A Language Detection System for Short Chats in Mobile Games

Pidong Wang **Nikhil Bojja** **Shivasankari Kannan**

Machine Zone Inc.
2225 East Bayshore Road, Suite 200
Palo Alto, CA 94303, USA

{pwang,nbojja,skannan}@machinezone.com

Abstract

Machine Translation system accuracies are often brought down due to inaccurate Language Detection (LD) of input phrases. The Language detection accuracy is further affected when the inputs are short and contain ungrammatical phrases, especially in a multilingual mobile game setting. Chat messages in mobile games are often short as they are typed on mobile devices and contain slang as a common communication preference. Previous work has shown that LD systems have a drop in accuracy when the inputs are short messages instead of long ones. This paper targets LD for short chat messages in mobile games. We propose a novel LD system which integrates text-based and user-based methods to achieve significantly better performance over current state-of-the-art LD systems.

1 Introduction

With the growth of social media, a huge amount of social media texts have become ubiquitous, e.g., Twitter messages, Facebook updates, game chat messages, etc. Due to their importance, Natural Language Processing (NLP) applications have been applied to social media texts, e.g., Liu et al. (2011) and Ritter et al. (2011) recognized named entities in Twitter messages, and Foster et al. (2011) investigated Part-Of-Speech tagging and parsing of Twitter messages. As the phenomenon is prevelant across the globe, social media texts are usually multilingual, while most of the NLP applications are language-specific. We usually have to know the language of a given message, in order to process the message using appropriate NLP applications. Accuracy of Language Detection (LD) is thus highly critical for subsequent NLP applications.

LD on long messages is widely considered a solved problem as its accuracy is often found to be high with latest methods (Ahmed et al., 2004; Hughes et al., 2006; Grothe et al., 2008). However, more and more researchers have recently noted that LD on short messages is very difficult. E.g, Baldwin and Lui (2010) found LD became increasingly difficult as we reduced the length of documents, and increased the number of languages. Carter et al. (2013) found LD of microblogs was challenging for state-of-the-art LD methods. Moreover, LD studies mostly focus on Twitter messages, as Twitter provides an API for researchers to crawl public Twitter messages, while no research is done on game chat messages, which in itself contains language that has quite a bit more slang than Twitter messages.

In this paper, we propose a novel LD system for chat messages in a mobile game which has a built-in chat translation system. The translation system helps players speaking different languages chat with each other. The LD system is used to detect language of chat messages such that the chat translation system could know which language a message should be translated from. Chat messages in mobile games are different from other social media texts, because it is inconvenient to type on mobile devices leading to an increased misspelling rate, and game chats tend to be much shorter than Twitter messages (see Section 3). Our work is further more challenging, as we are detecting 27 languages.

Our contributions are as follows: (a) as far as we know, this work is the first LD work on game chat messages, so our work may pave the way to NLP research on game chat messages which are a new kind of social media texts; (b) our work is also the first approach to apply user language profiles to the LD of game chat messages; (c) we have shown that LD of game chats is very difficult and also propose a novel LD system integrating both text-based and user-based methods to achieve significantly better performance over the current state-of-the-art LD systems.

2 Related Work

Language Detection (LD), or Language Identification (LI), has been extensively studied in previous work. One famous method is character n-gram-based approach (Cavnar and Trenkle, 1994) which was based on calculating and comparing profiles of n-gram frequencies via "Out Of Place" (OOP) distance, a ranking-based distance. The approach first computed a profile for each language in a multilingual training set. Given a test document, the approach computed a profile that was then compared to each language profile obtained from the training set. The document was detected as a language which had the smallest distance to the document's profile. This approach achieved a 99.8% accuracy on Usenet newsgroup articles. One of the disadvantages of (Cavnar and Trenkle, 1994) is that it requires the input to be tokenized. Another similar approach was done by Dunning (1994) who used byte n-grams instead of character n-grams, avoiding the tokenization problem. This approach achieved 99.9% accuracy on documents longer than 500 bytes.

Recently, many researchers have noticed the difficulty in LD for short documents/messages. For example, Baldwin and Lui (2010) presented a detailed investigation of what approaches were the best in varied conditions, and found that LD became more and more difficult when we increased the number of languages, reduced the size of training data and reduced the length of documents. Vatanen et al. (2010) investigated a LD task where the test samples had only 5-21 characters. The authors compared two approaches: one was a naive Bayes classifier based on character n-gram models, and the other was the OOP method of Cavnar and Trenkle (1994). To improve LD on short and ill-written texts, Tromp and Pechenizkiy (2011) proposed a graph-based n-gram approach (LIGA) which performed better than the character n-gram approach of Cavnar and Trenkle (1994) on Twitter messages. Based on LIGA, Vogel and Tresner-Kirsch (2012) further proposed some linguitistically-motivated changes to LIGA, achieving an accuracy of 99.8% on Twitter messages in 6 European languages, while the accuracy of LIGA was 97.9% on the same test set. Bergsma et al. (2012) focused on LD on short, informal texts in resource-poor languages, annotating and releasing a large collection of Twitter messages in 9 languages using 3 scripts: Cyrillic, Arabic and Devanagari. The authors also presented two LD systems which achieved very high accuracy on Twitter messages.

All the previous work focused on LD using text features. In contrast, our work utilizes user language profile as well, i.e., language distribution of messages sent by a user, to further improve LD on very short messages. The most relevant work was done by Carter et al. (2013). In order to improve LD of Twitter messages, the authors used post-dependent features (i.e., features from only texts) together with several post-independent features: the language profile of a blogger, the content of an attached hyperlink, the language profile of a tag, and the language of the original post. However, we could not directly apply their approach to our context, chat messages in games which are different from Twitter messages, e.g., chat messages have no hyperlink, no tag, etc. Furthermore, game chats are often much shorter than Twitter messages, so LD of game chats is much more challenging (Section 3).

Another relevant line of research is on LD for search engine queries in the context of Cross Language Information Retrieval (CLIR), as the queries are usually relatively short like game chats. Ceylan and Kim (2009) first generated a LD data set of search engine queries extracted from click-through logs of Yahoo! Search Engine, and then trained decision tree classifiers for LD. Moreover, the authors also experimented with a non-text feature, the language information of the country from which a user makes a search query. Gottron and Lipka (2010) used news headlines as short, query-style texts on which several typical LD approaches had been evaluated. In their experiments, the naive Bayes classifier with character n-gram features performed best among others. Nevertheless, search engine queries are different from our focus, game chats, in the sense that search queries are usually well-written words/phrases, while game chats could be ill-written, short phrases/sentences.

3 Chat Messages of Mobile Games

To better understand the differences between game chat messages and Twitter messages, we have crawled 2,308,264 Twitter messages using a Java implementation[1] of Twitter's stream API[2]. On average, each message has 73.51 characters. On the other hand, we have obtained 745,635,448 game chat messages from a chat log database of a Massively Multiplayer Online Role Playing mobile Game (MMORPG). Each game chat message has 34.43 characters on average.

From these statistics, we could see that game chat messages are about two times shorter than Twitter messages, despite the different language distributions of the two message sets.

4 Methods

In this section, we will first describe how we make a multilingual data set for LD based on a chat log database of a mobile game. We then present a novel approach to LD for game chat messages. Generally, our approach has two steps: the first step uses an alphabet-based LD method, and the second step uses a linear model (Fan et al., 2008) to integrate 3 methods together: a byte n-gram-based method (Lui and Baldwin, 2012), a dictionary-based method, and a method based on user language profiles. The alphabet-based LD method will be introduced, followed by the 3 methods. We then present our approach by explaining how we integrate the 4 methods together.

4.1 Game Chat Data Collection

In this subsection, we will describe how we make a multilingual data set of game chat messages, based on a chat log database of a mobile game. All the data are encoded in UTF-8.

Generally, a chat log database of a mobile game is accessible. The database contains many fields for a message. Among the fields, related ones to our work are the string of a message, a unique identifier for each user (user id) for the message sender, and the language of the last keyboard used to enter the message. What we want to make is a data set containing many chat messages, for each of which we need its true language and user id.

An important question to answer at this stage is whether we could rely on the keyboard language to find the true language for a message. The answer is no. There are two main reasons for this. The first one is that users might use a keyboard to input a message in a language which is different from the language of the keyboard, e.g., a French user might use English keyboard to input a French message to avoid the delay caused by changing keyboards. The other reason is that users tend to use special keyboards on mobile devices, e.g., a user could input an English message with an English keyboard and then an Emoji[3] with an Emoji keyboard, in which case the log database only records the last keyboard, i.e., the Emoji keyboard.

Motivated by Ceylan and Kim (2009) who generated a LD data set of search engine queries extracted from click-through logs of Yahoo! Search Engine, we could also use the chat log database to make a LD data set. More specifically, we first sample our chat log database to get a raw data set containing messages written using different keyboards according to the keyboard language field. For each message in the raw data set, the LD API of the Microsoft Translator[4] is used to detect the language of the message. If the detected language matches the keyboard language field, we consider the message as a valid message in the final LD data set.

4.2 ALPHA: Alphabet-Based LD

The most straight-forward way to do LD is to count the number of characters of each language, given a message, then picking the language with the highest number of characters. We call this method alphabet-based language detection whose algorithm is shown in Algorithm 1. We use a third-party library which

[1]`http://twitter4j.org/en/code-examples.html#streaming`

[2]`https://dev.twitter.com/streaming/overview`

[3]`http://en.wikipedia.org/wiki/Emoji`

[4]`http://msdn.microsoft.com/en-us/library/ff512411.aspx`

could return all the characters used by a given language.

Algorithm 1 Alphabet-Based Language Detection

```
INPUT: a raw message M whose length is N
RETURN: the detected language for M
1: initialize a map char2langList which maps a character to a list of languages;
2: initialize a map lang2count which maps a language to the count of characters of the language in M;
3: for $i \leftarrow 0$ to N-1 do
4:     for each lang in char2langList[M[i]] do
5:         lang2count[lang] ← lang2count[lang] + 1;
6: return the language in lang2count with the highest count;
```

This method is effective when distinguishing languages written in different scripts, e.g., Chinese and English. However, it is not good at distinguishing languages using similar scripts, e.g., languages using the Latin script. Thus, to achieve a good performance, this method should be used together with other methods, e.g., we could use this method to detect languages using almost separate scripts, e.g., Thai, Chinese, Japanese, Korean, etc. and then use other methods to detect other languages. Please note that here "almost separate scripts" depends on the target language set we want to detect, e.g., if the set contains Russian and Ukrainian both of which use the Cyrillic script, we'd better not use the alphabet-based LD method to detect Russian or Ukrainian, while if the set only contains Russian without Ukrainian, we could detect Russian with the method.

Another issue with this method is the situation that multiple languages have the same highest count. Our solution is to set a priority list of languages according to the language frequencies in the game and language-specific knowledge, and we choose the first language in the list with the highest count of characters as the detected language.

4.3 LANGID: Byte N-Gram-Based LD

Our LD system uses a byte n-gram-based LD approach (Lui and Baldwin, 2012). This approach essentially uses a naive Bayes classifier with byte n-gram features.

Lui and Baldwin (2012) have released an off-the-shelf LD tool written in Python as an implementation of the approach. We have rewritten the tool in C++ to get a higher processing speed. A pre-trained model is released with the tool, and was trained on a large amount of multilingual texts from various domains (Lui and Baldwin, 2011) in 97 languages. The tool also provides a way to limit the number of languages to a subset of the 97 languages, to achieve a higher accuracy and speed. Given an input, the tool has an API to normalize confidence scores for each language to probability values.

4.4 DICT: Dictionary-Based LD

Assuming words in an input message are space-delimited, we could count the number of words in each language, then picking the language with the highest number of words as the detected language. We call this method dictionary-based language detection whose algorithm is shown in Algorithm 2.

Algorithm 2 Dictionary-Based Language Detection

```
INPUT: a raw message M
RETURN: the detected language for M
1: tokenize M into a sequence of words WORDS whose length is N, ignoring punctuation;
2: initialize a map word2langList which maps a word to a list of languages;
3: initialize a map lang2count which maps a language to the count of words of the language in WORDS;
4: for $i \leftarrow 0$ to N-1 do
5:     for each lang in word2langList[WORDS[i]] do
6:         lang2count[lang] ← lang2count[lang] + 1;
7: return the language in lang2count with the highest count;
```

The advantage of this method is that it works well on short messages, even if the input message is only one word, while its disadvantage is from its assumption, i.e., the words of the input message should be space-delimited, which limits the applicability of this method. For example, without knowing an input message is Chinese, the input message cannot be tokenized into words properly, while knowing the language of the input message is just the job of LD. Furthermore, as we are dealing with game chat messages, users could use informal words, e.g., "u" instead of "you", "gtg" instead of "got to go", etc. which also pose difficulties for the dictionaries used in this method. To overcome this problem, e.g., we could use methods like (Liu et al., 2012) to extend our dictionaries to include informal words and slang terms. Another issue with the method is that multiple languages could have the same word, e.g., for a message containing only one word which occurs in two languages, we could also set a language priority list to solve this problem as in Section 4.2.

4.5 Profile: User Language Profile

As can be seen from Section 3, game chat messages are often very short. LD methods relying on only text-based features would perform poorly on game chats. In this subsection, we will introduce a novel method to LD of game chat messages: user language profiles. A language profile for a user is a vector of real numbers each of which represents the probability of sending a message in a particular language. The size of the vector is the same for all the users, i.e., the number of languages supported by the game, though most users only speak one or two languages. In order to build the language profiles, ideally, we should have many human annotators to annotate all the chat messages sent in the game, but it is impractical. We thus have to choose an automatic LD system to detect the language of each message sent by a user. As a result, we obtain a vector of the count of messages written in each language. We then normalize the counts into probabilities, getting a vector of probabilities as the language profile for the user. The Langid system (Section 4.3) is used here to build language profiles. Of course, the LD system might make errors, especially on short messages. However, the experimental results (Section 5.3) confirm this way of building language profiles is effective.

For a new user, the probabilities in the language profile are all 0, meaning we do not know what language the new user will use. If we use Profile individually, the first language in its language priority list is chosen.

4.6 Comb: Combined System

In this subsection, we will show how we integrate all the methods mentioned in this section together to make a high-performance LD system for chat messages sent in mobile games.

Work Flow: According to the characteristics of the methods mentioned in this section, our system has two phases: Phase 1 uses the Alpha LD (Section 4.2) to detect languages using "separate" scripts; Phase 2 uses a linear model to combine the byte n-gram-based method (Section 4.3), the dictionary-based method (Section 4.4) and the user language profile (Section 4.5) together to detect the rest of languages in the target language set. The work flow of the combined LD system is presented in Figure 1.

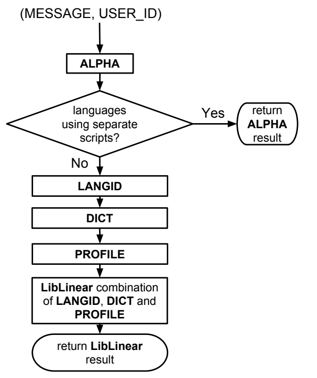

Figure 1: Work flow of the combined LD system.

LibLinear: A linear model is used to combine the three methods which are respectively presented in Section 4.3, 4.4, and 4.5. More precisely, we use the linear support vector machines in LibLinear (Fan et al., 2008) as the linear model. LibLinear is an open source library which is very efficient for large-scale linear classification. We have also tried the SVM model with linear kernel in LibSVM (Chang and Lin, 2011) instead of LibLinear, but LibSVM is much slower than LibLinear with similar accuracies. We thus choose LibLinear finally.

E.g., if the game language set is {Chinese, English, French, Thai}, in Phase 1, the Alpha method detects the 4 languages. If the result is in {Chinese, Thai}, we stop and return the result. Otherwise, English and French are detected in Phase 2. The input feature vector of LibLinear is a concatenation of the normalized output vectors from Langid, Dict, and Profile. Each of the output vectors has 2 real numbers indicating the probability of being English or French. The output vector of Dict may be shorter than that of the other two, when Dict is not applicable to some languages, e.g., a lack of dictionaries, or words which are not space-delimited.

5 Experiments

5.1 Evaluation Corpora

As far as we know, most previous LD work on short messages (Tromp and Pechenizkiy, 2011; Vogel and Tresner-Kirsch, 2012) focused on Twitter messages, and no previous work explored LD for game chat messages. We thus create a LD data set containing multilingual chat messages sent in mobile games with the method described in Section 4.1.

We first create a data set containing chat messages

Languages	Data set statistics (#lines / #characters)					
	TRAIN	**LEN1**	**LEN2**	**LEN3**	**LEN4**	**FULL**
Arabic	17153 / 444779	634 / 2834	984 / 8845	1000 / 13164	1000 / 16667	1000 / 25796
Catalan	8585 / 207730	452 / 2096	838 / 6968	918 / 11196	980 / 16143	1000 / 24190
Chinese	10705 / 110612	695 / 1379	992 / 3784	1000 / 5334	1000 / 6526	1000 / 10620
Czech	7612 / 259388	645 / 3059	965 / 8544	998 / 13336	1000 / 17347	1000 / 33426
Danish	11031 / 554513	367 / 1690	804 / 6992	969 / 12619	996 / 17397	1000 / 50660
Dutch	1201 / 52307	457 / 2206	891 / 8027	985 / 13576	997 / 18768	1000 / 44200
English	68035 / 3134196	473 / 2171	876 / 7736	985 / 13471	998 / 18692	1000 / 46310
Finnish	4003 / 151253	631 / 3286	949 / 9713	981 / 15171	996 / 20196	1000 / 36283
French	27555 / 1197251	416 / 1875	853 / 7024	982 / 12233	998 / 17021	1000 / 43362
German	10978 / 448484	509 / 2441	950 / 8681	997 / 14052	999 / 19214	1000 / 41333
Greek	18234 / 801094	517 / 2845	956 / 9498	999 / 14454	1000 / 18665	1000 / 43668
Hebrew	18553 / 512033	492 / 2115	911 / 7614	996 / 12363	1000 / 15945	1000 / 28842
Hungarian	6876 / 236017	634 / 3285	965 / 9503	997 / 14357	1000 / 18590	1000 / 32241
Indonesian	10285 / 380596	664 / 3309	987 / 9548	1000 / 14902	1000 / 19735	1000 / 35899
Italian	5528 / 256781	539 / 2790	958 / 9066	996 / 14509	998 / 19801	1000 / 46734
Japanese	12531 / 249595	718 / 1414	923 / 3563	976 / 5407	988 / 6938	1000 / 19193
Korean	16546 / 285583	615 / 1180	948 / 3518	986 / 5446	999 / 7152	1000 / 17464
Malay	8899 / 277092	643 / 3075	968 / 9039	993 / 14171	999 / 19154	1000 / 30792
Norwegian	7308 / 316033	419 / 1947	812 / 7031	964 / 12601	997 / 17406	1000 / 43412
Polish	382 / 16481	312 / 1563	483 / 4650	495 / 7587	496 / 10303	500 / 21699
Portuguese	10493 / 414901	549 / 2827	944 / 8794	994 / 14449	1000 / 19554	1000 / 39585
Romanian	9823 / 301461	543 / 2502	929 / 7969	990 / 12894	1000 / 17072	1000 / 30832
Russian	14060 / 592906	484 / 2589	833 / 7866	977 / 14091	998 / 19734	1000 / 41726
Slovak	7482 / 320377	551 / 2757	929 / 8313	994 / 13625	1000 / 18670	1000 / 41426
Spanish	4785 / 237427	477 / 2503	915 / 8322	992 / 13829	998 / 18995	1000 / 49421
Swedish	9044 / 410050	462 / 2243	888 / 7875	987 / 13247	1000 / 17760	1000 / 44829
Turkish	614 / 29611	615 / 3372	976 / 10771	1000 / 17285	1000 / 23733	1000 / 49008

Table 1: Statistics of the LD data sets created from game chat messages.

in 27 languages supported by the game, then splitting the messages for each language into a training set (named **TRAIN**) and a test set (named **FULL**). As our focus is on short messages, we also generate four other test sets based on **FULL**. We have truncated each message in **FULL** to retain the first n tokens[5], thus generating 4 new test sets named as **LEN**n where $n \in \{1, 2, 3, 4\}$. In **LEN**n, only unique messages are retained based on only texts, e.g., if we have (text="thx tom", userid="123", lang="en") and (text="thx boss", userid="456", lang="en") in **FULL**, we only keep one message (text="thx", userid="123", lang="en") generated from the two messages in the data set **LEN1**. The statistics of the resulted data sets are shown in Table 1.

Following Carter et al. (2013), we also use accuracy, i.e., the percentage of messages whose language is detected correctly, to evaluate the effect of LD.

5.2 Systems

We compare our proposed LD system (COMB of Section 4.6) against three baseline methods:
(1) LANGID: uses the byte n-gram-based LD method described in Section 4.3 with the 27 languages of Table 1; we have tried to train a new model with the data **TRAIN** in Table 1, but the new model works worse than the pre-trained model, which may be due to the fact that the amount of **TRAIN** is much smaller than that used to train the pre-trained model; the pre-trained model is thus used in our experiments; this system has already been shown superior to many methods, e.g., TextCat which is an implementation of (Cavnar and Trenkle, 1994) and CLD which is the embedded LD system used in Google's Chromium Browser, so we do not compare our COMB to these methods in this paper;
(2) DICT: uses the dictionary-based LD method described in Section 4.4 to detect 10 languages[6], as we only have dictionaries for the 10 languages;
(3) PROFILE: the user language profile method described in Section 4.5; the user language profiles of the 27 languages have been built using LANGID;

The COMB system uses the alphabet-based LD method (Section 4.2) to detect the 27 languages in Phase 1. If the result is in {Arabic, Hebrew, Greek, Russian, Chinese, Japanese, Korean}, we stop and

[5]if words are not space-delimited in a language, the first $2 \times n$ characters are kept

[6]the priority language list is {English, French, Spanish, German, Portuguese, Russian, Dutch, Polish, Italian, Turkish}

return the result. Otherwise, in Phase 2, COMB uses LibLinear (Section 4.6) to combine LANGID, DICT and PROFILE together to detect the 20 languages, which are the 27 languages of Table 1 minus the 7 languages detected by Phase 1. The LibLinear model is trained on the data **TRAIN** of Table 1.

5.3 Experimental Results

The experimental results on data set **LEN1** are shown in Table 2, from which we can see that for extremely short messages containing only 1 token, LANGID performs poorly with an average accuracy of 34.88%. DICT works better than LANGID on the 10 languages supported by DICT, which shows that dictionary-based methods are very useful in LD for short messages, though detecting 10 languages is much easier than detecting 27 languages. Moreover, PROFILE achieves very amazing accuracies on 1-token messages, which confirms the critical importance of user language profiles in LD of very short messages. At last, COMB successfully combines the three systems above and the alphabet-based LD method, achieving a relatively high accuracy of 73.69% on 1-token messages, which outperforms PROFILE by 11.48% accuracy. Note that the **AVERAGE** is macro-average.

Table 3, 4 and 5 respectively present the results on data set **LEN2**, **LEN3** and **LEN4**. LANGID's accuracy increases as the message length increases, as expected. Because more text is available. PROFILE maintains a stable accuracy at about 63.5% on all the 3 data sets, since it only depends on the user who sends the message, and is independent on texts. COMB again performs best among the 4 systems.

The experimental results on data set **FULL** are shown in Table 6. LANGID works much better on full-length messages than on shorter messages. PROFILE still keeps a stable accuracy at 63.57%. COMB performs best with an average accuracy of 84.61% on the 27 languages.

As a summary, the average accuracy of LANGID varies from 34.88% to 74.53% on the 5 test sets of messages of different lengths, which shows that the traditional LD methods relying on text-based features perform poorly on short messages. PROFILE works consistently well on the test sets with an accuracy at about 63%. Our proposed system COMB can effectively integrate LANGID, DICT, and PROFILE

Languages	LANGID	DICT	PROFILE	COMB
Arabic	96.85	N.A.	86.75	97.16
Catalan	1.55	N.A.	9.29	42.04
Chinese	94.96	N.A.	80.29	98.56
Czech	14.57	N.A.	47.75	62.33
Danish	14.44	N.A.	74.66	86.92
Dutch	8.10	25.60	53.17	70.46
English	89.43	100.00	99.79	94.08
Finnish	22.50	N.A.	42.16	57.05
French	13.22	29.09	93.51	82.69
German	24.36	35.76	91.55	93.52
Greek	90.33	N.A.	66.34	90.91
Hebrew	85.57	N.A.	84.76	92.48
Hungarian	20.03	N.A.	52.21	58.36
Indonesian	6.02	N.A.	23.04	82.23
Italian	9.09	32.28	89.98	92.58
Japanese	65.60	N.A.	40.53	65.18
Korean	85.04	N.A.	68.46	88.94
Malay	0.16	N.A.	0.00	14.62
Norwegian	2.39	N.A.	33.65	61.58
Polish	23.72	41.99	32.69	56.09
Portuguese	5.28	42.81	88.89	96.36
Romanian	7.37	N.A.	30.57	62.62
Russian	79.75	82.44	98.55	86.16
Slovak	8.53	N.A.	40.47	73.50
Spanish	22.64	42.14	93.71	43.40
Swedish	18.40	N.A.	67.53	87.45
Turkish	31.87	42.11	89.43	52.36
Average	34.88	N.A.	62.21	73.69

Table 2: Accuracies (%) of LD methods on **LEN1**.

Languages	LANGID	DICT	PROFILE	COMB
Arabic	99.80	N.A.	86.69	99.80
Catalan	1.79	N.A.	7.52	43.56
Chinese	95.46	N.A.	81.15	99.40
Czech	24.97	N.A.	50.88	70.98
Danish	33.96	N.A.	74.50	90.80
Dutch	25.48	31.43	54.10	70.15
English	77.40	100.00	99.66	92.58
Finnish	44.47	N.A.	43.84	62.91
French	37.16	33.06	94.37	85.23
German	43.47	46.32	91.89	92.53
Greek	98.43	N.A.	71.23	98.64
Hebrew	96.71	N.A.	86.17	99.67
Hungarian	37.20	N.A.	56.37	65.08
Indonesian	22.90	N.A.	21.99	85.31
Italian	32.25	46.03	91.86	94.05
Japanese	84.40	N.A.	41.93	83.42
Korean	91.56	N.A.	71.52	93.99
Malay	3.31	N.A.	0.00	14.88
Norwegian	16.87	N.A.	35.34	64.29
Polish	37.06	54.66	33.75	54.24
Portuguese	17.37	51.80	91.00	96.82
Romanian	12.59	N.A.	34.02	70.61
Russian	96.64	94.72	98.80	97.84
Slovak	14.10	N.A.	41.66	76.43
Spanish	50.38	52.02	94.54	46.89
Swedish	38.96	N.A.	68.69	91.55
Turkish	52.46	63.22	89.45	62.50
Average	47.67	N.A.	63.44	77.93

Table 3: Accuracies (%) of LD methods on **LEN2**.

Languages	LANGID	DICT	PROFILE	COMB
Arabic	99.90	N.A.	86.90	99.90
Catalan	8.17	N.A.	7.08	42.48
Chinese	96.60	N.A.	81.10	99.50
Czech	36.27	N.A.	50.60	72.34
Danish	48.30	N.A.	75.44	92.98
Dutch	40.61	46.60	54.31	74.31
English	70.76	100.00	99.70	91.17
Finnish	55.76	N.A.	43.02	66.26
French	51.12	40.02	94.91	88.19
German	59.18	63.59	91.98	94.08
Greek	99.50	N.A.	71.27	99.70
Hebrew	99.60	N.A.	86.25	99.90
Hungarian	44.93	N.A.	56.17	65.20
Indonesian	33.90	N.A.	22.30	87.70
Italian	54.12	64.96	92.07	95.18
Japanese	88.32	N.A.	42.62	88.32
Korean	93.71	N.A.	71.40	96.04
Malay	6.55	N.A.	0.00	13.90
Norwegian	34.02	N.A.	36.41	65.66
Polish	52.93	66.26	33.74	58.38
Portuguese	33.80	67.40	90.74	96.68
Romanian	23.13	N.A.	34.44	71.92
Russian	99.59	98.67	98.77	99.59
Slovak	23.74	N.A.	42.25	77.87
Spanish	62.20	69.66	94.96	61.09
Swedish	53.90	N.A.	68.69	93.52
Turkish	71.10	76.70	89.50	76.90
Average	57.10	N.A.	63.58	80.32

Table 4: Accuracies (%) of LD methods on **LEN3**.

Languages	LANGID	DICT	PROFILE	COMB
Arabic	99.90	N.A.	86.90	99.90
Catalan	15.31	N.A.	6.63	44.39
Chinese	97.00	N.A.	81.10	99.60
Czech	42.30	N.A.	50.60	72.30
Danish	54.12	N.A.	75.90	93.78
Dutch	53.66	59.48	54.56	76.83
English	75.75	100.00	99.70	92.89
Finnish	59.64	N.A.	42.37	66.57
French	65.93	51.80	94.99	92.18
German	69.07	73.67	91.99	95.20
Greek	99.50	N.A.	71.30	99.70
Hebrew	99.60	N.A.	86.30	99.90
Hungarian	50.40	N.A.	56.00	65.90
Indonesian	44.20	N.A.	22.30	88.80
Italian	69.14	77.25	92.08	95.49
Japanese	91.30	N.A.	43.22	91.70
Korean	95.60	N.A.	71.47	97.60
Malay	11.21	N.A.	0.00	13.31
Norwegian	48.95	N.A.	36.51	67.10
Polish	56.65	73.19	33.67	62.10
Portuguese	46.60	78.30	90.80	97.00
Romanian	29.20	N.A.	34.60	72.00
Russian	100.00	99.80	98.80	100.00
Slovak	30.10	N.A.	42.10	78.40
Spanish	73.15	81.76	94.89	72.65
Swedish	63.40	N.A.	68.90	94.20
Turkish	82.40	84.00	89.50	85.50
Average	63.85	N.A.	63.60	82.04

Table 5: Accuracies (%) of LD methods on **LEN4**.

Languages	LANGID	DICT	PROFILE	COMB
Arabic	99.90	N.A.	86.90	99.90
Catalan	22.50	N.A.	6.50	44.80
Chinese	97.10	N.A.	81.10	99.80
Czech	51.20	N.A.	50.60	72.40
Danish	61.80	N.A.	75.90	95.00
Dutch	80.70	86.70	54.50	80.80
English	90.60	100.00	99.70	96.80
Finnish	62.30	N.A.	42.20	66.80
French	88.60	83.80	95.00	96.10
German	85.00	90.10	92.00	96.30
Greek	99.60	N.A.	71.30	99.80
Hebrew	99.70	N.A.	86.30	100.00
Hungarian	54.80	N.A.	56.00	66.30
Indonesian	52.80	N.A.	22.30	89.80
Italian	91.10	95.10	92.10	96.10
Japanese	95.80	N.A.	43.10	97.70
Korean	97.90	N.A.	71.50	100.00
Malay	16.50	N.A.	0.00	12.10
Norwegian	67.50	N.A.	36.50	70.10
Polish	70.60	81.40	33.40	70.20
Portuguese	71.20	94.30	90.80	97.30
Romanian	42.60	N.A.	34.60	72.70
Russian	100.00	100.00	98.80	100.00
Slovak	47.40	N.A.	42.10	78.80
Spanish	94.40	99.00	94.90	95.00
Swedish	76.90	N.A.	68.90	94.50
Turkish	93.90	94.20	89.50	95.30
Average	74.53	N.A.	63.57	84.61

Table 6: Accuracies (%) of LD methods on **FULL**.

together, consistently outperforming all the baselines on test sets of messages of different lengths. COMB achieves a relatively consistent and high accuracy on messages of varied lengths from 73.69% to 84.61%. These results confirm the potential of the proposed system.

We also found both LANGID and COMB perform poorly on Malay and Catalan, which may be due to the fact that Malay is very similar to Indonesian, and that Catalan is similar to French and Spanish.

6 Conclusion

This paper presents a novel LD system for chat messages in mobile games. The system can effectively integrate both text-based and user-based LD methods. In our experiments, we achieve highly statistically significant ($p < 0.0001$ in T-test) improvements (10.08%-18.19% in absolute accuracy) over strong baselines on 27-language test sets which contain messages of various lengths.

Future work can investigate how to preprocess or normalize game chat messages to further improve LD. Moreover, adding more dictionaries may also be a future direction to improve the accuracy of the proposed LD system.

References

Bashir Ahmed, Sung-Hyuk Cha, and Charles Tappert. 2004. Language identification from text using n-gram based cumulative frequency addition. In *Proceedings of Student/Faculty Research Day, CSIS, Pace University*.

Timothy Baldwin and Marco Lui. 2010. Language identification: The long and the short of the matter. In *Proceedings of NAACL-HLT*.

Shane Bergsma, Paul McNamee, Mossaab Bagdouri, Clayton Fink, and Theresa Wilson. 2012. Language identification for creating language-specific Twitter collections. In *Proceedings of the Second Workshop on Language in Social Media*.

Simon Carter, Wouter Weerkamp, and Manos Tsagkias. 2013. Microblog language identification: Overcoming the limitations of short, unedited and idiomatic text. *Language Resources and Evaluation*, 47(1):195–215.

William B. Cavnar and John M. Trenkle. 1994. N-gram-based text categorization. In *Proceedings of the Third Symposium on Document Analysis and Information Retrieval*.

Hakan Ceylan and Yookyung Kim. 2009. Language identification of search engine queries. In *Proceedings of ACL-IJCNLP*.

Chih-Chung Chang and Chih-Jen Lin. 2011. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Ted Dunning. 1994. *Statistical identification of language*. Computing Research Laboratory, New Mexico State University.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef Van Genabith. 2011. # hardtoparse: POS tagging and parsing the twitterverse. In *Proceedings of the AAAI Workshop On Analyzing Microtext*.

Thomas Gottron and Nedim Lipka. 2010. A comparison of language identification approaches on short, query-style texts. In *Advances in information retrieval*, pages 611–614. Springer.

Lena Grothe, Ernesto William De Luca, and Andreas Nürnberger. 2008. A comparative study on language identification methods. In *Proceedings of LREC*.

Baden Hughes, Timothy Baldwin, Steven Bird, Jeremy Nicholson, and Andrew MacKinlay. 2006. Reconsidering language identification for written language resources. In *Proceedings of LREC*.

Xiaohua Liu, Shaodian Zhang, Furu Wei, and Ming Zhou. 2011. Recognizing named entities in tweets. In *Proceedings of ACL-HLT*.

Fei Liu, Fuliang Weng, and Xiao Jiang. 2012. A broad-coverage normalization system for social media language. In *Proceedings of ACL*.

Marco Lui and Timothy Baldwin. 2011. Cross-domain feature selection for language identification. In *In Proceedings of IJCNLP*.

Marco Lui and Timothy Baldwin. 2012. langid.py: An off-the-shelf language identification tool. In *Proceedings of ACL System Demonstrations*.

Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. 2011. Named entity recognition in tweets: An experimental study. In *Proceedings of EMNLP*.

Erik Tromp and Mykola Pechenizkiy. 2011. Graph-based n-gram language identification on short texts. In *Proceedings of the 20th Machine Learning conference of Belgium and The Netherlands*.

Tommi Vatanen, Jaakko J. Väyrynen, and Sami Virpioja. 2010. Language identification of short text segments with n-gram models. In *Proceedings of LREC*.

John Vogel and David Tresner-Kirsch. 2012. Robust language identification in short, noisy texts: Improvements to LIGA. In *Proceedings of The Third International Workshop on Mining Ubiquitous and Social Environments*.