Student's name: Balazs Varhegyi **Date**: 02.01.2015

Homework exam task

Implementing language identification

Task description

For many automated text analytics task (e.g. machine translation) it is crucial to know the language of the source text. Identifying the language is in most cases trivial for humans, but can be challenging to perform automatically. The simplest algorithms consider only the frequencies of letters and their combinations (aka. n-grams). Implement such an algorithm and evaluate it on one of the benchmark datasets.

Used techonolgy

The program was implemented in Java using Eclipse a well-known Java IDE. Maven was used for dependency management and the only dependency was Jsoup, a library I used only to remove html tags from text files.

Used dataset

The dataset[1] used for language identification described in:

Baldwin, Timothy and Marco Lui (2010) Language Identification: The Long and the Short of the Matter. In Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics. Los Angeles, USA, pp. 229-237.

It contains three sub-corpora:

- EuroGOV: derived from the WebCLEF EuroGOV document collection
- Wikipedia: sourced from wikipedia.org
- TCL: originally constructed by the Thai Computational Linguistics Laboratory

I used only EuroGov: 1500 files 10 languages and 26,2Mb data and Wikipedia ~5000 files, 75 languages and 7,3 MB for language identification.

Algorithm

I used the n-gram classification approach. Where n-gram is a substring containing N consecutive characters (e.g.: „th", 3-gram: „the"). As the first step of the process I created language profiles for each language that holds the most frequent n-grams of the texts in sorted order. Then for the classification part I started to read in the documents and predict their language based on distance from the profile. For calculating the difference

1 datasource: http://www.csse.unimelb.edu.au/research/lt/resources/naacl2010-langid/

between documents and profiles I used the following approach. If n-gram exists in a document and in a profile then distance is calculated like:

Document ngram list	Profile ngram list	Distance
asd	asd	0
abc	ad	1
ad	abc	1
dfg	not-exists	max

Packages, classes, methods

The whole project is available on github: https://github.com/varh1i/language_detection

In hu.bme.language_detection there's only the Main class that has two functions to analyse and classify the two datasets. It also contains logs to check how long it takes for the algorithm to run.

```
Main

    analyseAndPredictEuroGov

    analyseAndPredictWikipedia
```

The hu.bme.language_detection.model package contains the model object used in the program. Document has a constructor that accepts a text as an argument and gets the most frequent terms and creates a hashtable to store the positions of each n-gram. Language has a constructor that does the same as Language but it has an id field that is the iso639-1 language code. Ngram has two fields: text and frequency to store the frequency of each n-gram.

```
Document

    Map<String, Integer> positions

Language

    String id

    Map<String, Integer> positions

Ngram

    String text;

    Integer frequency
```

The hu.bme.language_detection.util is the package where interesting thing happens. It has the Profiler class that reads in the files and gets the most frequent n-grams from the input text file and by putting them into a list sorted by the occurence of the n-grams. After we have the most frequent terms in a sorted list. Profiler creates a hashtable where keys are the n-grams and values are the index in the sorted list. So when distance calculated between document and language profile we have O(n) complexity. I tried using ArrayList and LinkedList but when positions table is created getting values from LinkedList yields extra time resulting in around 10 times slower analysis for Wikipedia dataset. LanguagePredictor class only goes through the input files and compare the distance described above previously. ReaderUtil is the class used for reading in metadata of EuroGov dataset. I didn't have to read in metadata of Wikipedia because the filename already contained language id.

```
LanguageAndDocumentProfiler

    HashMap<String, Language> languages

    createLanguageProfilesEurGov(Map<String, List<String>>)

    createLanguageProfilesWikipedia()

    getLanguageIdFromFileName(File)

    createLanguage(String, String)

    getMostFrequentTerms(int, int, String)

    getFrequencyTable(int, String)

    createPositions(List<NGram>)

LanguagePredictor

    predictEuroGov(Map<String, Language>, Map<String,
    List<String>>)

    predictWikipedia(Map<String, Language>)

    getDistance(Language, Document)

ReaderUtil

    getFilesByLangFromEurGovMetaFile()

    readFile(File)
```

Performance Analysis

- Creating most frequent terms: in case of a 3-gram O(3n) and putting them into a sorted list in worst case: O(k * n) where k is the number of 1-grams, bi-grams, trigrams
- Creating position table from list O(n)
- Calculating distance between document and profile: O(n)

Results

We can see the ouput of the program below that summarize the results of the classification. The classification algorithm worked well on both dataset but worked better for the EuroGov dataset. The used parameters were:

- longest n-gram: 3
- number of most frequent n-grams: 400

```
Calculation started for EuroGov dataset...

Created profile for 10 languages in: 8527 ms.

correct: 98.73333333333333% (1481/1500)

Time for making predictions: 8589 ms

Calculation started for Wikipedia dataset...

Created profile for 75 languages in: 2280 ms.

correct: 88.01128349788434% (4368/4963)

Time for making predictions: 8140 ms
```