Language recogniser based on bigram counts of characters

DCU: Graduate Diploma in IT
Marek Dano
10211124

ABSTRACT - *Language recognition is a core element in field of Natural Language Processing (NLP), which is an area of research and application that explores how computers can be used to understand and manipulate natural language text to do useful things. Applications of language recognition include machine translation, natural language text processing, language user interfaces, artificial intelligence and many more.*

Recognising languages might base on an n-gram method where matching solution is applied or detailed linguistic method where deeper analysis of the language is needed. In this study we used the n-gram method.

The study contains analysis of theory, how n-gram method is used to recognise the language, also training and testing phase, the experimental results and conclusions.

1. INTRODUCTION

Language recognition models can be applied and based on grammar model or n-gram model.

The grammar model requires knowledge of syntax of studied language. Lexical and grammar rules must be provided for this model. More complex analysis of the language is also needed.

For our language recogniser we implement the n-gram model and design the application for two languages Slovakian and English.

2. BACKGROUND

2.1. Theory

The n-gram model is based on a text corpus from a given language. The corpus can be any text with a large amount of real language data. In training phase the language model is created from this given text. It is based on the machine computed probabilities. Probabilities are calculated by occurrences of characters or words. Each given language generates a probabilistic model. Sequences of characters or words are defined from a given text. N-grams of these sequences are created.

The n-gram can be specified on characters-based or word-based n- gram. A character-based n-gram is a consecutive sequence of n characters selected from a word. The type of n-gram is based on the number n. For example taking n value of 1 is series of unigram, for n of 2 series of bigrams, n of 3 series of trigram and so on. This language recogniser uses character-based bigram, n is value of 2.

2.2. Probabilistic model

For a given unseen text *(T)* in some language *(L)* an n-gram enables us to compute the probability $P(L|T)$ of this unseen language text. If we have a set of module $U = \{L_1, L_2, \dots L_N\}$ the calculation of the language L_{chosen} can be represented by this formula:

$$L_{chosen} = argmax_{\,Li \in U}\ P(L_i \mid T)$$

The language L_{chosen} will be the language which returns the highest probability for the unseen text *(T)* from all languages in the set *(U)*.

But $P(L_i|T)$ is difficult to calculate so the Bayes' rule of probability is used to simplify this formula:

$$L_{chosen} = argmax_{\,Li \in U}\ P(T \mid L_i)P(L_i) / P(T)$$

Assuming that $P(T)$ is the same for each calculation and all languages are equally likely: $P(L_1) = P(L_2) = P(L_3) = \dots = P(L_N)$ the formula is given:

$$L_{chosen} = argmax_{\,Li \in U}\ P(T \mid L_i)$$

for each language in set of all languages *(U).* The chosen language L_{chosen} will be the language which computes the highest probability.

2.3. Bigram probabilities

For training the language model based on bigrams we must store all probabilities of all characters from the training language corpus. It is presented by a sequence of bigrams:

$$b_1^N = b_1 b_2 b_3 \dots b_N$$

Than we can determine the probability of this sequence using the chain rule of probability [1]:

$$P(b_1^N) = P(b_1)\, P(b_2|b_1)\, P(b_3 \mid b_1^2) \dots P(b_N \mid b_1^{N-1})$$

$$P(b_1^N) = \prod_{i=1}^{N} P(b_i \mid b_1^{i-1})$$

Further simplification using Maximum Likelihood Estimation (MLE) [1] which derives probabilities of a bigram b_i from a training corpus, where *C()* is a counting function.

$$P(b_i \mid b_{i-1}) = \frac{C(b_{i-1}b_i)}{\sum_b C(b_{i-1}b)} = \frac{C(b_{i-1}b_i)}{C(b_{i-1})}$$

Each bigram is generated from training data corpus. It is rated as a sequence of characters $(c_{i-1}c_i)$. Then we can derive the probability of each bigram b_N.

$$P(b_N) = P(c_i \mid c_{i-1}) = \frac{C(c_{i-1}c_i)}{C(c_{i-1})}$$

$$P(b_N) = \frac{frequency\ of\ (c_{i-1}c_i)}{count\ of\ (c_{i-1})}$$

To develop this language model based on bigrams we require knowledge of calculation the frequency of bigrams and the count of occurrences of each character. This knowledge is required for creating probabilistic model.

3. METHOD

3.1. User interface

An object oriented program was devised with MS-DOS written to allow users to train and test different sources of data.

The training set of two languages data is taken to the program as two arguments. Asking users to identify a language of input text files. The test corpus text file is detected into program after inserting the name of testing file which is in the same directory with this executed program file. The program shows an unseen text and the most probable language of this tested text. The result of all probabilities for each language is also displayed. After displaying the result, users are allowed to test more testing text data. The relevant question is implemented to process it. Possible errors in reading and accessing training and testing text data were also implemented. When an error is occurred in reading from the training text file or text file is empty, the system is terminated.

3.2. Representation of data structures

Data Structures are represented by:

unigrams[ ] – stores the occurrences of each character
bigrams[ ] – stores bigram nodes containing the bigram key, occurrences of each bigram and relative probability of each bigram

The language recogniser is designed to read a text file containing the Unicode characters. Data structures are required for processing characters. The text from text file is pre-processed. That means the text is changed into lower case letters and removed all punctuation. All accented characters are replaced with unaccented ones. The result of this process is a set of 27 characters ('space' and small letters a-z).

Each single character represents the unigram stored in 27 slotted *unigrams[ ]* as shown in Figure1.

0	1	2	3		24	25	26
'space'	'a'	'b'	'c'	....	'x'	'y'	'z'

Figure 1: Representation of all unigrams.

Accessing to *unigrams[ ]* uses the hashing based on Unicode decimal value of character. The following assignment is applied for character *c,* where *c* is a set of letters *a, b, c, ..., y, z*:

Unicode(*c*) – 96

The character 'space' maps to zero slot in *unigrams[ ].*

All bigrams are represented by all possible combinations of 27 characters, which are $27^2 = 729$. Knowing all possible bigrams, the array *bigrams[ ]* with 729 slots is applied. All combinations of the character can categorized as:
" " (two 'space's), " a", " b", ..., "aa", "ab", ..., "a ", "bb", ..., "zy", "zz". Hashing function uses to map all bigrams based on Unicode value of bigram to array *bigrams[ ].*
The hashing algorithm for bigram (ch_1ch_2), if ch_1, ch_2 are characters a-z and 'space', is:

1. if ch_1 is 'space'
   1.1. $c_1 = 0$
2. if not the 'space'
   2.1. c_1 = unicode(ch_1) - 96
3. if ch_2 is 'space'
   3.1. $c_2 = 0$
4. if not the 'space'
   4.1. c_2 = unicode(ch_2) – 96
5. $key = c_1 * 27 + c_2$;

where c_1, c_2 are numbers between 1-26 and *key* is hashcode. The 'space' character is assigned to value zero.

Implementation of mapping all bigrams to array *bigrams[ ]* is shown in Figure 2.

" "	" a"			" z"	"a "	"aa "		"z "		"zy"	"zz"
0	1	....	....	26	27	28	....	702	....	727	728

Figure 2: Representation of *bigrams [ ]* through hashing function

We know that training for each train language executes only ones. Than the manipulation as inserting, deleting and searching in this hashtable represented by key-indexed array has time complexity *O(1)* [2]. That has major effect on speed of the program.

3.3. Training and testing phase

Implementation of the language recogniser is divided into two phases, a training phase and a testing phase.

The training phase generates the bigrams for each training language in set of language modules. Then computes the probability of each bigrams of the train language and stores it in an array.

The testing phase calculates empirical probability of an unseen testing text for each trained language and chooses language which returns the highest probability.

3.3.1. Training phase

The training text data is read, pre-processed and trained using an algorithm in this training phase. The algorithm for training data selects and counts occurrences of each characters and bigrams in training corpus. For each bigram is calculated relative probability. All those data are stored in relevant slot in the array.

Algorithms for training

1. For each character and bigram in the training text increment the occurrence count of that character and bigram. The character is stored in *unigrams[]* and bigram in *bigrams[ ].*
2. For each bigram compute the relative probability of that bigram and store it in *bigrams [ ].*

The calculation of relative probabilities of bigrams assumes that all bigrams appeared in training text. That assumption is needed to further usage of that data.

To achieved that we perform a smoothing function to the language model. There are many different smoothing algorithms known as Laplace smoothing, Good Turing discounting, Witten-Bell discounting and Kneser-Ney smoothing [1].

In this application is used Laplace smoothing. This arithmetic formula for calculation of bigrams:

$$P(b_N) = P(c_i \mid c_{i-1}) = \frac{C(c_{i-1}c_i)}{C(c_{i-1})},$$

is modified for calculation of probabilities for each possible bigram. The Laplace formula is used:

$$P_{Laplace}(c_i \mid c_{i-1}) = \frac{C(c_{i-1}c_i) + 1}{C(c_{i-1}) + \mathrm{V}},$$

where V is an vocabulary of characters (small letters a-z and 'space', $V=27$) . The Laplace smoothing is applied to each bigram in bigrams[ ]. The algorithm for calculation of probability with considering this Laplace smoothing is shown as:

1. for each bigrams in *bigrams[ ]*
   1.1. get number of frequencies of the bigram through hashing method
   1.2. get the counts of unigram for the particular bigram
   1.3. compute relative probability of the bigram using this formula:

bigram probability= (number of frequencies of the bigram + 1) / (counts of unigram + vocabulary of characters).

3.3.2. Testing phase

The test data of an unseen text are read, pre-processed and then all bigrams are generated from it. We calculate an empirical probability of testing language model for each bigram in that text data. For bigrams $b_1, b_2, \ldots, b_N$ from the testing text *T* following calculation of empirical probability is:

$$P(T \mid L) = P(b_1) * P(b_2) * \ldots * P(b_N).$$

Assuming that each probability is a small number we can modified the arithmetic formula as:

$$P(T \mid L) = \log(P(b_1)) + \log(P(b_2)) + \ldots + \log(P(b_N)).$$

During testing text data following algorithm is used.

Algorithm for testing

1. Initialise probability for each new testing text.
2. Read probability of each extracted bigram from testing text
3. Compute the empirical probability, probability += log(probability of bigram).

4. RESULTS & DISCUSSION

For testing this application we chose training and test corpus of text data. The results of testing text depend on the training text. Test corpus needs to look like training corpus and be sized as a 10% of training corpus [1].
The training corpus is a set of text files from different language styles. A set of training and testing text is a combination of texts from newspapers, academic papers, web pages, fiction and poems.

The language recogniser recognises from two different languages, Slovakian and English. Slovakian text is pre-processed as it is shown in Table 1. The rules shown in this table are applied for training and testing text. Then processed text contains only letter (a-z) and 'space'.

Slovakian letters	Processed characters
á ä	**a**
é	**e**
í	**i**
ô ó	**o**
ú	**u**
ý	**y**
č	**c**
ľ ĺ	**l**
ň	**n**
š	**s**
ť	**t**
ž	**z**

Table 1: Slovakian pre-processed text rules

The training text data is created as a text from newspapers [3, 4], web pages [5, 6] and poems [7, 8, 9]. Each text file contains around 12 000 characters.

The set of test corpus is created as *T1* and *T2*. Each testing text file contains around 1200 characters. *T1* set is extracted from web pages and it is written in academic style [10, 11]. The results of testing *T1* corpus are shown in Figure 3.

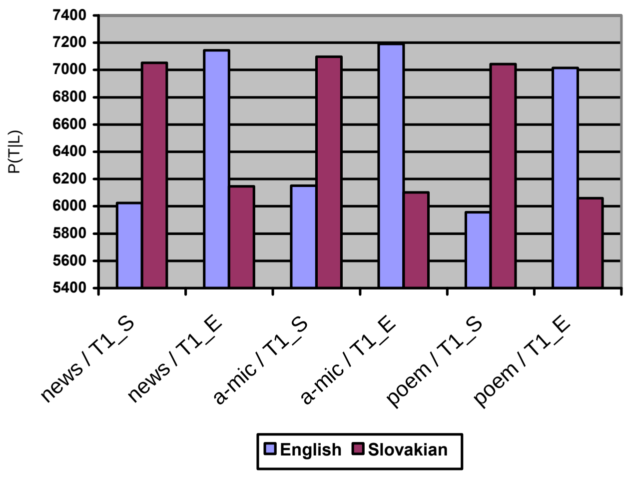

Figure 3: Results of testing text *T1* with three training sets

The *T2* set is written in fiction style [12, 13]. Results of this testing set are shown in Figure 4.

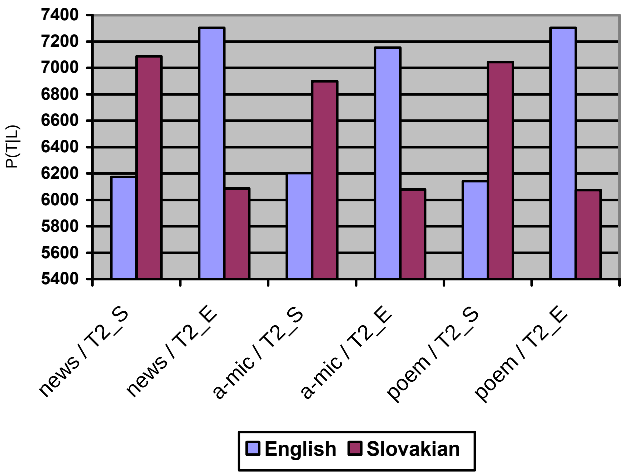

Figure 4: Results of testing text *T2* with three training sets

As seen in Figure 3, 4 results of all testing show us that the language recogniser chose language with the highest probability. Success of choosing the correct language is 100%. But difference between empirical probabilities of languages is not in wide range.

Analysing an average length of words in both languages we state the reason of narrow differences. The average length of words in Slovakian is 5-6 characters [14], in English it is 5 characters [15]. We assume that the difference between calculated probabilities of other chosen languages would be different, but that could be a subject of further study.

5. CONCLUSIONS & RECOMMENDATIONS

The Language recogniser based on bigram counts of characters was successfully implemented. This implementation required deep analysing and studying of the bigram probabilistic model. It also required choosing adequate data structures and algorithms to maximise the effectiveness of this application. The study of this bigram model involved analysing some linguistic aspects of languages. Reasonable training and test corpus is made for our testing. All results of choosing correct language are successful within not a wide range of language empirical probabilities.

That can be as a reason for further study of lexical analysis training and testing languages, which involves

- implement trigram probabilistic model for accurate empirical probabilities of languages or
- added another characters sets and data structures into the application or
- testing this application with another size of training and test corpus.

For useful interaction with users and more attractive application a graphical user interface (GUI) can be implemented in another stage of improvement.

6. REFERENCES

[1] Daniel Jurafsky and James H. Martin, Speech and Language Processing: An introduction to speech recognition, computational linguistics and natural language processing, draft of January 7, 2007

http://www.mit.edu/~6.863/spring2009/jmnew/4new.pdf

[2] Elliot B. Koffman, Paul A. T. Wolfgang, Data Structures: abstraction and design using Java, 2nd edition, John Wiley & Sons, New York, 2010

[3] http://www.independent.ie
[4] http://www.sme.sk
[5] http://en.wikipedia.org/wiki/Volcano
[6] http://sk.wikipedia.org/wiki/Sopka
[7] http://www.william-shakespeare.info/william-shakespeare-poem-venus-and-adonis.htm
[8]http://zlatyfond.sme.sk/dielo/17/Sladkovic_Marina/2#ixzz1LK98r9hj
[9http://zlatyfond.sme.sk/dielo/85/Botto_Smrt-Janosikova/1#ixzz1LK83JCjb]
[10] http://sk.wikipedia.org/wiki/Zem
[11] http://en.wikipedia.org/wiki/Earth
[12]http://www.catholicireland.net/component/cifeed?option=com_cifeed&task=readings&lang=eng
[13] http://lc.kbs.sk/
[14] http://www.juls.savba.sk/ediela/sr/1967/6/sr1967-6-lq.pdf
[15] http://www.groupsrv.com/science/about451323.html

APPENDIX

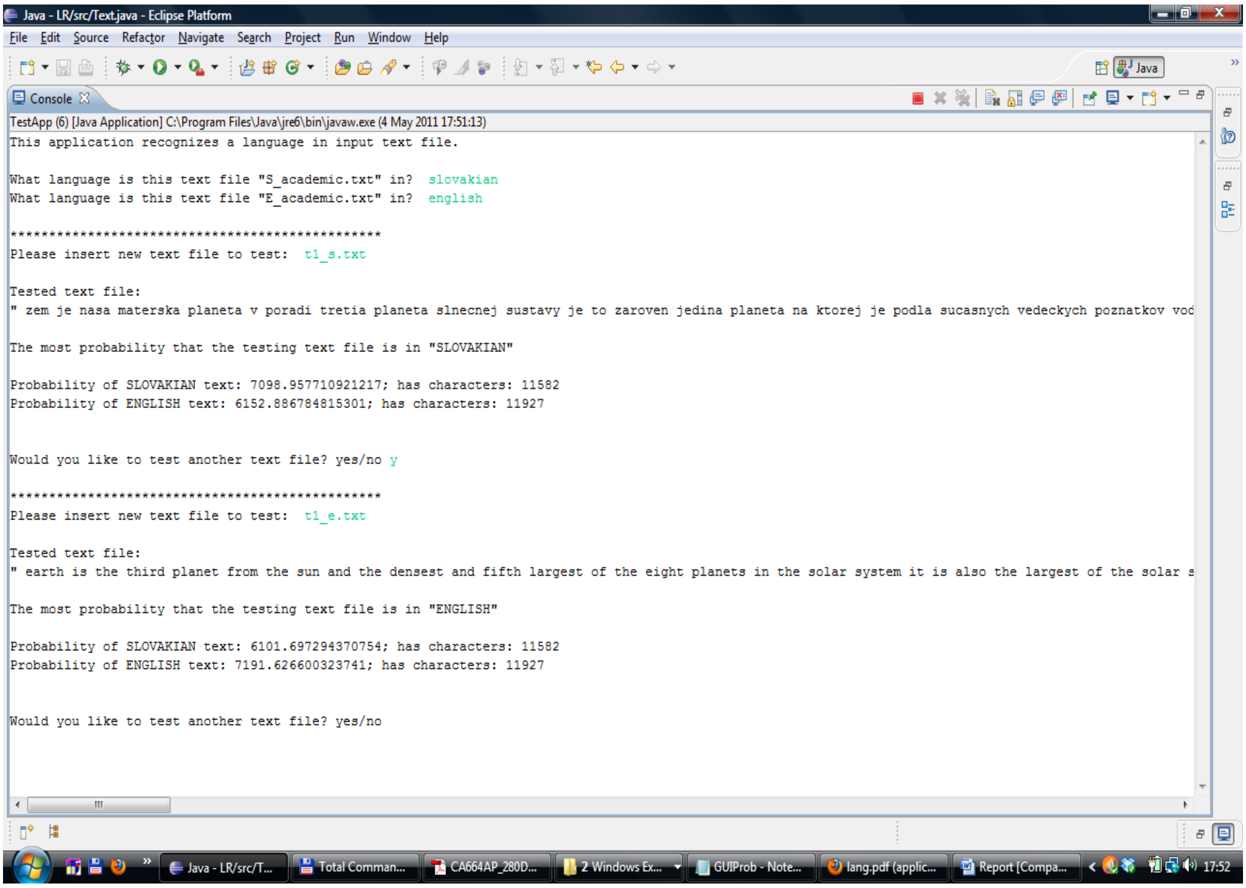

Figure 1: Training academic text files with test1 (T1) text files.

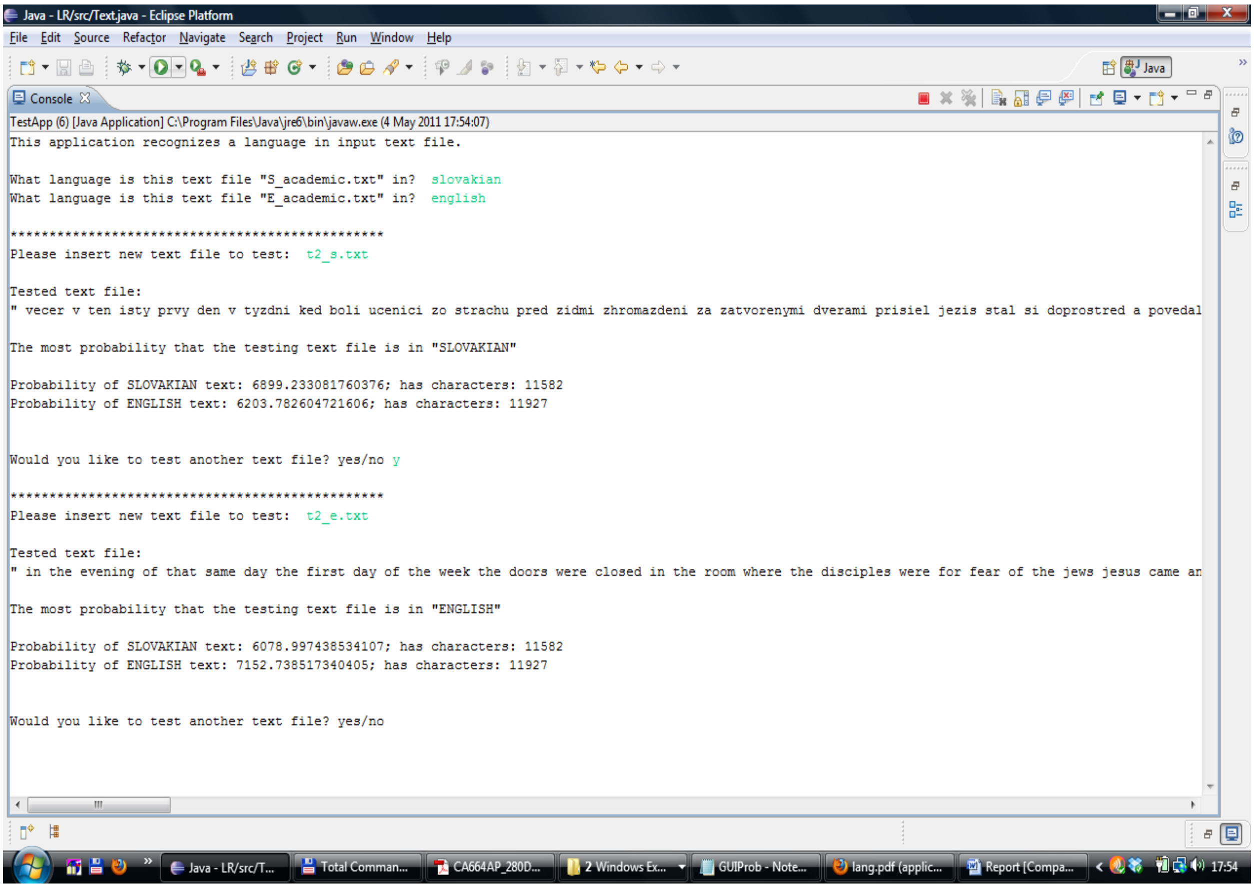

Figure 2: Training academic text files with test2 (T2) text files.

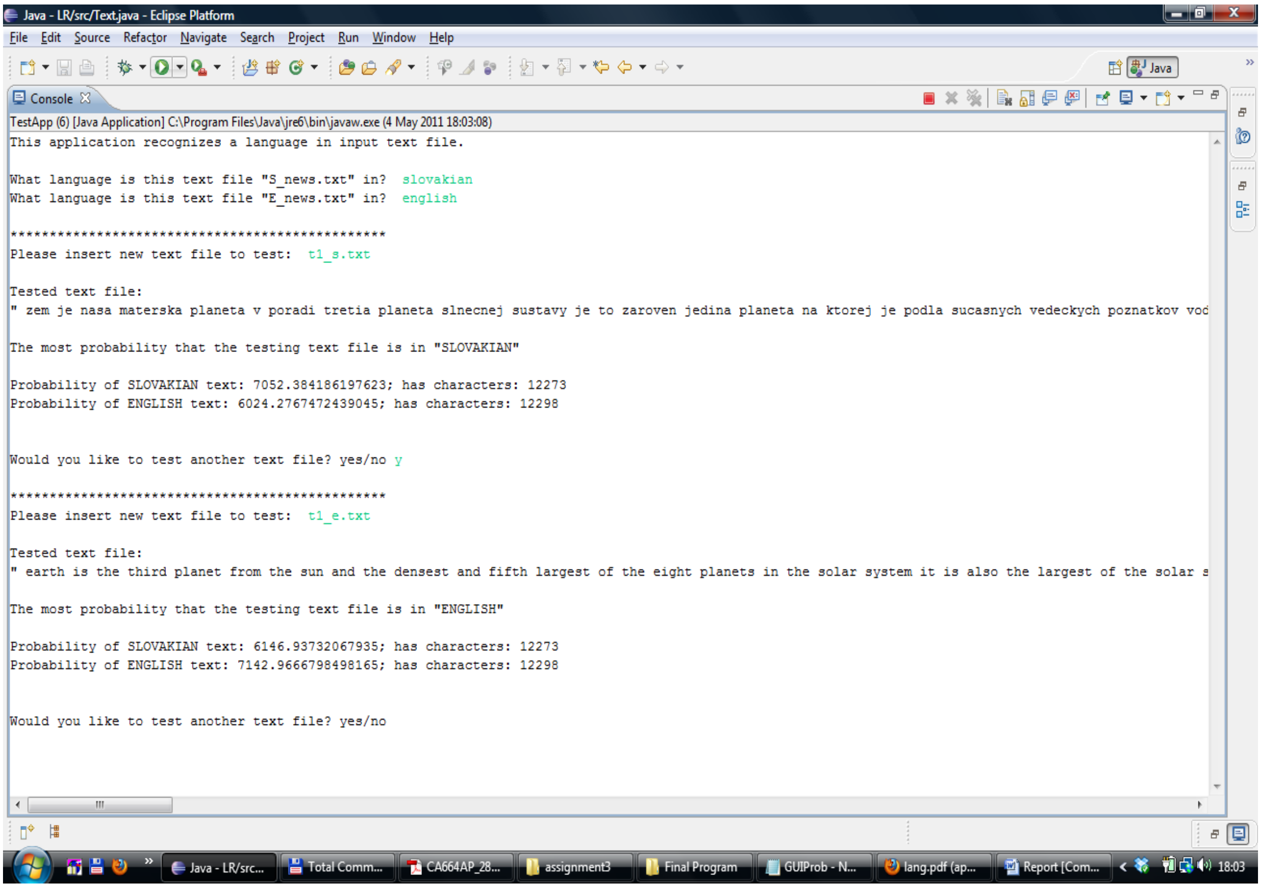

Figure 3: Training newspapers text files with test1 (T1) text files.

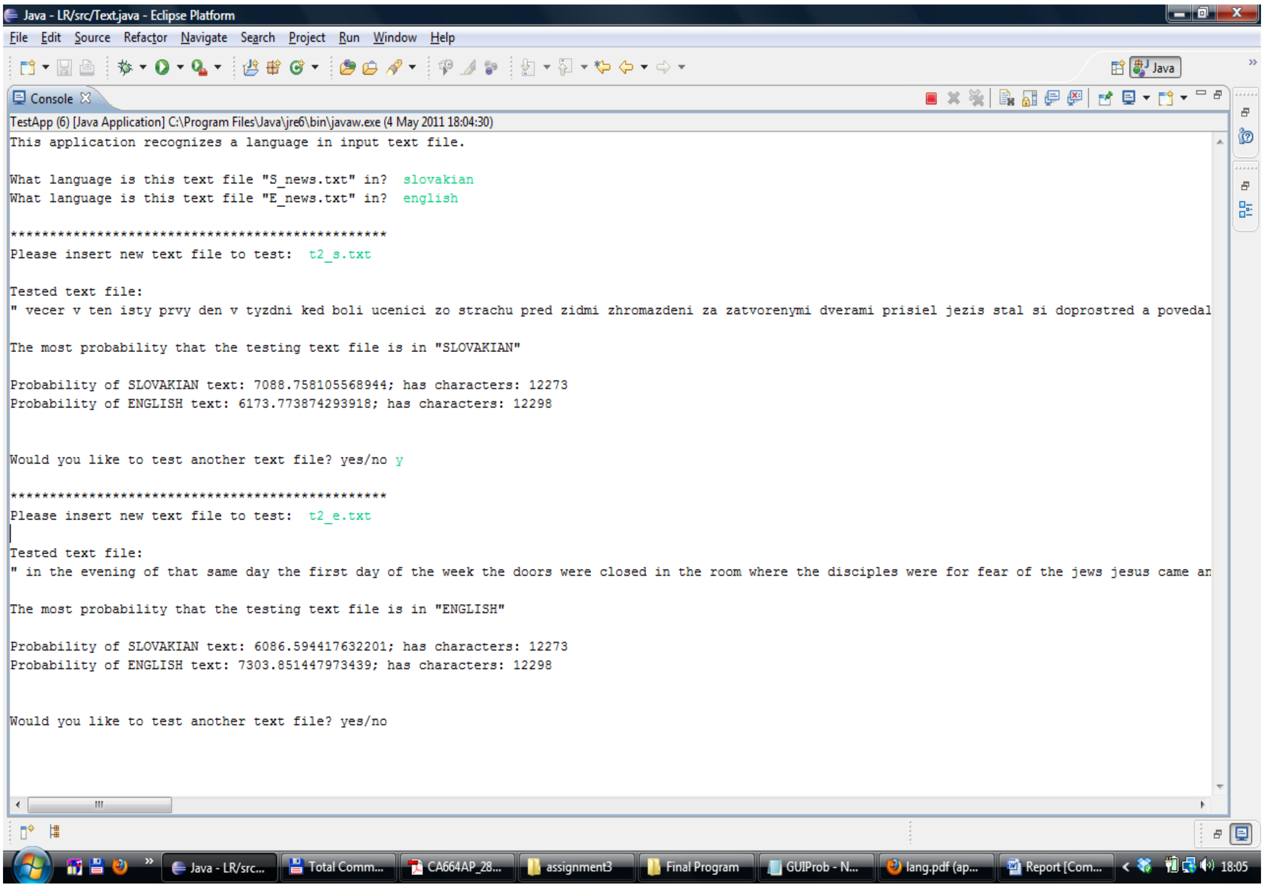

Figure 4: Training newspapers text files with test2 (T2) text files.

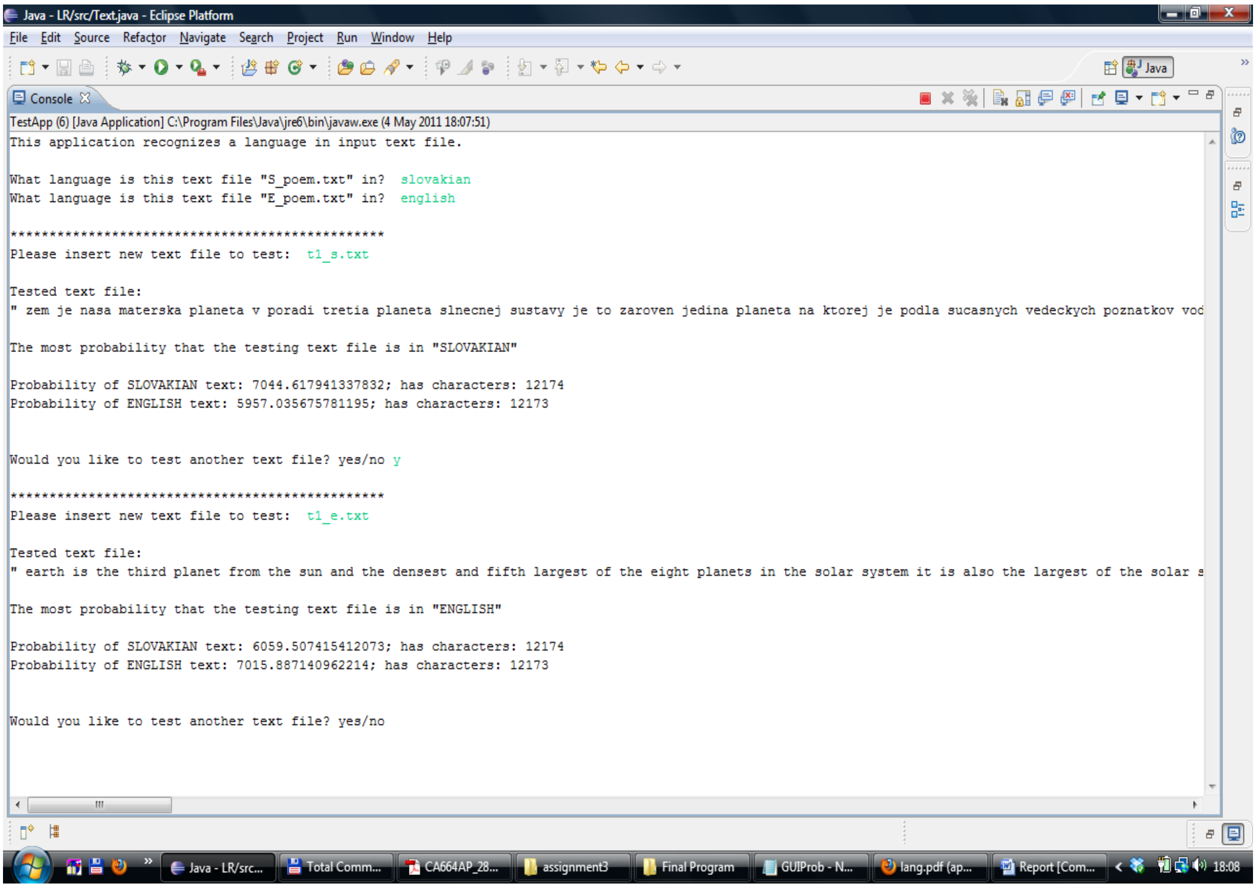

Figure 5: Training poems text files with test1 (T1) text files.

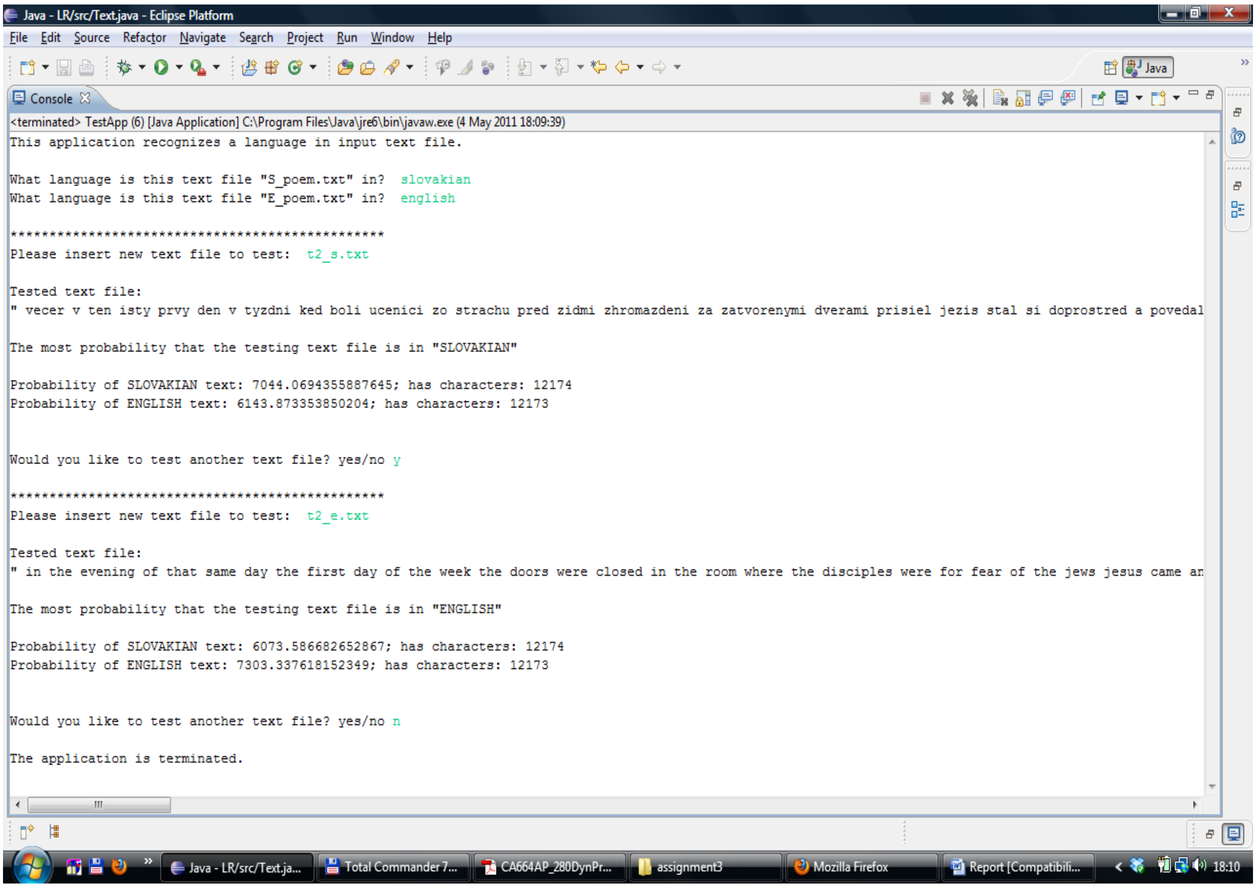

Figure 6: Training poems text files with test2 (T2) text files.

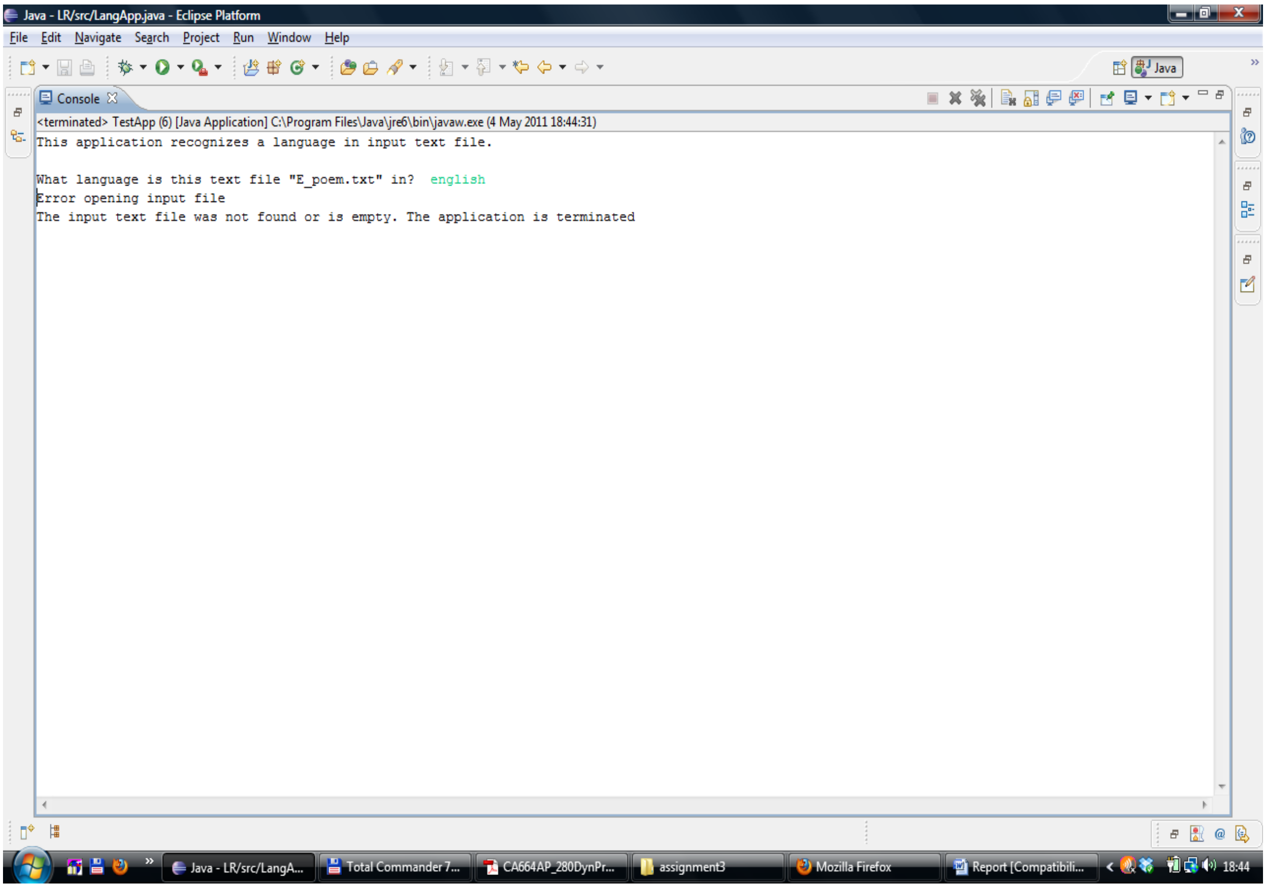

Figure 7: The second input text file was not found and system was terminated.

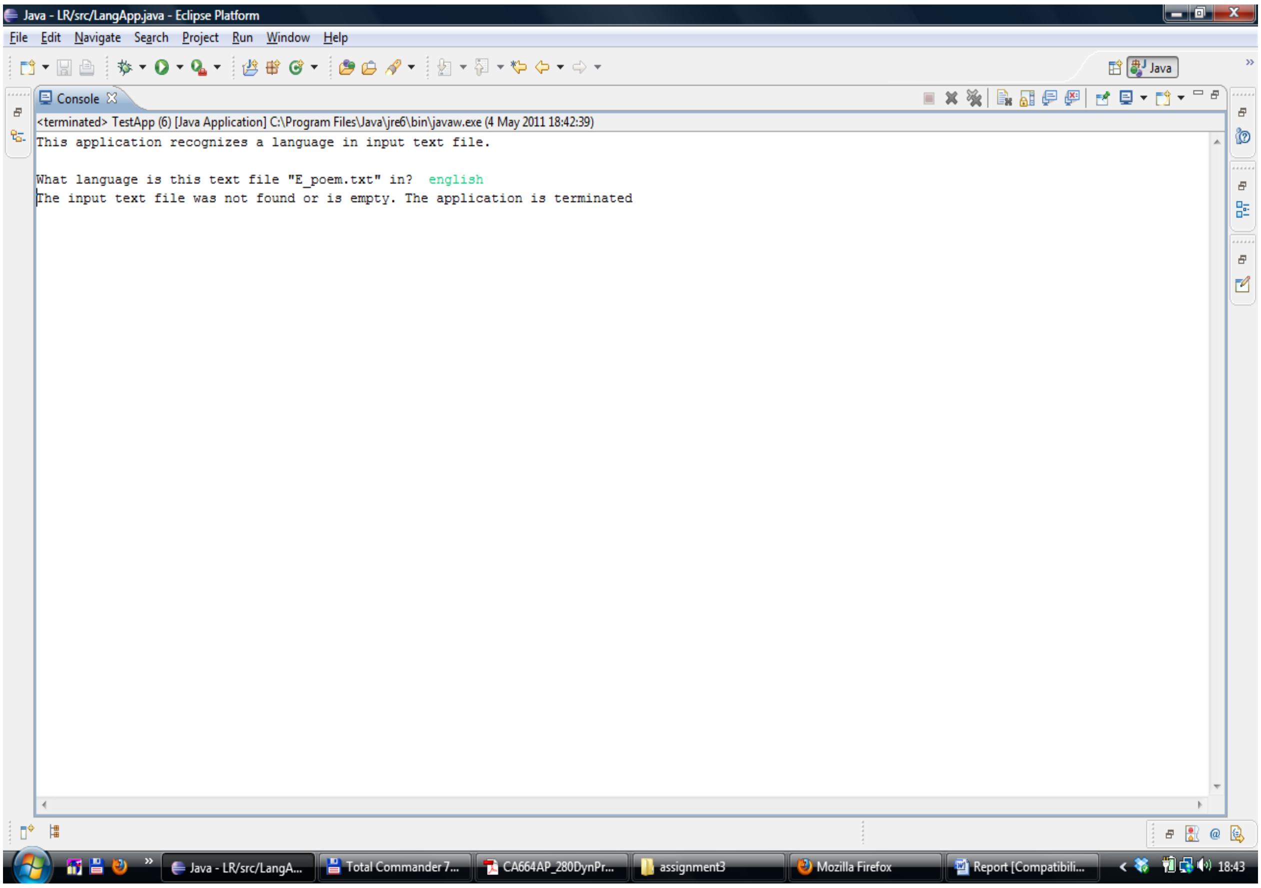

Figure 8: The second input text file is empty.