Language models applied to the task of language identification

Jiarui Xu
University of Illinois at Urbana-Champaign
jxu57@illinois.edu

Table of Contents

File Structure

```
ling406-jxu57

/lingid
···./src
·····../langid              //package
···········...Langid.java          //main function *
···········...bidict.java          //dictionary class
···········...letterLangId.java    //letter bigram class
···········...wordLangId.java      //word bigram (add-one smoothing) class
···········...wordLangId2.java     //word bigram (good-turing smoothing) class
···········...testResult.java      //result testing class

letterLangId.out
wordLangId.out
wordLangId2.out
```

Program Structure

For more information about the functions, please refer to source code files

Langid.java

```
public class Langid {

    public static void main(String[] args) throws UnsupportedEncodingException, IOException
{
        //Problem 1
        letterLangId.dictInit();
        letterLangId.testLang();
        compare("LangId.sol","letterLangId.out", "Q1: LetterGram Result:\n");
        //Problem 2

        wordLangId.dictInit();
        wordLangId.testLang();
        compare("LangId.sol","wordLangId.out","Q2: WordBiGram (Add-one smoothing)
Result:\n");

        //Problem 3
        wordLangId2.dictInit();
        wordLangId2.testLang();
        compare("LangId.sol","wordLangId2.out","Q3: WordBiGram (Good-Turing smoothing)
Result:\n");
    }
}
```

bidict.java

```
public class bidict {

    HashMap hm=new HashMap();//[String: Frequency]
    HashMap freq=new HashMap();//[freq: Frequency of freq]

    HashMap newpGT=new HashMap();//[String: new Frequency]
    HashMap newpGT_norm=new HashMap();//[String: probability]
```

```
    double sum;//sum of all post words
    int nonzeroSum=0;//number of distinct words
    //Constructors
    bidict()
    bidict(String pair)
    bidict(String pair, String op)
    //add method
    public void add(String newChar)
    //Add-one dicts
    public void addLibOne(String newChar)
    public double prob(String theChar)
    //Good Turing dicts
    public void addLibGt(String newChar)
    public double probGTuring(String postWord)
}
```

letterLangId.java

```
public class letterLangId {
    private static final HashMap enDict = new HashMap();
    private static final HashMap frDict = new HashMap();
    private static final HashMap itDict = new HashMap();

    private static final String en = "LangId.train.English";
    private static final String fr = "LangId.train.French";
    private static final String it = "LangId.train.Italian";
    public static void dictInit()
    public static void testLang()
    private static void enrichLibDict()
    private static void splitLetters(String word, String op)
    private static String testLetters(String line)
}
```

wordLangId.java
wordLangId2.java

```
    //Dictionaries
    private static final HashMap enDict = new HashMap();
    private static final HashMap frDict = new HashMap();
    private static final HashMap itDict = new HashMap();

    /*
    En/Fr/It Words HashMaps to calculate the total number of words in that  langugae
    allWords HashMap it used to calculate the total number of words in all languages
    */
    private static final HashMap enWords = new HashMap();
    private static final HashMap itWords = new HashMap();
    private static final HashMap frWords = new HashMap();
    private static final HashMap allWords = new HashMap();

    //filenames under root
    private static final String en = "LangId.train.English";
    private static final String fr = "LangId.train.French";
    private static final String it = "LangId.train.Italian";
    public static void dictInit()
    public static void testLang()
    private static void enrichLibDict()
    private static void splitWords(String line, String op)
    private static void dictWords(ArrayList<String> warray,String op)
```

```
    private static void hashWords(String par, String chi, String op)
    private static String testWords(String line)
    private static String calcuWords(ArrayList<String> wordArray)
```

testResult.java

```
    public static void compare(String file1, String file2, String method)
```

Method

Comparing Standard
$$W_1 \ldots W_n \text{ as } W_1^n$$ $$P(W_1^n) = P(W_1)P(W_2 \mid W_1)P(W_3 \mid W_1^2 \ldots P(W_n \mid W_1^{n-1})$$ $$= \prod_{k=1}^{n} P(W_k \mid W_1^{k-1})$$ [1] Calculate probabilities for three languages and choose the corresponding language with the maximum probability.
Models
letter bigram model word bigram model (add-one smoothing) word bigram model (Good-Turing smoothing)

Assumption

This program has some assumptions as follows:

1	For the [String 1| String 2] bigram, if “String 2” does not exist in all three language corpuses, then: $P_{\text{language A}}[\text{String 1}\mid \text{String 2}] = N_{\text{words in language A}}/N_{\text{distinct words in all three languages}}$
	Reason: The more words a particular language has, then the more likely that an unseen word belongs to that language. The practical problem resulting in this assumption is that, since it’s not in corpus, we don’t have a bidict.class for that, so that this could case program exception.

	For example, English Corpus has 100 words, French Corpus has 50 words, Italian Corpus has 50 words, All corpuses have 180 distinct words (there may be some words existing in two or more languages) Sentence “UIUC is” “UIUC” doesn’t belong to the three corpuses, so that $P_{English}$[“is” | “UIUC”] = 100/180
2	For the last term $String_{end}$ in a sentence, a pair [$String_{end}$, “|THIS IS ENDING|”] will be processed to dictionaries.
	Reason: It’s necessary to consider the last term and make pair with a special string to indicate that this letter/word is likely to be at the end of a sentence.
3	In Good Turing smoothing, $c_0^* = c_1$, so that Frequency of frequency 1 is used to represent the number of unseen objects. If Frequency of frequency 1 is zero, which means that frequency 1 doesn’t exist, we used the frequency min to represent the frequency 1 in Good Turing formula.
	Reason: Sometimes, the dictionary doesn’t have frequency 1 letter/word. For example: A possible dictionary for word ”am” [good: 2] [bad: 3] [kind: 3] [cold: 10] [happy: 10] [sad: 10] [student: 11] [very: 100]
4	For the Good-Turing formula: $c^* = (c+1)\frac{N_{c+1}}{N_c}$ [2] if N_{c+1} =0 (does not exist), then $N_{c+1} = N_1$ (if N_1=0, see assumption 3 statements) [3]
	Reason: A possible dictionary for word ”is” [what: 1] [good: 2] [bad: 3] [cold: 11] [student: 11] [very: 100]

	Then the Frequency of frequency table is: [1:1] [2:1] [3:1] [11:2] [100:1] To calculate the adjusted value for 3, we have $(3+1)*N_4/N_3$, since $N_4=0$, we assign N_1 to N_4

Result

Q1:	**LetterGram**
	Misclassified: 26; Correctly classified: 274; Correction Rate: 0.9133333333333333
Q2:	**WordBiGram (Add-one smoothing)**
	Misclassified: 8; Correctly classified: 292; Correction Rate: 0.9733333333333334
Q3:	**WordBiGram (Good-Turing smoothing)**
	Misclassified: 9; Correctly classified: 291; Correction Rate: 0.97

Analysis

Advantages and disadvantages of the three models

Q1:	**LetterGram**
Pros	1. Easy to implement 2. Much smaller dictionary size. Since each language has limited number letters so that the dictionaries are much smaller that other models 3. Much quicker. It's time-efficient as it has smaller dictionary size, so that the time needed for looking up items will be less 4. It requires smaller running space. (due to the smaller dictionary size) 5. Almost impossible to come across unseen letters
Cons	1. Correction rate is low. 2. Not comprehensive enough 3. It's not able to identify different language well because letter combinations may overlap between languages, e.g. "e|g" may exist in all three languages

	and the difference is not large
Q2:	**WordBiGram (Add-one smoothing)**
Pros	1. Easier to implement than Good-Turing smoothing), but still harder than letterBiGram. 2. Correction rate is high 3. Quicker than GT smoothing but slower than letterBiGram 4. Not common in practice
Cons	1. Much bigger dictionary size. 2. Need some assumptions to deal with exceptions 3. Using word bigram amplifies the difference between languages 4. Too much portion of probability is transferred to the unseen items [4] 5. Probability of frequent n-grams is underestimated [4] 6. Probability of rare (or unseen) n-grams is overestimated [4]
Q3:	**WordBiGram (Good-Turing smoothing)**
Pros	1. Correction rate is high 2. Limited mass of probability is shifted 3. Keep the portion of high occurrence probabilities 4. More frequently used in practice
Cons	1. Hard to implement 2. Slow in identifying 3. Much larger dictionary size 4. Need much more calculations including summation, multiplication and normalization 5. There are many exceptions requiring assumptions to solve. (For example, we may not have items with frequency n+1)

Problem

1. **The result might be different from same model's using different methods**

There are probably different methods for implementing the three models, such as linear regression for estimation non-existed n+1 frequency[5], but here I only implemented one method for each model with some certain assumptions. To compare the three models, we may need to try different methods. Since this is a case study assignment, I didn't implement all the methods.

2. **The result is not obvious in showing the advantage of GT over Add-one**

The two smoothing methods showed very similar result, which could be ignored as the difference may result from the floating arithmetic precision problem or the relatively small size of corpuses. However, Good-Turing smoothing should be better than the Add-one smoothing

theoretically. Here, due to the limitation of corpus and test set, this wasn't reflected in my result.

Improvement

1. Use larger corpuses
2. Use larger test set
3. Try different methods or assumption

Conclusion

The case study of three models in identifying languages shows that Word Bigram (add-one and GT) has better correction rate than Letter Bigram. Add-one and GT smoothing methods have very close correction rates due to the limited sizes of corpuses and test set and probably floating arithmetic precision issues. Theoretically, Good-Turing smoothing is more scientific and comprehensive than Add-one smoothing.

Source

[1] Prof. Girju Lecture4: N-gram slides p10
[2] Prof. Girju Lecture4: N-gram slides p54
[3] Prof. Girju (Feb 26th, 2015, after lecture)
[4] Prof. Dorr, Prof. Monz, Lecture 5: Smoothing slides , http://www.umiacs.umd.edu/~christof/courses/cmsc723-fall04/lecture-notes/Lecture5-smoothing-6up.pdf p2
[5] Gale, William, and Geoffrey Sampson. "Good-Turing smoothing without tears." *Journal of Quantitative Linguistics* 2.3 (1995): 217-237.