School of Computing and Information Systems
The University of Melbourne
COMP90042
WEB SEARCH AND TEXT ANALYSIS (Semester 1, 2017)

Sample solutions for discussion exercises: Week 2

Discussion

1. Give some examples of text processing applications that you use on a daily basis.

    - There are lots! For example, Google (or other web search engines), Siri (or other speech-to-text systems), predictive messaging, spelling correction, machine translation, and so on.

2. What is **tokenisation** and why is it important?

    - Tokenisation is the act of transforming a (long) document into a set of meaningful substrings, so that we can compare with other (long) documents.
    - In general, a document is too long — and contains too much information — to manipulate directly. There are some counter-examples, like **language identification**, which we need to perform before we decide how to tokenise anyway.

    (a) What are **stemming** and **lemmatisation**, and how are they different? Give examples from the `preprocessing` iPython notebook.

    - Both stemming and lemmatisation are mechanisms for transforming a token into a canonical (base, normalised) form. For example, turning the token *walking* into its base form *walk*.
    - Both operate by applying a series of rewrite operations to remove or replace (parts of) affixes (primarily suffixes). (In English, anyway.)
    - However, lemmatisation works in conjunction with a **lexicon**: a list of valid words in the language. The goal is to turn the input token into an element of this list (a valid word) using the rewrite rules. If the re-write rules can't be used to transform the token into a valid word, then the token is left alone. (For example, the token *lemming* wouldn't be transformed into *lemm* because the latter isn't in the word list.)
    - Stemming simply applies the rewrite rules, even if the output is a garbage token (like *lemm*).
    - One further idea is the difference between **inflectional morphology** and **derivational morphology**:
        - Inflectional morphology is the systematic process (in many but not all languages) by which tokens are altered to conform to certain grammatical constraints: for example, if the English noun *teacher* is plural, then it must be represented as *teachers*. The idea is that these changes don't really alter the meaning of the term. Consequently, both stemming and lemmatisation attempt to remove this kind of morphology.

    - Derivational morphology is the (semi-)systematic process by which we transform terms of one **class** into a different class (more on this next week). For example, if we would like to make the English verb *teach* into a noun (someone who performs the action of *teaching*), then it must be represented as *teacher*. This kind of morphology tends to produce terms that differ (perhaps subtly) in meaning, and the two separate forms are usually **both** listed in the lexicon. Consequently, lemmatisation doesn't usually remove derivational morphology in its normalisation process, but stemming usually does.

  - Another example, from the notebook, is the token *this*. Using the lemmatiser, the token remains unchanged, because it is already listed in the lexicon. The stemmer, however, strips the *-s* suffix, so that we end up with *thi*.

3. What is **text classification**? Give some examples.

   - Numerous examples from the lectures: genre identification, author identification, etc.

   (a) Why is text classification generally a difficult problem? What are some hurdles that need to be overcome?

   - The main issue is in terms of **document representation** — how do we identify **features** of the document which help us to distinguish between the various classes?
   - The principal source of features is based upon the presence of tokens (words) in the document (known as a **bag–of–words** model). However, many words don't tell you anything about the classes we want to predict, hence **feature selection** is often important. On the other hand, single words are often inadequate at modelling the meaningful information in the document, but multi-word features (e.g. bi-grams, tri-grams) suffer from a **sparse data problem**.

   (b) Consider some (supervised) text classification problem, and discuss whether the following (supervised) machine learning models would be suitable:

   - The answers will vary depending on the nature of the problem, the feature set, the class set, and so on. One possible solution, for a generic genre identification problem using an entire bag–of–words model (similar to the notebook) is as follows:

   i. k-Nearest Neighbour using Euclidean distance

   - Often this is a bad idea, because Euclidean distance tends to classify documents based upon their **length** — which is usually not a distinguishing characteristic for classification problems.

   ii. k-Nearest Neighbour using Cosine similarity

   - Usually better than the previous, because we're looking at the distribution of terms. However, k-NN suffers from high-dimensionality problems, which means that our feature set based upon the presence of (all) words usually isn't suitable for this model.

   iii. Decision Trees using Information Gain

    - Decision Trees can be useful for finding meaningful features, however, the feature set is very large, and we might find **spurious** correlations. More fundamentally, Information Gain is a poor choice because it tends to prefer **rare** features; in this case, this would correspond to features that appear only in a handful of documents.

  iv. Naive Bayes

    - At first glance, a poor choice because the assumption of the **conditional independence** of features and classes is highly untrue.
    - Also sensitive to a large feature set, in that we are multiplying together many (small) probabilities, which leads to biased interpretations based upon otherwise uninformative features.
    - Surprisingly somewhat useful anyway!

  v. Logistic Regression

    - Useful, because it relaxes the conditional independence requirement of Naive Bayes.
    - Since it has an implicit feature weighting step, can handle large numbers of mostly useless features, as we have in this problem.

  vi. Support Vector Machines

    - Linear kernels often quite effective at modelling some combination of features that are useful (together) for characterising the classes.
    - Need substantial re-framing for problems with multiple classes (instead designed for two-class (binary) problems); most text classification tends to be multi–class.

(c) In the `text_classification` iPython notebook, which machine learning model(s) works best on the given classification problem based on the Reuters corpus? Why do you think that is?

  - Logistic regression and SVMs appear to be the strongest performers, agreeing with our expectations.
  - For SVMs, however, note that the problem is framed purely in terms of the most frequent class (`acq`) vs. everything else — which is more suited for an SVM. If we were attempted to classify all 90 (!) classes, it probably wouldn't work quite so well.