Week 2 lab

Designing a classification scheme

One of the most common tasks in language processing is *text categorisation* (also called *text classification*). In rare cases we are given well-established pre-defined categories into which our statistical system should learn to classify the texts. Often the language processing expert is (or should be) involved in designing a *classification scheme* (or *schema*).

We can **group texts in different ways** depending on the application we intend it for:

- by theme (e.g. business vs entertainment)
- by genre (e.g. news vs. literature vs. legal contract)
- by ontological properties of the subject (e.g. it describes a plant vs. an animal?)
- by the attitudes expressed (e.g. positive vs. negative vs. objective)
- by how readers tend to respond (e.g. exciting vs. scary)
- by quality (e.g. well written/argued vs. poorly)
- …

In all cases, we must decide on the broad **nature of the scheme**:

- should the categories be broad (coarse-grained), or specific (fine-grained)?
- do we select one category per item (*multiclass classification*) or allow multiple categories per item (*multilabel classification*) or multiple categorical features of each item (*multitarget/multifaceted classification*)?
- should the categories be flat, or hierarchical?
- do we have an “other” category?
- is there an existing scheme we can/should reuse?

We have a few different **driving factors**:

- what level of discrimination between objects is useful in target *applications*?
- what features of a scheme make it *feasible* for someone to label each document with a category? (This labelling is often called *annotation* in the NLP community, but *coding* in the social sciences. *Annotation* may also be used to describe automatic labelling and labelling of parts of documents rather than entire documents.)
- what schemes are possible to learn and predict with an automatic text classification model?

As a class we will discuss the design of a category scheme for the corpus collected for manual annotation in Assignment 1 and automatic classification in Assignment 2. The next couple of labs will consider aspects of the assignment.

Scenario

The data we're looking at comes from the **Sharing News Online (http://sydney.edu.au/arts/slam/research/projects/sharing_news_online.shtml)** project, which tried to look at why people shared online news media on Facebook and Twitter. Here we have a sample of articles that were shared over a few months from various news web sites. The researchers want to know: what topics of article get shared? Do these change over place? Over time? Does topic help differentiate between "been shared" and "went viral"?

A note on annotation scheme refinement

Away from the artificial context of the assignment, it is common practice for annotators to iterate over a cycle of:

1. independent annotation;
2. discussion and adjudication; and
3. scheme refinement.

This may be repeated until a pre-determined level of agreement is reached, or until a budget is exhausted, before a larger set of documents is labelled.

Ordinarily the independence of annotations is very important to estimate the true difficulty of the task, although we violate that here for the sake of teaching.

Take away

- We often want to categorise texts into a small number of categories.
- We have to define a scheme that is useful for some task, and allows annotators and predictive models to make consistent decisions (within reason).
- There are some hard choices in deciding on the set of categories. These are influenced by task-related, human and statistical/engineering factors.
- Assigning individual items to categories will be subject to ambiguity and disagreement.

Please start taking notes on what you might want to discuss in the assignment about the challenges of developing a classification task.

Homework exercises with regular expressions

While you don't have much to work on for the assignment yet, regular expressions are a useful skill to develop.

1. Using regular expressions is a skill that can be strengthened with practice exposure. Relevant and fun exercises include:
    - **Regex Golf game (https://alf.nu/RegexGolf)**
    - **SLP3 on regular expressions (https://web.stanford.edu/~jurafsky/slp3/2.pdf)** (exercises at the end before references)
    - **Regular Expressions for NLP by Steven Bird and Ewan Klein (http://courses.ischool.berkeley.edu/i256/f06/papers/regexps_tutorial.pdf)**

2. Familiarise yourself with how to do the following in your choice of programming language:
    - Find matches for some pattern within a string. For example: extract all words starting with (case-insensitive "jo").
    - Replace matches for some pattern with a fixed string. For example: replace all words starting with case-insensitive "jo" with "Joe". Be careful not to include any punctuation following the jo- word.
    - Replace matches for some pattern with a string that is determined from the match. For example: make all words starting with "jo" uppercase. I.e. "joey" would become "JOEY".

3. On tokenisation: Manning and Schütze exercise 4.1: "for most purposes, we'd want to treat some hyphenated things as words (for instance *co-worker*, *Asian-American*), but not others (for instance, *ain't-it-great-to-be-a-Texan*, *child-as-required-yuppie-possession*). Find hyphenated forms in a corpus (e.g. the **Brown Corpus (http://www.nltk.org/nltk_data/)**) and suggest some basis for which forms we would want to treat as words and which we would not. What are the reasons for your decision?
    - Python users might use this snippet to get hyphenated terms from the Brown Corpus in a trigram of context:

    ```
    [(p, w, n) for p, w, n
     in nltk.ngrams(nltk.corpus.brown.words(), 3)
     if len(w) > 2 and '-' in w]
    ```

    (You may need to use `nltk.download('brown')` to get the Brown Corpus first. You may also need to configure the `NLTK_DATA` (https://www.nltk.org/data.html) environment variable if you cannot write to the default path.)

4. Extension: "Truecasing". Take a text and transform it to be all lowercase. Build a system (e.g. a set of regular expression searches) to identify which characters were uppercase in the original document.
    - Find the errors in recovering true case. How many are there relative to the total number of letters?
    - What kinds of errors does your initial system make?

- Can you modify your regular expressions to reduce the number of errors? Measure your error relative to the total number of characters.
- What kinds of capitalisation information is hard to capture with regular expressions?
- Does that improvement persist if you choose a different text to test on?