Language Identification on Short Textual Data

LANGUAGE IDENTIFICATION ON SHORT TEXTUAL DATA

BY

YEXIN CUI, B.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2019) McMaster University

(Electrical & Computer Engineering) Hamilton, Ontario, Canada

TITLE:	Language Identification on Short Textual Data
AUTHOR:	Yexin Cui B.Eng., (Electronics and Information Engineering) Zhengzhou University, Zhengzhou, China
SUPERVISOR:	Dr. Jun Chen
NUMBER OF PAGES:	ix, 51

To my dear parents and my love

Abstract

Language identification is the task of automatically detect the languages(s) written in a text or a document given, and is also the very first step of further natural language processing tasks. This task has been well-studied over decades in the past, however, most of the works have focused on long texts rather than the short that is proved to be more challenging due to the insufficiency of syntactic and semantic information. In this work, we present approaches to this problem based on deep learning techniques, traditional methods and their combination. The proposed ensemble model, composed of a learning based method and a dictionary based method, achieves 89.6% accuracy on our new generated gold test set, surpassing Google Translate API by 3.7% and an industry leading tool Langid.py by 26.1%.

Acknowledgements

I would like to first express my sincerest and deepest gratitude to my supervisor Dr. Jun Chen from Department of Electrical and Computer Engineering, McMaster, not only for his initial and ongoing support for this thesis, but also for his patient, careful and consistent supervision throughout my Master program. I would not have completed this work successfully without his support and guidance.

Furthermore, I would like to thank Mr. Chris Priebe, Mrs. Liza Wood and Mr. Shuo Liu for their fruitful advice and full support for this project. It has been a great time and honor for me to work with them.

Last but not least, I would like to thank Dr. Jiankang Zhang and Dr. Dongmei Zhao for being members of my defence committee. I appreciate their time for reviewing my thesis and providing valuable feedback.

Notation, Definitions, and Abbreviations

AWS	Amazon Web Service
BERT	Bidirectional Encoder Representations from Transformers
Bi-RNN	Bidirectional Ruccurent Neural Network
CBOW	Continnuous Bag of Words
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
NLP	Natrual Language Processing
OOV	Out-Of-Vacabulary
OOP	Out-of-Place
RNN	Recurrent Neural Network
SVM	Support Vector Machine

Contents

Abstract **iv**

Acknowledgements **v**

Notation, Definitions, and Abbreviations **vi**

1 Introduction and Problem Statement **1**

1.1 Introduction 1

1.2 Thesis Structure 3

2 Background and Related Work **5**

2.1 Traditional Solutions 5

2.2 Learning-based Solutions 8

3 The Proposed Methods **13**

3.1 Construction of New Multilingual Datasets 13

3.2 Learning-based Methods 19

3.2.1 fastText 19

3.2.2 BERT 21

3.3 Dictionary-based Method 26

3.3.1 Construction of Dictionary 26
3.3.2 Algorithm 28
3.4 Ensemble Model 31

4 Implementation and Experimental Results **32**
4.1 Datasets for Training and Testing 32
4.2 Implementation 34
4.3 Evaluation Metrics 35
4.4 Results and Comparison 38

5 Conclusion and Future Work **43**

List of Figures

2.1 Out-of-Place measure calculation between two profiles (adapted from Cavnar and Trenkle (1994)). 7

2.2 Training procedure of the RNNs ensemble (source from Mathur *et al.* (2017)). 12

3.1 This is the model architecture of fastText for a sentence input with character N-gram features $x_1, x_2, ..., x_N$. Then the features are embedded and processed to form the hidden layer. 20

3.2 The structure of Transformer (source from Vaswani *et al.* (2017)). 22

3.3 The structure of multi-head attention (source from Vaswani *et al.* (2017)). . 24

3.4 Example of the dictionary in a language 27

3.5 Example of the format in Dict1 27

3.6 Example of word frequency in English list 28

3.7 Work flow of Ensemble Model integrating fastText and LMK for language identification 31

Chapter 1

Introduction and Problem Statement

1.1 Introduction

With the growth of people relying on social media services, a significantly large amount of data has been generated by users across the world in different languages, where the textual data made up of chat messages, Internet news and blogs form an important component. However, some of this information are of bias and discrimination, and even violence. More and more people suggest that there is a need to apply Natural Language Processing (NLP) techniques on such data and filter out the toxic information in order to maintain a healthy online community for everyone, especially for teenagers. Prior to this, automatic language identification is regarded as the first step for further NLP tasks in a multilingual environment such as machine translation, cyber bullying detection and toxicity detection.

Language identification (LID) aims to detect the language(s) given a text or a document written in one or more languages. LID on monolingual texts is known as a multi-class classification problem, in which a text is assigned to one of the K classes. This can be estimated by Eq.1.1 for One-vs-All classification, and Eq.1.2 for All-vs-All (all-pairs) classification

adapted from Knerr *et al.* (1990) and Friedman (1996).

$$f(x) = \underset{i}{argmax}\ f_i(x) \tag{1.1}$$

Where f_i is the ith classifier for distinguishing class i from all K classes, namely, taking samples from i as positive and other classes as negative.

$$f(x) = \underset{i}{argmax}\ (\sum_j f_{ij}(x)) \tag{1.2}$$

Where f_{ij} is the classifier distinguishing each pair of class i and j, i.e. taking class i as positive and class j as negative.

On the contrary, LID on multilingual texts is considered as a multi-label classification task, where a text can be mapped onto a subset of labels from a larger closed label set (Lui *et al.*, 2014). Accordingly, one assumption we make in this thesis, is that each text we process is monolingual.

LID has been studied over decades and is seemed to be addressed properly as its accuracy is found to be high from a number of isolated experiments over various unacknowledged datasets with small sets of languages (Hughes *et al.* (2006); Grothe *et al.* (2008)). However, most of the previous work were focused on long texts. Recently while some researchers turning their attention to LID on short texts, they have noted that it is more challenging than the long. As Baldwin and Lui (2010) mentioned, LID becomes increasingly difficult as we augment the number of languages and reduce the length of the texts. Carter *et al.* (2013) also points that microblogs like tweets that contains a majority of short posts is challenging even for state-of-the-art LID methods.

In this work, we present four methods for LID on short texts, three of them are individual approaches either based on traditional or learning based methods, and the other one is an ensemble model integrating fastText and our dictionary based method LKM. These methods are trained and evaluated on new multilingual datasets we generated from online resources and real-world experimental data provided by our collaborators TH, as there are not public datasets meeting our needs. The ensemble model achieves the best performance compared with the individual models, and even the remarkable classifiers Google Translate API and Langid.py.

The main contribution of our work is that we have proposed an competitive LID ensemble model, also implemented and compared the traditional based methods with leaning based on our task-specific datasets. Noting that our ensemble model is also able to perform multilingual classification task since we have appended an extra configuration for it. Once we achieve considerably results on current task, we will move on to the multilingual task in the future. Due to the convenience in configuration, our ensemble model can be further developed as an off-the-shelf product to address this kind of issue effectively without any training work.

1.2 Thesis Structure

The rest of this thesis is structured as follows: we successively review the early and current state-of-the-art solutions to LID in Chapter 2, which is categorized into two parts traditional and learning based solutions. Chapter 3 introduces our own built multilingual datasets and the four proposed methods, including the main steps of construing the datasets and technical details of the model architecture. In Chapter 4, we show the detailed implementation and the experimental results, as well as numerical comparisons between our

proposed methods and the industry leading solutions. At the end, we conclude our work and discuss the future plans of LID on short texts in Chapter 5.

Chapter 2

Background and Related Work

Language identification on written texts, also known as language detection and sometimes as language recognition, is a classification task in NLP (Gottron and Lipka, 2010), which has been constantly studied over decades. Many solutions have been proposed to this task, which more specifically, could be categorized into traditional solutions and learning-based solutions.

2.1 Traditional Solutions

Traditional solutions to LID are originally derived from word-based method, one of them is called *short word-based*, which merely uses words up to a certain length to construct the language model (Grothe *et al.*, 2008). Grefenstette (1995) and Prager (1999) extract words up to four and five characters separately after tokenized for each language to build the model. It is noted that common words like determiners, conjunctions and prepositions could be good hints for detecting a language, and they are all short in most cases. Given an input text tokenized, tokens captured mostly by the word list of a certain languages,

then this language is taken as the result. Another word-based method called *frequent word-based* brings in word frequency referring to the number of occurrences in corpus. It has been proposed by Souter *et al.* (1994) and Ludovik *et al.* (1999) in different work. Souter *et al.* (1994) extract the top 100 words for each language from training data to generate ordered frequency wordlists and 91% accuracy was achieved during the test. He also points out that its accuracy is highly dependent on the composition and size of the test file, though this method has advantage in identifying short text and languages of small quantities. The main limitation of word-based method is a large amount of corpus required to create the language model during training phase (Brodić *et al.*, 2017)

Going forward, a famous method of generating language model is by n-gram model, which takes the place of word-based method. An n-gram is a sequence of n character slicing from a longer string , where $n \in [1, 2, 3, 4, ..., n]$. Generally, special symbols would be appended at the beginning and ending of the string in order to help with distinction between prefixes, suffixes and other slices, for instance by underscore character ("_") and angel quotes ("<", ">"). For example, the word "hello" would be decomposed into the following n-grams:

$$bi - gram(2 - grams) : \_h, he, el, ll, lo, o\_,$$

$$tri - gram(3 - grams) : \_he, hel, ell, llo, lo\_$$

Cavnar and Trenkle (1994) apply n-gram in LID task by calculating and comparing the n-gram frequency of profiles via Out-of-Place (OOP) measure shown in Fig.2.1, a rank-order statistic referring to the distance of an n-gram in one profile to its place in another profile. They first compute n-gram frequency profiles per language on the training sets, then compare that from a test file with category profile of each language using OOP, and

finally pick the category with the smallest distance. This solution achieved a 99.8% accuracy on Usenet newsgroup articles written in different languages. One of weaknesses in their method is that it require inputs to be tokenized, limiting the applicability in cases where tokenization is difficult. Another similar approach addressing this issue has been proposed by Dunning (1994) who takes into account byte n-gram rather than character n-gram introduced above. This approach achieved 92% accuracy on test set with only 20 bytes (about 3 to 4 words in English) and 50K bytes of training, which could be improved to 99.9% when longer text with 500 bytes were tested, assuming both the training and test data are encoded in sequences of bytes. However, an challenge would rise in calculation and comparing steps if the inputs are multilingual.

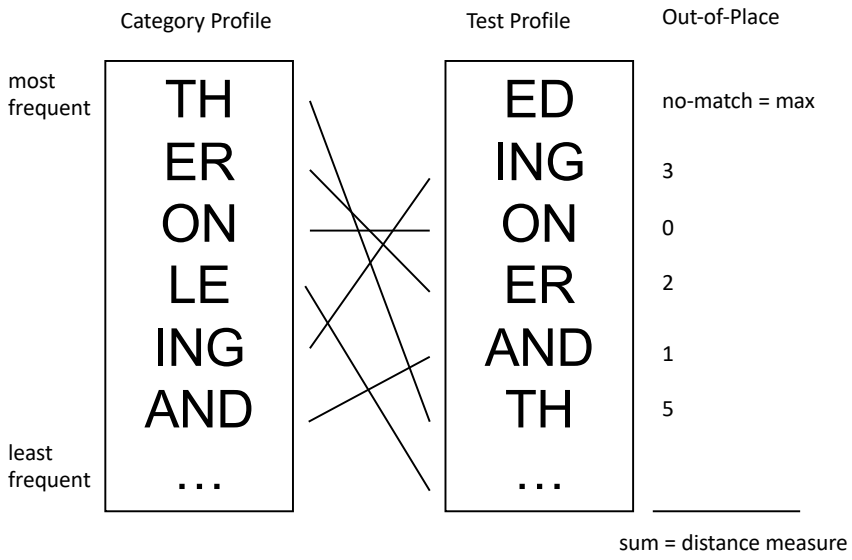

Figure 2.1: Out-of-Place measure calculation between two profiles (adapted from Cavnar and Trenkle (1994)).

Subsequent solutions are mainly derived from training the language model from either character n-gram or byte n-gram on different classifiers such as naive Bayes, SVM and Markov models. `langid.py`, an off-the-shelf LID tool, is trained over a naive Bayes classifier in combination with a multinomial event model and byte n-gram features (Lui and Baldwin, 2012). This tool is originally encapsulated and open sourced via Python package, and also can be utilized in three ways: command-line tool, Python library and web service. As they suggested, importing Python library is the fastest and least resource-consuming way to use the tool as command-line tool is weak at processing large amount data and web service is more preferable by invoking from other programming languages. It should be noted that a pre-trained model is released along with `langid.py`, which is trained on a large amount of corpus in 97 languages from a variety of domains using LD feature selection (Lui and Baldwin, 2011). In comparison with other four standalone LID tools, `langid.py` supports the most languages and generally outperformed them in terms of accuracy and speed in most test sets. Thus, it has been a state-of-the-art in open source LID tools used in industry.

2.2 Learning-based Solutions

In recent years, deep learning (DL) techniques have been extensively developed in both research and industry areas, achieving impressive accomplishment in computer vision (CV) and NLP areas. Over the past five years, more and more researchers have applied learning-based solutions to solve textual language identification.

One of the early works on LID involved with neural networks is done by Simões *et al.* (2014), who constructed a simple neural network (NN) with only one hidden layer. The network takes an input vector containing values obtained from feature extraction, and outputs

a vector with probability values corresponding to each language. Unlike many simply use n-gram features for training, they combined trigram features with alphabet features to get better understanding of the data. The extracted features can be categorized into two levels: one related with the language alphabets, and the other with the character trigram frequency information. After defining 10 classes classifying all the used characters in training set based on morphology as well as Unicode range, the alphabet features are obtained by a list of empirically defined rules, which then can be used directly while training. The extraction of trigram features is more conventional, processed by a ready-made tool the Perl module **Text::Ngram** (`https://metacpan.org/pod/Text::Ngram`), then the obtained values are used for computing their relative frequency. At the end, they merged these two types of features into a big list and computed it with the training data in the form of matrix, resulting in the training matrix with 565 different features for each input.

The implementation of this network is based on the logistic function defined by $g(z)$ in Eq. 2.1, namely playing a role as activation function in each layer.

$$g(z) = \frac{1}{1 + \exp(-z)} \tag{2.1}$$

Backward propagation is also applied for implementing gradient descent algorithm in order to minimize the cost function and obtain the weight matrices. Rather setting a threshold of accuracy on validation set or the cost, they chose to stop training with a fixed number of iterations.

The experimental result shows that their model achieves 96% and 97% of accuracy with 1,500 and 4,000 iterations separately on the test set of 25 languages. Also, the result proves that adding alphabet features do help to improve the performance of the model by reducing the trigram features needed, especially for the four Asian languages Traditional

Chinese, Simplified Chinese, Japanese and Korean. Nevertheless, it should be noted that their solution performs badly on short texts because of the insufficient number of trigrams selected by language, which is similar to the challenge we tackle.

Kocmi and Bojar (2017) employed a RNN based solution, bidirectional RNN (Bi-RNN) with only one single hidden layer of GRU cells, to both monolingual and multilingual language identification tasks. The model takes a fixed window size (normally set to 200) of characters sliding through each text as input and outputs a probability distribution over all supported languages. They originally considered several RNN variants like the original one proposed by Elman (1990), LSTM and GRU. Elman's RNN is not able capture the following information within the fixed window size while processing the current input, which does not fit their setting. Therefore, they turned to a bidirectional RNN, reading the input from both left-to-right and in reverse simultaneously, whose hidden layer h_t at time step t is computed based on the previous state h_{t-1} and the future state h_{t+1} as shown in Eq. 2.2 & 2.3.

$$\overrightarrow{h_t} = tanh(\overrightarrow{W}x_t + \overrightarrow{V}\overrightarrow{h}_{t-1} + \overrightarrow{b_1}) \tag{2.2}$$

$$\overleftarrow{h_t} = tanh(\overleftarrow{W}x_t + \overleftarrow{V}\overleftarrow{h}_{t+1} + \overleftarrow{b_1}) \tag{2.3}$$

where U, V and W are weight matrices, b_1 and b_2 are bias, and the left and right arrows indicate the direction. Eq. 2.4 shows the output y_t is derived from applying softmax function to h_t.

$$y_t = Softmax(\overrightarrow{U}\overrightarrow{h}_t + \overleftarrow{U}\overleftarrow{h}_t + b_2) \tag{2.4}$$

Later on, they replaced the tanh unit with GRU proposed by Cho *et al.* (2014) as tanh is

difficult to train due to its non linearity. Changes to equations above due to replacement can be found in Cho *et al.* (2014), which is quite similar. This model is trained on their own built dataset, as a result, it supports the most languages compared to some famous LID tools such as Langid.py and CID2. Also, it beats the other in identifying short texts, achieving 95.0% accuracy on all languages in the test set and 95.5% on common languages that every model supports.

Similarly, Mathur *et al.* (2017) explored the use of RNNs on LID tasks, and compared it with two typical machine learning models Multinomial Naive Bayes (MNB) and Logistic Regression (LR). The utilization of RNNs makes it possible to learn the structure of a language better to distinguish between the similar ones. Their final model is an ensemble of 5 RNNs, where each one using a different feature set, ranging from character 2-grams to 5-grams as well as word unigrams. Since RNNs have problems of exploding and vanishing gradients while training for long sequences of input, thus, gated recurrent units are adopted for the single hidden layer for each RNN. To construct the ensemble model, rather than manually integrating the models by weights, they trained a LR model to get the final prediction. This LR takes the outputs from the 5 RNNs as features and is tuned by 5 fold cross validation to get the final output. The complete training procedure is shown in Fig. 2.2.

One surprising finding of the results is that an individual RNN cannot even outperform the MNB and LR model with little structure information captured. But they are fought back by the ensemble of 5 RNNs, exceeding 0.6% of accuracy than the MNB and 0.63% than the LR model. Although the ensemble model gains the best result on testing, it does not surpass much than others. The biggest challenge is posed by distinguishing very much similar languages, for instance Bosnian, Croatian and Serbian are all in South Western Slavic

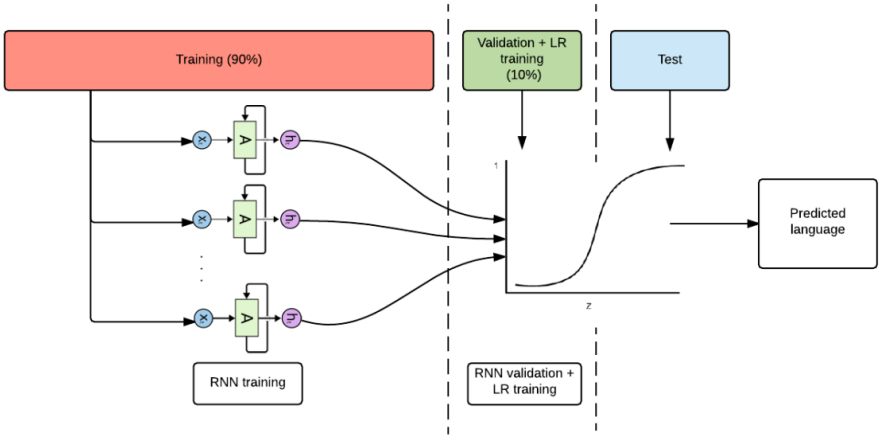

Figure 2.2: Training procedure of the RNNs ensemble (source from Mathur *et al.* (2017)).

language group. These languages share plenty of words in common and cannot be uniquely identified by simply using n-grams features. Thus, building more deeply structured models and designing rule based features could be ways to handle this problem.

Chapter 3

The Proposed Methods

In this chapter, we will first go through the process of building up our new multilingual datasets based on online resources and private experimental data. We then present four different methods for our LID task, which are fastText, BERT, dictionary-based method and the ensemble model in detail. The experimental results and model comparison will be showed in Chapter 4.

3.1 Construction of New Multilingual Datasets

TweetLID is a short text dataset released from a shard task about LID organized by Zubiaga *et al.* (2016). It consists of 34,984 tweets in 10 classes: Spanish, Portuguese, Catalan, English, Galician, Basque, Underterm., Multilingual, Ambiguous and Other. Apparently, there are only 6 languages inside, which is insufficient for our task.

Datasets from Discriminating Similar Languages (DSL) shared task (Tan *et al.*, 2014) are originally aiming at being benchmark for LID in languages with considerably little difference, such as Czech and Slovak, Indonesian and Malay, Brazilian Portuguese and

European Portuguese etc. The limitation is that they do not cover as many as we need over language families.

WiLI-2018 is a new benchmark dataset for monolingual written LID, publicly released by Thoma (2018). It contains 235,000 paragraphs in total, 1000 paragraphs of 235 languages, theoretically appropriate in terms of quantity and diversity. However, the minimum paragraph length is 140 character and the average length is 371 across the whole dataset. We believe it might be too long for our short text classification task.

In order to address the challenge of LID in short texts, we decide to create our own datasets. Collaborating with an AI security company (alias TH), we generated two new multilingual text datasets successively. The first one (named *TRAIN*) contains 17,239,085 records in total across 50 languages while the second one (named *TEST*) has 152,793 records with 22 languages. *TRAIN* and *TEST* were both generated from diverse sources, mainly made up of open source and real-world experimental data. The language range in *TEST* is a subset of that in *TRAIN* and there is no overlap in data between them. All the data are encoded in UTF-8.

The majority of *TRAIN* were extracted from Twitter and Google books Ngrams (4-grams and 5-grams mainly), the rest of it came from experimental data - chat messages and short dialogue records owned by TH for research. Twitter and Google books Ngrams typically contain data dominant in several languages which people speak mostly all over the world, while some rare languages holding large gap in amount. Thus, we set a threshold for the amount of upper bound per language in order to keep the data as balanced as possible. Then we extracted the data and pre-processed following the steps below:

1. Data Cleaning: clean up noise like alias, website addresses and hash tags, and remove duplicates.

2. Build Confidence Level: give each text a confidence level indicating how trustable it is to be a true label. There are three models fastText default, an off-the-shelf LID tool `Langid.py` and Google Translate API used for predicting the most likely label of each text with probability.

3. Blend Confidence Level: after checking the labels predicted by the three models, if all three have the same predictions as the text's original label, we would certainly tag it confidence level 100, otherwise we mark it a specific value ranging from 0 to 100 based on blending the predicted probabilities by our rules.

It is worth noted that some of the data from online sources appear to be mislabeled due to mixed language texts and wrong categorization when publishing, which is the reason why we bring in confidence level. As a result, we obtained a dataset with each text tagged a confidence level $C \in \{00, 10, 25, 50, 75, 90, 100\}$ (00 refers to 0), and a full table about the final distribution of *TRAIN* is shown in Table 3.1. For the sake of different purposes, we truncated the *TRAIN* into smaller subsets by confidence level (e.g. truncate by 00 containing all data with C over 0, which is exactly same as *TRAIN*; truncate by 25 containing data with C higher than 25 and so on). This led to the generation of seven new subsets named as *TRAIN*-C separately. The seven subsets could be used for different purposes, for example, if we want to get the cleanest and most trustable dataset, we would take *TRAIN-100*. Moreover, *TRAIN-00* has the largest amount of data.

The source of data is even more miscellaneous when it comes to *TEST*. The majority is made up of experimental data from TH, but consisting of more distributed data compared to *TRAIN*, and the rest is from Twitter. The subsequent processing steps are similar to that of *TRAIN*. After that, human labeling is also appended to verify the correctness and availability of the data, which ensures the robustness for the following experiments.

Table 3.1 and Table 3.2 show the distributions of *TRAIN-100* and *TEST* we created. It should be noted that the distributions are both not balanced in the two datasets with some languages dominant, especially for the *TRAIN*, for which we adopt macro average to evaluate the results, as described in Chapter 4, in order to prevent bias from evaluation metrics.

Language (code)	**#Records**	**Percentage(%)**
English (en)	1682212	13.833
French (fr)	1234500	10.151
Spanish (es)	1087093	8.939
German (de)	985722	8.106
Russian (ru)	787114	6.473
Chinese (zh)	776704	6.387
Italian (it)	654711	5.384
Thai (th)	555698	4.570
Portuguese (pt)	549058	4.515
Turkish (tr)	498465	4.099
Polish (pl)	472085	3.882
Dutch (nl)	415889	3.420
Japan (ja)	402840	3.313
Korean (ko)	374836	3.082
Arabic (ar)	242015	1.990
Persian (fa)	236621	1.946
Greek (el)	200845	1.652
Swedish (sv)	200068	1.645
Hebrew (he)	189508	1.558
Catalan (ca)	127503	1.048
Finnish (fi)	125490	1.032
Vietnamese (vi)	106204	0.873
Czech (cs)	64285	0.529
Ukrainian (uk)	47426	0.390
Danish (da)	35671	0.293
Hungarian (hu)	26966	0.228
Hindi (hi)	18382	0.151
Norwegian (no)	14922	0.123
Romanian (ro)	13441	0.111
Serbian (sr)	11424	0.094
Indonesian (id)	10318	0.085
Latvian (lv)	5003	0.041
Bulgarian (bg)	4053	0.033
Basque (eu)	3724	0.031

Table 3.1: Distribution of *TRAIN* dataset, sorted in order of descending in Records.

Language (code)	**#Records**	**Percentage(%)**
Spanish (es)	9665	6.326
Polish (pl)	8785	5.750
Russian (ru)	8534	5.585
English (en)	8387	5.489
Dutch (nl)	8089	5.294
Portuguese (pt)	7793	5.100
Japan (ja)	7639	5.000
Finnish (fi)	7466	4.886
Chinese (zh)	7376	4.827
Vietnamese (vi)	7176	4.697
Thai (th)	7065	4.624
German (de)	6774	4.433
Italian (it)	6759	4.423
Korean (ko)	6738	4.410
Swedish (sv)	6544	4.282
Turkish (tr)	6360	4.162
French (fr)	6194	4.054
Arabic (ar)	5816	3.806
Danish (da)	5531	3.620
Norwegian (no)	5512	3.607
Indonesian (id)	5383	3.523
Hindi (hi)	3207	2.099

Table 3.2: Distribution of the *TEST* dataset, sorted in order of descending in Records.

3.2 Learning-based Methods

3.2.1 fastText

fastText is an open source and simple classifier for text classification but also achieves competitive performance in various NLP tasks. It has two highlighted features introduced below.

1. Fast training and running: Joulin *et al.* (2017) carried out experiments showing that fastText is able to process half a million sentences with 312K classes in less than a minute, after trained on over a billion words in less than ten minutes with a standard multi-core CPU. The fast training and running speed is largely owing to its simple architecture showed in Fig. 3.1. As we can see, it is a linear model with rank constraint and there is only one hidden layer in between.

Another improvement on running time is the utilization of hierarchical softmax (Goodman, 2001) and Huffman coding tree (Mikolov *et al.*, 2013b). Generally, while training on large corpora with the number of classes growing dramatically, the computational consumption of the linear classifier becomes expensive. Mathematically, the computational complexity is $O(Kd)$ where K denotes the number of classes and d the dimension of text representation. After apply hierarchical softmax, the computational complexity drops down to $O(dlog_2(K))$ during training. The hierarchical softmax also has advantage on reducing the computational complexity at testing in the way of targeting the leaves with maximal probability when the depth of tree is fixed beforehand.

2. Subword information: Bojanowski *et al.* (2017) proposed an improved word embedding method which can better capture the morphology and internal information of words, inspired by skip-gram model (Mikolov *et al.*, 2013b). Each word w is represented as a

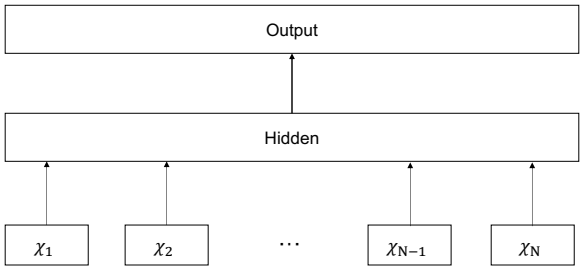

Figure 3.1: This is the model architecture of fastText for a sentence input with character N-gram features $x_1, x_2, ..., x_N$. Then the features are embedded and processed to form the hidden layer.

bag of n-gram characters. In addition, special boundary symbols $<$ and $>$ are added to the beginning and the end of words, indicating their own prefixes and suffixes for distinguish from other words. Also, a word w itself is included in the bag of its n-gram characters. Assume that we have a word $letter$ and set $n = 3$, the word would be represented as follows:

<le, let, ett, tte, ter, er>

and itself

<letter>.

Given a word w, we associate a vector representation v to each n-gram character. A word can be represented as the sum of its vector representations, allowing us to identify the Out-of-Vocabulary (OOV) words.

3.2.2 BERT

Bidirectional Encoder Representations from Transformers (BERT) is an state-of-the-art approach of pre-training language representations, which is able to train a language understanding model on large amount unlabeled corpora. Given the language model trained, we can then use it for some downstream NLP tasks such as text classification, question answering and automatic speech recognition with simple task-specific fine-tuning work.

Unlike word2vec (Mikolov *et al.*, 2013a) and Golve (Pennington *et al.*, 2014), simply using numerical vector to represent a word in the corpus, BERT instead works in a contextual way to get the word representations based on surrounding words. For instance, 'can' in the sentence 'Can you can a can as a canner can a can?' has completely different meanings. The first 'can' is regarded as a modal auxiliary, the second and fourth 'can' regarded as verb and the third and fifth is taken as noun. It does not make sense to just embed a polysemous word by one single vector representation. Moreover, another key innovation of BERT is that it implement a bidirectional learning through a Transformer encoder in pre-training contextual representations. There are some models like ELMo (Peters *et al.*, 2018), ULM-Fit (Howard and Ruder, 2018) and Generative Pre-training (Radford *et al.*, 2018) also able to capture contextual representations during pre-training, however, all of them are either unidirectional or pseudo-bidirectional. This means that they learn the word representations of each word by the words to its left or right, or simply through a combination of language models trained via left-to-right and right-to-left.

BERT's model architecture is composed of a multi-layer bidirectional Transformer encoder and is largely identical to the the original one proposed in Vaswani *et al.* (2017). The framework of Transformer is built upon Sequence-to-Sequence proposed in Sutskever *et al.* (2014), both consisting of an encoder and a decoder, which mainly convert an input

sequence into an arbitrary output sequence. However, the difference is that Transformer relies entirely on attention mechanisms, in contrast to Sequence-to-Sequence models made up of deep neural networks (DNNs) such as Long Short-Term Memory (LSTM), Recurrent Neural Networks (RNN) and Gated Recurrent Units (GRU). Replacing of DNNs by attention mechanisms has been proved to achieve better performance in some sequential problems. The structure of Transformer is shown in Fig. 3.2.

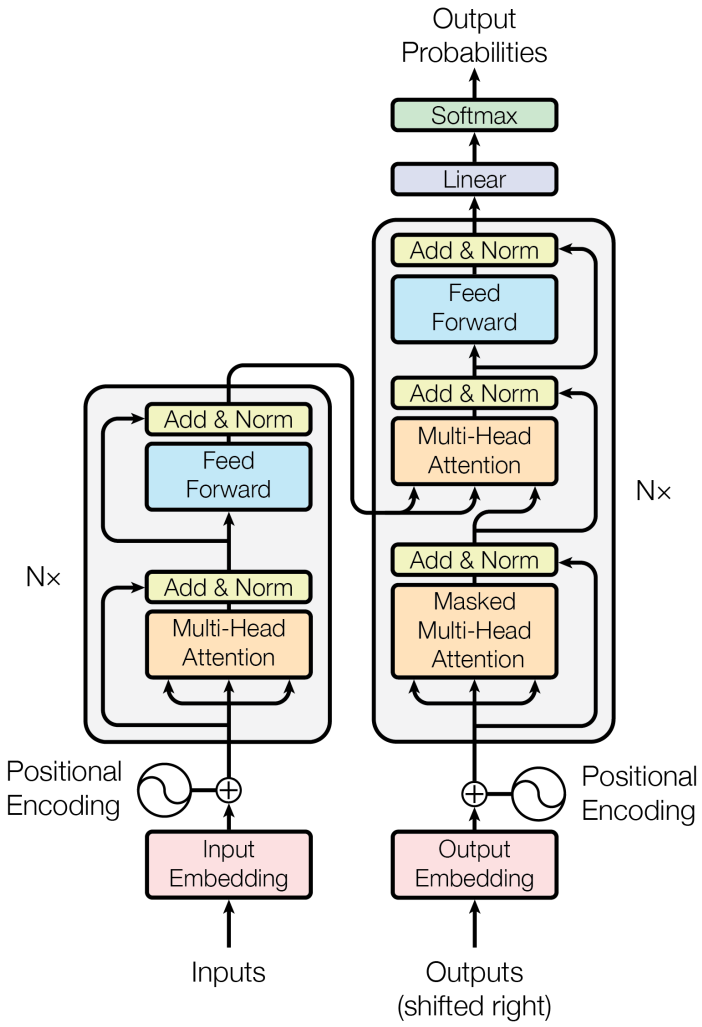

Figure 3.2: The structure of Transformer (source from Vaswani *et al.* (2017)).

The left side is the encoder consisting of N identical layers stacked, where each layer has two sub-layers and is concatenated to the previous one if there exists. More specifically, the output of the previous layer is the input of the latter, and the output of the last layer is the result of the encoder. The right side is the decoder whose structure is similar and also stacked by the same amount of identical layers, but containing one more attention based sub-layer. Multi-head attention and fully connected feed-forward networks are the two innovations of the encoder structure. In multi-head attention, queries (Q), keys (K) and values (V) are projected h times linearly by another attention called scaled dot-product, then the resulting outputs are concatenated and projected again to form the final values. This procedure allows the model to perform parallelly and address the position information from different subspaces. The structure of multi-head attention is shown in Fig. 3.3 and the output of it can be calculated by Eq. 3.1.

$$\begin{aligned} Multi\text{-}Head(Q,K,V) &= Concat(head_1,...,head_h)W^O, \\ \text{where } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V). \end{aligned} \tag{3.1}$$

There are two exact steps to construct a BERT model: pre-training and fine-tuning. Pre-training is the procedure of generating a language model by training on large amount unlabeled data on two different tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). In order to get a bidirectional representation, MLM first mask a certain percentage of the tokens in the input sequence (conventionally 15%), then run the entire sequence through bidirectional Transformer encoder and lastly predict the masked tokens. An example of how MLM works is shown below:

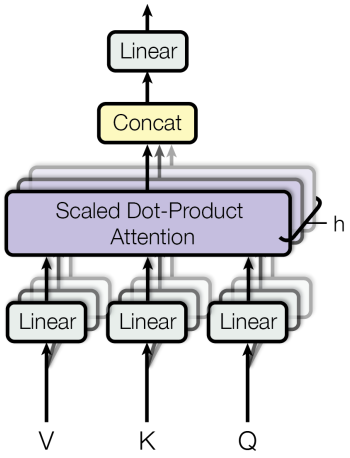

Figure 3.3: The structure of multi-head attention (source from Vaswani *et al.* (2017)).

Masked Language Model (MLM)

Input: The man was $[MASK]_1$. He wanted to buy a $[MASK]_2$ of water.
Labels: $[MASK]_1$ = thirty; $[MASK]_2$ = bottle.

Another task NSP is developed to learn the relationships between sentences from any monolingual corpus, which could not directly obtained by language modeling. Examples of NSP are shown below:

Next Sentence Prediction (NSP)

Ex.1:
Input: The man was thirty. He wanted to buy a bottle of water.
Labels: IsNext.

Ex.2:
Input: The man was thirty. How is the weather today.
Labels: NotNext.

It should be noted that it is fairly expensive to pre-train a language model (generally four days on 4 to 16 Cloud TPUs) from scratch but handy to utilize the pre-trained ones released along with the paper (Devlin *et al.*, 2018) as they are trained for language understanding and compatible with most of the NLP downstream tasks. Thus, we decide to opt for the pre-trained model and fine-tune the output layer on our task with appropriate data structure.

Fine-tuning is relatively straightforward and inexpensive compared to the pre-training step. For our LID task, we first pre-process the task-specific inputs and outputs into acceptable format, load them to BERT and lastly fine-tune all the default parameters passed from pre-trained model.

3.3 Dictionary-based Method

Assuming we have a dictionary containing all the words of languages being used all over the world, then LID problem would be addressed easily and brute-forcedly by simply looking up the input text in the dictionary. Before building up such a huge dictionary, we first need to construct a dictionary for every single language that maps each word to a corresponding language. However, many languages share common words (e.g. Japanese borrows words from Chinese, Indonesia borrows words from English, Danish, Swedish and Norwegian share words since a long time ago as they are geographically aggregated), which pose difficulties to LID. One way to solve this problem is bring in a frequency value to each word per language. We call this idea Look-up Method (LKM) and will introduce it in detail in the remainder of this section.

3.3.1 Construction of Dictionary

We built two dictionary, Dict1 was generated from the word vectors available in Grave *et al.* (2018). We only extracted the words from the word vector for each language, excluding the 300 dimension vectors, where the words aligned in descending frequency order. Each word then was mapped to a frequency value $\mathcal{F}$ given by

$$\mathcal{F} = \frac{1}{\mathcal{R} \times \log(\mathcal{C})}, \tag{3.2}$$

where $\mathcal{R}$ is the row number of the word and $\mathcal{C}$ the total number of words. After that, we assigned a language code indicator (Robin Cover, 2001) to each word in the same language, and each dictionary per language is formed as shown in Fig. 3.4.

```
{..., 'word': (language code, frequency value), ...}.
```

Figure 3.4: Example of the dictionary in a language

Next step is to merge the dictionaries with each other. When the same words with different values (i.e language codes and frequency values) conflicted while integrating into a large combined dictionary, we reformed the structure of key-value pairs, putting all corresponding values together for one single key. After integration, the final dictionary was constructed in the form of:

```
{..., 'word': [(language code 1, frequency value 1),
(language code 2, frequency value 2), ...], ...}.
```

Figure 3.5: Example of the format in Dict1

The second dictionary Dict2 was built by following the similar steps above, but with different source available on website Hermit Dave (2018), which is originally generated from the subtitle corpus OpenSubtitles (2018). The source we utilized were word frequency lists in over 60 languages, where each list contains mappings from words to frequency values indicating the number of occurrences in the corpus. By example in English list, the content is formatted as shown in Fig. 3.6.

We initially extracted 50k words per language in order to have the distribution of languages balanced, which then proved to be insufficient for certain languages. We finally took the whole list from each languages to build Dict2. Note that we did not assign frequency value like Equation 3.2 to words in Dict2, but adopted their original frequency values shown in Fig. 3.6. The final format of Dict2 is exactly same as Dict1 shown in Fig

```
you 28787591
i 27086011
the 22761659
to 17099834
a 14484562
it 13631703
and 10572938
that 10203742
...   ...
```

Figure 3.6: Example of word frequency in English list

3.5.

3.3.2 Algorithm

The algorithm of our LKM on language identification is illustrated in Table 3.3. This method works better in classifying shared words across multiple languages than some traditional dictionary-based method such as the one proposed in Wang *et al.* (2015). For example, original English words 'ok', 'okay' and 'hello' are extensively used in most languages, original Chinese words '今' and '日' are now being used by both Chinese and Japanese. Frequency value is then utilized to solve this issue, distinguishing same words in different languages by the number of occurrences. Thus, the Top 1 language with highest value means the word occurs most frequently in it, which could be consider as the most likely prediction in single label classification. Also, LMK works well on short text. As long as the words captured in dictionary are rich enough, and the degree of purity with each language is high enough, LMK would perform presumably good. The performance of this method is highly relying on the quality of dictionary, where is also the weakness derived from. The input text is required to be cleaned up and tokenized at the beginning,

which shows the limitation of this method. Moreover, abbreviation and colloquialism pose challenge for detecting languages with LKM in such situation

Algorithm of LKM: Dictionary-Based Method

INPUT: A raw text **M**
RETURN: (language code, frequency value, not found ratio).
A tuple containing the detected language, the frequency value and the not found word ratio within Dict2.

STEPS:
0. Initialize the **Dict** which maps a word to a list of corresponding language and frequency;
1. Initialize a mapping **results** ;
2. Initialize a counter **notfound** to be zero ;
3. Tokenize **M** into a sequence of words **words** by space whose length is **words_size**, ignoring punctuation, numbers and symbols;
4. For each **word** in **words** do
    if **word** in **Dict.keys** then
        preds ← **Dict**[word]
        normalize(**preds**)
        for each **pred** in **preds** do
            results[pred[0]].append(pred[1])
    else **notfound** += 1;
5. If **results** is not empty then
    for **lang, score** in **results** do
        results[lang] ← sum(score)**/words_size**
    top_1_lang = *max*(**results**)
6. **return** (top_1_lang, results[top_1_lang], not_found/words_size)

Note:
1. *normalize* function takes a sequence of language and frequency value pairs as input and return a normalized frequent value to each detected language.
2. *max* function returns the language with highest frequency value.

Table 3.3: Algorithm of Dictionary-Based Method on language identification

3.4 Ensemble Model

In this section, we will show how to integrate fastText and LKM introduced above to form a ensemble model which combines the advantages of both models for language identification.

To integrate the two models, we first tried the SVM model available in Chang, Chih-Chung and Lin, Chih-Jen (2011), which turned out to be of low performance as SVM works better for combining more than three models. Later on, we found a traditional way to integrate the models by simply setting prediction threshold and logic gate. The threshold of fastText's prediction probability is set to 0.85 under multiple experiments and 'notfound' ratio is limited to 0 since we want to maximize the confidence we have on LKM's prediction. A clear flow chart of how the ensemble model works is shown in Fig. 3.7.

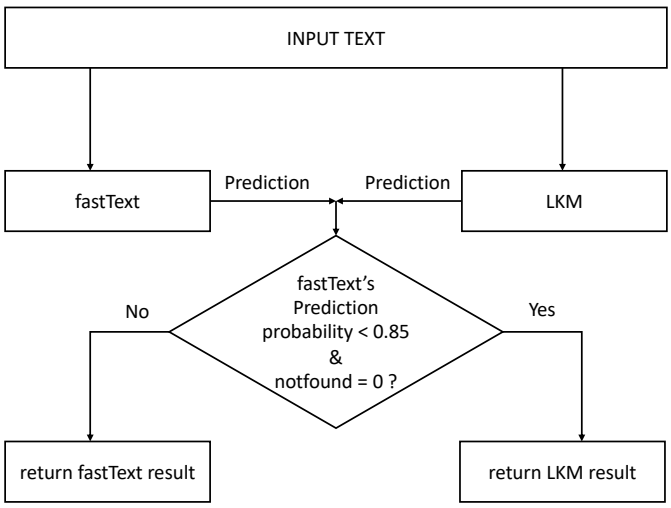

Figure 3.7: Work flow of Ensemble Model integrating fastText and LMK for language identification

Chapter 4

Implementation and Experimental Results

In this section, we will first go through the datasets for training and testing in our experiments, then the implementation details about training. The evaluation metrics used will be introduced as well, followed by the experimental results and comparison of our methods.

4.1 Datasets for Training and Testing

For training and validation, we want to make sure all data in the dataset are 100% truly labeled, thus we take *TRAIN-100* (12,160,796 records) introduced in Chapter 3.1 and divide it into 98% (11,917,580 records) training and 2% (243,216 records) validation set. For testing, a gold test set *TEST* (152,793 records) is used for individual test in order to better assess the generalization capacity of the models. The distributions of *TRAIN-100* and *TEST* are shown in Table 3.1 & 3.2. Also, Table 4.1 clearly demonstrates that the data in these two datasets are both short texts compared with WiLI-2018 whose average length

is 371, which exactly fits our case.

Dataset	#Records	#Labels	Avg. Length	Std.
TRAIN-100	12,160,796	34	50	58
TEST	152,793	22	28	42

Table 4.1: Statistics for *TRAIN-100* and *TEST*.

Each text in the two datasets associated with its corresponding single language code label, which is also considered as ground truth. The language codes adopted as labels are under ISO 639-1 standardized nomenclature (Robin Cover, 2001), where each language is assign a two-letter lowercase abbreviation (e.g 'English' as 'en', 'French' as 'en', 'Chinese' as 'zh' and 'Portuguese' as 'pt'). Particularly, fastText requires the input for training to be formatted in a specific way which is "`__lang.label__ text`" as shown in Table 4.2 below.

Input (Lable + Text)
`__lang.en__ welcome everybody`
`__lang.fr__ Bienvenue a tous`
`__lang.es__ buen clima hoy`
`__lang.it__ Era meglio se non si iniziava sta guerra`
`__lang.tr__ cellat`
`__lang.de__ wie alt bist du`
`__lang.en__ we will meet there`

Table 4.2: Examples of input format for fastText.
(Note: These examples are for explanatory purpose only and do not reflect real data in above datasets)

4.2 Implementation

The fastText related experiments are carried out on an Amazon Web Service (AWS) machine learning instance with 4 vCPUs and 16GB memory. Training and testing on BERT are more computationally-intensive, so they are carried out on Valohai deep learning management platform with 32 vCPUs, 488GB RAM and 8 $\times$ K80 GPUs with 96GB memory.

fastText

fastText is a computationally-friendly model that can be trained on multi-core CPU efficiently. There are two ways to train it, one is using command line tool and the other is by importing its Python library binding the complete C++ source code (Facebook Research, 2019). We train fastText by its Python library in order to have some self-defined functions integrated in order to better illustrate the performance with statistical information and graphs.

We first try the fastText classifier *lid.176.bin* with default parameters setting available on their website Facebook Inc. (2019). After obtaining the result, we find it could be enhanced by customizing the training parameters. Thus, we fine-tune the fastText model by applying a wide range of parameters, while keeping the rest default. The options we use for fine-tuning are shown in Table 4.3, where minn and maxn stands for the minimal and maximal length of character n-grams, dim for the size of word vectors and minCount for the minimal number of word occurrences.

Parameters	Options
learning rate	[0.25, 0.5, 0.8, 0.9]
epoch	[6, 8, 9, 10]
minn	[1, 2, 3]
maxn	[3, 4, 5]
pretrainedVectors	[yes, no]
minCount	[15, 20, 30, 31]
dim	[100, 300]
window size	[3, 4, 5]
loss function	['ns', 'hs', 'sm']

Table 4.3: Parameter options for fine-tuning fastText model.

BERT

As pre-training BERT is expensive in resources and time, we instead use the pre-trained language model *BERT-Base, Multilingual Cased* released by Google Research (2019) for our classification task. For fine-tuning, we initial our hyper-parameters learning rate as 0.00002, batch size of training as 256, number of epoch as 3 and warm-up proportion as 0.1. Without using the default deep learning framework TensorFlow released by Google, we implement the whole experiment in Pytorch version adapted from a third party NLP researchers available on `https://github.com/--huggingface/transformers`.

4.3 Evaluation Metrics

For classification task, we use confusion matrix shown in Table 4.4 containing True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) to evaluate the classification results of each class.

Prediction	Truth: A	Truth: Not A
A	TP	FP
Not A	FN	TN

Table 4.4: Confusion matrix of example class A.

Also, some conventional metrics for classification task - accuracy, precision, recall, F_1-score, micro and macro average are adopted for evaluation. Typically, accuracy refers to how close of a measurement to a true or accepted value and is defined as:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}. \tag{4.1}$$

Precision refers to how reproducible the measurements are and how stable the classifier is:

$$Precision = \frac{TP}{TP + FP}. \tag{4.2}$$

Recall, also know as sensitive reflects how complete the results are:

$$Recall = \frac{TP}{TP + FN}. \tag{4.3}$$

F1-Score is an overall evaluation metric and simply defined as the harmonic mean of precision and recall:

$$F_1\text{-}Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{4.4}$$

Since we deal with a multi-class classification task, thus we evaluate the performance from both micro and macro view. Assume that we have K classes (K language classes)

where $K \in \mathbb{Z}^+$, precision and recall from micro view are defined as follows:

$$Micro\ Avg\ of\ Precision = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K} TP_i + \sum_{i=1}^{K} FP_i}, \tag{4.5}$$

$$Micro\ Avg\ of\ Recall = \frac{\sum_{i=1}^{K} TP_i}{\sum_{i=1}^{K} TP_i + \sum_{i=1}^{K} FN_i}. \tag{4.6}$$

From macro view, precision and recall are defined as:

$$Macro\ Avg\ of\ Precision = \frac{\sum_{i=1}^{K} Precision_i}{K}, \tag{4.7}$$

$$Macro\ Avg\ of\ Recall = \frac{\sum_{i=1}^{K} Recall_i}{K}, \tag{4.8}$$

where $Precision_i$ refers to the precision in class i, $Recall_i$ the recall in class i and $i \in (0, K]$.

4.4 Results and Comparison

Table 4.5 shows a comparison about the performance on validation and test set of the models we experiment with. Apparently, both learning-based methods, fastText and BERT, perform fairly well on validation set, achieving accuracy close to 100% separately. It proves our fine-tuning on fastText and training on BERT does not cause over-fitting. In terms of the performance on our gold test set, one surprising finding is that BERT, a state-of-the-art pre-trained language model in NLP domain, turns out to be the worst model among four. It is even not able to beat the performance of LKM, a dictionary-based method that has little knowledge about text representation and sentence structure. One possible reason is that BERT has little advantage in addressing short text material which does not incorporate enough syntactic and semantic information inside. Compared with BERT, another learning-based method fastText gains 89.6% accuracy and 6% over it. Unlike the bidirectional transformers used in BERT, the meaningful information of the input text, i.e. morphology and semantic information of words are embedding into vector representations in fastText, which may attribute to the better result. It is worth noting that simple structure and traditional word representation approaches seem to work better in the sense of LID in short text.

The best result on gold test set is achieved by the ensemble model of fastText and LKM, which is 89.9% accuracy, 0.3% slightly higher than fastText and 5.1% over LKM. The intention of ensemble model is to combine the advantages of the component models, achieving a performance to be the best, or at least perform as well as the worst one in the ensemble. Therefore, we select the best two performing models to build the ensemble. In addition, it should be noted that fastText and LMK are easier to be integrated together due to the simple structure compared to BERT, which is another reason of opting for them. The

result below has proved our assumption of trying model combination.

Model	**Accuracy**	
	Validation	**Test**
fastText	0.998	0.896
BERT	0.999	0.836
Look-up Method (LKM)	N.A.	0.848
Ensemble Model (EM)	N.A	**0.899**

Table 4.5: Performance in accuracy of various models on validation and test set. The results in this table are calculated under macro metrics and the same for the following results in this section.

The performance of the models on individual language are shown in Table 4.6. Apparently, the ensemble model is dominant in classifying the individual language correctly, followed by fastText, LMK and BERT. Specifically, the ensemble model works fairly well on the 15 out of 22 languages with accuracies above the average 90.0%, and 8 of them are higher than 95% and accuracy in Korean is even close to 100%. It is obvious that the biggest challenge comes from identifying Hindi(hi) and Norwegian(no) with accuracies of 60.0% and 70.8% separately. Looking back to the training and test sets, we notice that Danish(da) and Norwegian(no) share a lot of words in common, because they are geographically-linked and both belong to North Germanic languages. With regard to the worst result of individual language, Hindi(hi), we note that there exist two different scripts of 'hi' in gold test set, Devanagari script and Arabic script, while the latter is the only kind in training set. Thus, there is no way for models to capture semantic information for one script in a language but trained on another script for the same language.

Language	fastText	BERT	LKM	Our EM
ar	0.981	0.957	0.916	**0.983**
da	0.789	0.724	0.773	**0.792**
de	0.920	0.870	0.874	**0.921**
en	0.817	**0.919**	0.866	0.852
es	0.904	0.851	0.871	**0.908**
fi	0.889	0.821	0.850	**0.910**
fr	**0.923**	0.840	0.829	0.915
hi	0.587	**0.606**	0.554	0.600
id	0.791	0.572	0.685	**0.799**
it	0.895	0.851	0.871	**0.903**
ja	**0.933**	0.928	0.908	0.931
ko	**0.997**	0.995	**0.997**	**0.997**
nl	0.925	0.854	0.874	**0.932**
no	**0.722**	0.542	0.561	0.708
pl	**0.963**	0.865	0.876	**0.963**
pt	**0.915**	0.789	0.805	0.903
ru	0.977	0.967	0.945	**0.979**
sv	0.871	0.768	0.853	**0.876**
th	0.999	0.992	**1.000**	0.999
tr	**0.967**	0.798	0.846	0.965
vi	**0.959**	0.887	0.912	0.955
zh	0.981	0.986	**0.989**	0.984

Table 4.6: Performance of individual language in accuracy of various models.

Comparion with Other Systems

Apart from the results above, we also evaluate the performance of our EM by comparing its result with other state-of-the-art LID solutions on the same test set, such as Google Translate API (`https://cloud.google.com/translate/`) and an off-the-shelf tool Langid.py mentioned in Chapter 2.

Google Translate is configured a feature of language detection for 104 languages so far, while Langid.py supports 97 languages. It is surely fair to conduct this comparison on gold test set because the languages in it are both supported by Google Translate and Langid.py. Also, we do not employ any changes to these solutions as they are claimed to be ready for off-the-shelf use.

Table 4.7 shows the comparison result, and obviously, our EM outperforms the other two advanced solutions by 3.7% and 26.1% separately. More specifically, Table 4.8 demonstrates the detailed performance on individual language of the them. The accuracy of Langid.py is almost inferior to the other two across the board, except for Chinese(zh) where it is equal to Google Translate. The performance of EM and Google keep competitive, however, EM is still dominant in detecting the languages correctly for most of them. It is notable that Google is relatively good at identifying languages with shared words such as 'da', 'no' and 'sv', achieving better results compared with EM. Also, the accuracy of 'hi' is improved to 81.4% by Google, which is 21.4% over than that of EM.

Solution	**Accuracy**
Our EM	89.9%
Google Translate API	86.2%
langid.py	63.8%

Table 4.7: Comparison of benchmark results with other systems in decreasing order.

Language	**Our EM**	**Google Translate API**	**Langid.py**
ar	98.3%	90.9%	79.9%
da	79.2%	77.5%	48.3%
de	92.1%	88.7%	71.5%
en	85.2%	86.7%	76.8%
es	90.8%	83.1%	57.4%
fi	91.0%	83.9%	61.8%
fr	91.5%	85.4%	58.9%
hi	60.0%	81.4%	39.5%
id	79.9%	60.5%	21.6%
it	90.3%	86.7%	57.6%
ja	93.1%	95.4%	88.4%
ko	99.7%	99.9%	96.8%
nl	93.2%	88.8%	61.8%
no	70.8%	78.0%	29.1%
pl	96.3%	85.9%	70.2%
pt	90.3%	83.3%	44.8%
ru	97.9%	79.8%	55.7%
sv	87.6%	85.3%	53.7%
th	99.9%	100.0%	99.7%
tr	96.5%	81.4%	52.1%
vi	95.5%	95.7%	81.5%
zh	98.4%	97.1%	97.1%

Table 4.8: Performance of individual language in accuracy of benchmark results.

Chapter 5

Conclusion and Future Work

In this work, we have proposed four language identification methods on short textual data, including two learning based methods fateText and BERT, a traditional dictionary based method LKM and an ensemble model. This ensemble model is composed of fastText and LKM, and is hard integrated using logic gates and prediction thresholds. The experimental results show that the ensemble model achieves about 90% accuracy, outperforming the other three even the state-of-the-art pre-training language model on gold test set created by us. Moreover, it beats the performance of Google Translate API and an industry leading tool Langid.py by 3.7% and 26.1% separately, showing its competitiveness to state-of-the-art solutions on this task.

However, there is still much space for it to get improved. One of its component, LKM, is proved to be low accurate in predicting some languages from the results. This can be improved by adding more words into its dictionary. Also, LKM requires the input to be tokenized, restricting its application into a wide range of user cases. Moreover, the size of the dictionary in LKM would explode once we increase the supporting languages largely. Accordingly, We need to rethink the structure of storing the word-frquency pairs, and come

up with an efficient way to decide the amount of words retained.

Unlike language identification on long texts considered as a 'solved' task, the task on short texts is still a challenging problem in NLP. Some issues arise from it happening in our experiment. Specifically, countries in northern Europe are geographically-linked and share a number of words in common, posing a big challenge for our model to identify them accurately. This is more particularly difficult when it comes to short texts that provide little syntactic and semantic information. In fact, our LKM has already been able to distinguish languages with same words somehow by its built-in frequency values. And we also notice expanding the corpus of training for these languages does not help much, because no unique information of the shared words would be detected during training. Thus, we think further improvement can be relied on appending rule based or domain based features by languages specialists or native speakers.

Another challenge related to Hindi(hi) with worst accuracy of 60% comes from the generic nature of our experiment design. We aim to evaluate the generality of models by training and testing them on different distribution datasets without data overlapping, which causes the problem of different scripts of a language individually appearing in either training or test set. One simple solution to it is enlarging the corpus of the missing script in LKM's dictionary.

Bibliography

Baldwin, T. and Lui, M. (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*, pages 229–237, Los Angeles, California. Association for Computational Linguistics.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, **5**, 135–146.

Brodić, D., Amelio, A., and Milivojević, Z. N. (2017). An image texture analysis method for minority language identification. In V. E. Brimkov and R. P. Barneva, editors, *Combinatorial Image Analysis*, pages 280–293, Cham. Springer International Publishing.

Carter, S., Weerkamp, W., and Tsagkias, M. (2013). Microblog language identification: overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, **47**(1), 195–215.

Cavnar, W. B. and Trenkle, J. M. (1994). N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175.

Chang, Chih-Chung and Lin, Chih-Jen (2011). Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, **2**(3), 27:1–27:27.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dunning, T. (1994). Statistical identification of language. Technical report.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, **14**, 179–211.

Facebook Inc. (2019). Models for language identification. `https://fasttext.cc/docs/en/language-identification.html`. Retrieved: Oct., 2019.

Facebook Research (2019). Library for fast text representation and classification. `https://github.com/facebookresearch/fastText`. Retrieved: Oct., 2019.

Friedman, J. H. (1996). Another approach to polychotomous classification. Technical report, Department of Statistics, Stanford University.

Goodman, J. (2001). Classes for fast maximum entropy training. *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.01CH37221)*, **1**, 561–564 vol.1.

Google Research (2019). Library for bert. https://github.com/google-research/bert. Retrieved: Nov., 2019.

Gottron, T. and Lipka, N. (2010). A comparison of language identification approaches on short, query-style texts. In C. Gurrin, Y. He, G. Kazai, U. Kruschwitz, S. Little, T. Roelleke, S. Rüger, and K. van Rijsbergen, editors, *Advances in Information Retrieval*, pages 611–614, Berlin, Heidelberg. Springer Berlin Heidelberg.

Grave, E., Bojanowski, P., Gupta, P., Joulin, A., and Mikolov, T. (2018). Learning word vectors for 157 languages. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*.

Grefenstette, G. (1995). Comparing two language identification schemes. In *Proceedings of JADT*, volume 95.

Grothe, L., Luca, E. W. D., and Nürnberger, A. (2008). A comparative study on language identification methods. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco. European Language Resources Association (ELRA).

Hermit Dave (2018). Frequency words list. https://github.com/hermitdave/FrequencyWords. Retrieved: Nov., 2019.

Howard, J. and Ruder, S. (2018). Universal Language Model Fine-tuning for Text Classification. *arXiv e-prints*, page arXiv:1801.06146.

Hughes, B., Baldwin, T., Bird, S., Nicholson, J., and MacKinlay, A. (2006). Reconsidering

language identification for written language resources. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy. European Language Resources Association (ELRA).

Joulin, A., Grave, E., Bojanowski, P., and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*.

Knerr, S., Personnaz, L., and Dreyfus, G. (1990). Single-layer learning revisited: a stepwise procedure for building and training a neural network. In F. F. Soulié and J. Hérault, editors, *Neurocomputing*, pages 41–50, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kocmi, T. and Bojar, O. (2017). LanideNN: Multilingual language identification on character window. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 927–936, Valencia, Spain. Association for Computational Linguistics.

Ludovik, Y., Zacharski, R., and Cowie, J. R. (1999). Language recognition for mono-and multi-lingual documents.

Lui, M. and Baldwin, T. (2011). Cross-domain feature selection for language identification. In *Proceedings of 5th international joint conference on natural language processing*, pages 553–561.

Lui, M. and Baldwin, T. (2012). langid.py: An off-the-shelf language identification tool. In *Proceedings of the ACL 2012 System Demonstrations*, pages 25–30, Jeju Island, Korea. Association for Computational Linguistics.

Lui, M., Lau, J. H., and Baldwin, T. (2014). Automatic detection and language identification of multilingual documents. *Transactions of the Association for Computational Linguistics*, **2**, 27–40.

Mathur, P., Misra, A., and Budur, E. (2017). Lide: Language identification from text documents. *arXiv preprint arXiv:1701.03682*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a). Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013b). Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781.

OpenSubtitles (2018). Opensubtitles. `http://www.opensubtitles.org/`. Retrieved: Oct., 2019.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proc. of NAACL*.

Prager, J. M. (1999). Linguini: Language identification for multilingual documents. *Journal of Management Information Systems*, **16**(3), 71–101.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*.

Robin Cover (2001). Code for the representation of the names of languages. `http://xml.coverpages.org/iso639a.html`. Retrieved: Oct., 2019.

Simões, A., Almeida, J. J., and Byers, S. D. (2014). Language identification: a neural network approach. In *3rd Symposium on Languages, Applications and Technologies*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Souter, C. *et al.* (1994). Natural language identification using corpus-based models. *HERMES-Journal of Language and Communication in Business*, (13), 183–203.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Tan, L., Zampieri, M., Ljubešic, N., and Tiedemann, J. (2014). Merging comparable data sources for the discrimination of similar languages: The dsl corpus collection. In *Proceedings of the 7th Workshop on Building and Using Comparable Corpora (BUCC)*, pages 11–15, Reykjavik, Iceland.

Thoma, M. (2018). The WiLI benchmark dataset for written language identification. *arXiv e-prints*, page arXiv:1801.07779.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Wang, P., Bojja, N., and Kannan, S. (2015). A language detection system for short chats in mobile games. In *Proceedings of the third International Workshop on Natural Language Processing for Social Media*, pages 20–28, Denver, Colorado. Association for Computational Linguistics.

Zubiaga, A., Vicente, I. S., Gamallo, P., Pichel, J. R., Alegria, I., Aranberri, N., Ezeiza, A., and Fresno, V. (2016). Tweetlid: a benchmark for tweet language identification. *Language Resources and Evaluation*, **50**(4), 729–766.