Detecting the Language

This is a brief document explaining my approach, along with my code, to identify the language of an inputted text which was the second task of this round of application process.

`detectMyLanguage.java`

The main() function initially asks the user :

```
Do you wish to train your model first (y/n)?
```

On entering 'y', the programs processes a huge corpora of sentences in different languages in order to ***train*** the model. This training algorithm is briefly explained afterwards. Just know that the output of this training process is a collection of files, corresponding to each language, in which each file contains unique **tri-grams(preferred)** or **4-grams** along with their probability of occurances in that language corpus.

On entering 'n', the program asks the user to enter a text in any language. The text size can be upto 100 words. And then, the program returns the detected language. The algorithm to detect the language from the **trained set** is also mentioned afterwards.

Training the model:

The corpora used for training is the Europarl corpus consisting of parallel texts in different languages of the Europian Union. It can be found at - **http://www.statmt.org/europarl/**

The languages that I used are – Bulgarian, Czech, Danish, English, French, German, Greek, Italian, Polish, Spanish.

In total, these languages contain combined text of 2.6 GB.

Pre-requisites for training :

1. The corpus files should be kept in a folder named – 'Training Set' on the desktop of your system.
2. In `detectMyLanguage.java`, change the value of the String variable – 'desktopPath' to whatever the path to your desktop is. Don't forget the '/' at the end.
3. In the same file, change the value of the variable 'n' to 3 to build trigrams(or 4 to build 4-grams).
4. The names of the corpus files must be the same as the language with the first letter in upper case(refer the values in the 'languages' array)
5. On desktop there should be a folder 'Test Set 3 grams' (or 'Test Set 4 grams').

Steps followed by the program to train:

1. A file is opened (as mentioned in the 'languages' array).
2. For each file, do the following:
   - Read a line from the file.
   - Create tri-grams from that sentence by using the object of NgramBuilder.java as explained in the following steps.
   - Firstly, filter the line by removing the unwanted punctuation, multiple spaces and then split the line into words.

- For each word, create tri-grams. Eg – for the word 'sentence', the tri-grams would be – sen, ent, nte, ten, enc and nce. For the word 'is' – the tri-gram would be – 'is'.
- These trigrams are stored in the Hashmap with tri-gram as key and its count as the value. The logic for this is pretty simple and can be understood by going through the function 'createNgrams' of NgramBuilder.java. HashMap is chosen as it is efficient to look for a value if a key is known.

3. Once the entire file is read, the tri-grams are stored into the folder 'Test Set 3 grams'. The file being written contains the tri-grams generated along with their probability of occurance in that language. This probability is calculated as - (the number of occurances of a tri-gram / total number of tri-grams).

The folder 'Test Set 3 grams' has a size of 3.5 MB.

I have focused mainly on tri-grams because in my experience the result of language detection was better using tri-grams than with 4-grams.

Identifying the language of the inputted text:

Pre-requisites for detecting the language :

1. The test set files should be kept in a folder named – 'Test Set 3 grams' on the desktop.
2. In `detectMyLanguage.java`, change the value of the String variable – 'desktopPath' to whatever the path to your desktop is. Don't forget the '/' at the end.
3. Change the value of the variable 'n' to 3.

The main() function initially asks the user :

```
Do you wish to train your model first (y/n)?
```

As the user enters 'n', the program reads the test set files and stores the tri-grams and their probability values for each language in a hashmap with the tri-gram as key and its count as its value. This hashmap is stored as a value in another hashmap whose key is the language for which the tri-grams were read. Again, the code is pretty much self-explainatory.

Once the outer hashmap has been created, the inputted text is broken down to form tri-grams in a similar way as done while training the model.

Now each tri-gram is looked into the hashmap against each language. If found for a particular language, its probability value is multiplied with the previous probability value for that language. After all the tri-grams have been parsed, the language with the highest product of probabilities is chosen as the detected language.

I have tried to incorporate the Naive Bayesian model while calculating the probability. But I formalized it a little bit according to my needs, like I ignored the probabilities of language (assumed to be equally likely) and of the inputted text (assumed to be constant for all languages).

References:

In order to get a start for this task, I watched a few videos from here - https://www.coursera.org/course/nlp