `<EXCHANGES>`.

Any document can contain any number of labels (i.e. `<topics>` could include "cocoa", "earn", and "acq"). So there are actually multiple classification tasks, each with two categories:

"cocoa" vs. "not cocoa"

"earn" vs. "not earn"

"acq" vs. "not acq"

Etc.

2. Parsing the Corpus

The example code folder for this week contains the Reuters-21578 corpus in a subfolder called "Reuters21578". This is a slightly cleaned up version of the corpus that is designed to be parsed using the Python `xml` package. The cleanup involved removing some non-text characters and adding a `<CORPUS>` tag to each of the ".sgm" files so that it became legal XML.

The parse_reuters.py file contains some code that will read the corpus into a list of 21578 dict objects.

```
>>> docs = read_reuters()
```

Optional parameters include the path to the Reuters21578 folder and a limit on the number of documents to be read.

Here is an example of the first document after the call to `read_reuters()` above.

```
docs[0] {
    date: '26-FEB-1987 15:01:01.79',
    topics: ['cocoa'],
    places: ['el-salvador', 'usa', 'uruguay'],
    people: [],
    orgs: [],
    exchanges: [],
    title: 'BAHIA COCOA REVIEW',
    dateline: ' SALVADOR, Feb 26 - '
    body: 'Showers continued throughout the week in\nthe Bahia
    cocoa zone, alleviating the drought since early\nJanuary
    and improving prospects for the coming temporao,\nalthough
    normal humidity levels have not been restored,\nComissaria
    Smith said in its weekly review...',
}
```

There is also a function that returns a list of the class labels of a given type. The labels are listed along with their frequencies (i.e. the number of documents they appear in), sorted in descending order of frequency.

```
>>> get_labels(docs, "places")

[('usa', 12542), ('uk', 1489), ('japan', 1138), ('canada', 1104),
('west-germany', 567), ('france', 469), ('brazil', 332),
('australia', 270), ('china', 223), ('ussr', 216), ('belgium',
214), ('switzerland', 214), ('netherlands', 196), ('iran', 179),
…
```

This code could be combined with the Naïve Bayes code from last week to produce a classifier for each label.

3. Classifying the Corpus

Here's a recipe for using the Naïve Bayes code from week 5 to build classifiers for the Reuters-21578 corpus:

1. Decide which label type you are interested in (places, topics, etc) and then choose on a target label for your classifier.
2. Use `read_reuters` to read the corpus and store it in a variable.
3. Loop through the documents in the collection to get two lists: one containing the lists of words from each document (call `textParse` for this), and one containing the labels (convert these to 1 if the document has the label and 0 if it does not).
4. Create the vocabulary by calling `createVocabList`. Store the result in a variable.
5. Create the word vectors using `bagOfWords`.
6. Separate the word vectors and labels into two sets – one for training and another for testing
7. Create and train a `MultinomialNB` object.
8. Test its predictions.
9. Report the results.

4. Tips for code development:

Start with a subset of the Reuters documents to speed things up during development (e.g. use `limit=1000` when calling `read_reuters`).

Report statistics and results as the code runs (e.g. number of docs, number of positive examples, vocabulary size, training and testing set size, top 10 features for the positive and negative classes, etc.)

Use print statements to track the progress of your code (e.g. "reading data", "training data", "testing data", etc.).

In a long loop, output progress messages (e.g. report every 100 docs added to vocabulary list).

To speed things up, try starting with just the `<title>` text. Then add the `<body>` and `<dateline>` text later.