IIITV@INLI-2018 : Hyperdimensional Computing for Indian Native Language Identification*

Ashish Patel[0000−0002−0409−736X] and Pratik Shah[0000−0002−4558−6071]

[1] Indian Institute of Information Technology, Vadodara
[2] Gujarat, India
{201771002,pratik}@iiitvadodara.ac.in

Abstract. Native Language Identification (NLI) is the task of determining user's native language based on their second language. NLI is an important task that has applications in different areas such as social media analysis, authorship identification, second language acquisition, and forensic investigation. In this paper, we propose to use Hyperdimensional Computing (HDC) as supervised learning for identifying Indian Native Languages from the user's social media comments written in English. HDC represents language features as high dimensional vectors called hypervectors. Initially, comments are broken in character bi-grams and tri-grams which are used for generating comment hypervectors. These hypervectors are further combined to create different language profile vectors. Profile hypervectors are then used for classification of test comments. The proposed approach was validated on 70% of training instances with 65.13% of accuracy. The accuracy of classification on the test set for bi-gram and tri-gram based feature encoding is 29.2% and 31.5% respectively.

Keywords: Native Language Identification · Hyperdimensional Computing · n-gram · Indian Languages

1 Introduction

The diversity of languages used in India has given rise to a large number of challenging problems, both for the linguist and for the computer scientist. The 2001 Indian census recorded 30 languages which were spoken by more than a million native speakers in India. English plays prominent role in every sector especially in higher education in India. It is certainly the most preferred language for reading and in official communication. The number of second or third language speakers of English is increasing day by day. Influence of their regional and native languages is observed in their usage of English. It is possible to identify the native speaker based on their English pronunciation and accent. But,

* Indian Native Language Identification (INLI) is a shared task held in conjunction with FIRE 2018. Aim of the task is to identify Indian language from user's social media comments written in English.

it is challenging to identify the native users based on their written comments in their non-native language. Native Language Identification (NLI) is important for a number of applications, such as authorship identification, forensic analysis, tracing linguistic influence in multi-author texts, and to support Second Language Acquisition research. It can also be used in educational applications such as developing grammatical error correction systems. The extensive use of social media and online interactions can be a reason of violent threat, which is a common issue faced by commenters. If a comment or a post poses any type of threat, then identifying the native language of the person will be one of the significant feature in finding the source.

In the INLI track [1], the task is to identify the native language of the user, based on their social media comments in the English language. Six Indian languages namely Tamil (TA), Hindi (HI), Kannada (KA), Malayalam (MA), Bengali (BE) and Telugu (TE) are considered for this shared task.

2 Related Work

NLI challenge has gained a lot of attention and has been part of shared tasks in various events [2, 3, 4] worldwide. This challenge can be broken into two subtasks: feature selection and classification. Different techniques are used to extract features from non-native language in terms of topic biases, spelling usage, and frequency of word usage, etc. Most of the researcher have considered unigrams, bigrams, and character n-grams [5], POS-tagged n-grams [6] as features. Current state-of-the-art uses Support Vector Machine (SVM) [6, 7], Multinomial Bayes [5], Random Forest [7], Logistic Regression [5] and feed-forward neural network [8] for classification. Authors in [9] have discussed a possible use of HDC for language identification.

2.1 Hyperdimensional Computing

The architecture of a computer that is in use since last 70 years, has been a milestone in mechanizing arithmetic of symbols. Efforts are reported in the form of architectural proposals starting from the representation of entities, memory, and processing units for computing [10]. Convenience and robustness of two value (binary) systems, in terms of realization, has been instrumental in its widespread use. It has been agreed upon that in the current state of evolution of the human brain, adding or multiplying large numbers is a difficult task, barring a few exceptions. At the same time, recognizing objects, using language to communicate and being able to reason are some tasks in which humans are efficient at [11]. In contrast, the computer is good at the former task whereas it needs considerable efforts and hand-holding to perform the later ones. The human brain contains large circuits having billions of neurons and synapses. It is believed that the concepts and entities are being stored distributively in such circuits [10]. Inspired by the architecture of the brain, artificial neural networks have been designed and have gained considerable recognition in the field of machine learning [12].

Properties of High dimensional spaces such as holistic representation, one-shot learning, and robustness against noise show similarity with the brain behavior. Instead of computing with numbers, brain-inspired hyperdimensional computing (HDC) emulates computing with hypervectors. In HDC the entities are represented using high dimensional vectors called hypervectors. In literature, there are different representation schemes proposed for HDC, for example HRR [13], BSC [14], Bipolar Spatter Code [15], MBAT [16] etc. In this paper, we have used Bipolar spatter code hypervectors typically comprises of randomly distributed +1's and -1's. Basic algebraic operations on hyperdimensional vectors are binding (multiplication), superposition (addition) and rotation (ρ). Binding of two hypervectors is an element-wise multiplication where as superposition is an element-wise addition. The addition preserves the similarity between added hypervectors and resultant hypervector. Multiplication binds two hypervectors in one and maps it to a hypervector in hyperspace which is nearly orthogonal to multiplied hypervectors. Rotation also generates a nearly orthogonal vector. Usually the similarity between two hypervectors is calculated by using cosine distance.

3 Task Description

Language	Training Instances
Bengali (BE)	202
Hindi (HI)	211
Kannada (KA)	203
Malayalam (MA)	200
Tamil (TA)	207
Telugu (TE)	210

Table 1. Languages and Training Instances

INLI is a multiclass classification task. INLI@FIRE2018 shared task focuses on identifying native languages namely Bengali, Hindi, Kannada, Malayalam, Tamil, and Telugu, from XML files. Table 1 describes the training data that is used for the task. Training XML files are annotated as per their native language. These XML files contain extracted social media comments in English.

4 Proposed Approach

For Indian native language identification task, training instances of six languages are available. For each language, the number of training instances (user comments) are mentioned in Table 1. It is assumed that the comments are from non-native English users. Since the comments are scribed in English (Latin) alphabets, a preprocessing step sets rid of special characters. Preprocessing involves removal of non-English characters, special characters and converting

the text in lowercase (alphabets). This ensures that the resulting string will contain twenty-seven different symbols in total, i.e. 'a,...,z' and a '_' - blank space. Each of these 27 symbols is assigned a hypervector, i.e. a bipolar spatter code. Each hypervector is a random string of +1's and -1's of length 10000. It is easy to show that with very high probability the generated 10k dimensional hypervector will have equal number of +1's and -1's. Moreover the mean of bitwise distance between any two randomly generated hypervectors will be 5000 bits. So the generated 27 hypervectors are all almost equidistant from each other with almost same number of 1's and -1's For more on bipolar spatter codes, refer to [15]. In the training phase, we scan through comments sequentially and create a profile hypervector for a given language class. For a given comment, we find out overlapping character tri-grams (or bi-grams) and for each of the tri-gram, we generate a hypervector from the character hypervectors as shown in the Fig. 1. Let $H : \{a, b, \ldots, z, \_\} \rightarrow \{-1, 1\}^{10000}$, be a function that maps a symbol to the hyperdimensional space and, the rotation $\rho : \{-1, 1\}^{10000} \rightarrow \{-1, 1\}^{10000}$ be defined as a function corresponding to a left circular shift operation on strings. Given a trigram "xyz", the corresponding trigram hypervector is $\rho\rho H(x) * \rho H(y) * z$, where $*$ is called the binding operation which in case of bipolar spatter code is element-wise multiplication. Finally, all the tri-gram hypervectors of a given class are superimposed to generate a class profile vector (language hypervector). Superposition of hypervectors is defined as the element-wise addition operation. For more details on the bipolar spatter codes and related algebra, refer to [15,17]. In this proposed approach HDC works as supervised learning algorithm and considers bi-grams/tri-grams as features. The occurrence of bi-grams/tri-grams in a language is a reasonable feature of that language. The language hypervector is a superposition (addition) of all bi-gram/tri-gram hypervectors occurring in the language and it is similar to all hypervectors added together.

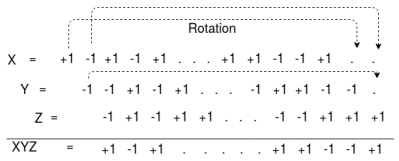

Fig. 1. Tri-gram hypervector generation

Algorithm 1 shows the training procedure, which takes comments (training set) as input, computes hypervectors for tri-grams and generates a class hypervector for each language. Tri-gram hypervector is generated by adding twice rotated hypervector of first character, once rotated hypervector of second character and third character's hypervectors. Similarly we can create class hypervector using bi-gram.

Algorithm 2 is used for predicting the class of test instances. The query hypervector (Q) is generated by similar process as the language hypervector is generated, i.e. superposition of tri-gram hypervectors corresponding to test/query

instance. This query hypervector is compared with all the language hypervectors. The similarity between bipolar hypervectors is calculated using the cosine of the angle between them. The language which has the highest similarity (lowest angle) with query hypervector is written in the output file with the test dataset name.

Algorithm 1: Training Algorithm

Input : Training Dataset
Output: Language profile Hypervectors
#N : Number of Languages;
#M_n : Number of training instances of n^{th} language;
#H: {a,...,z,_} $\rightarrow \{1,-1\}^{10000}$;
#$\mathcal{L}_n$: Profile hypervector of language n;
#$t_j^n(k)$: k^{th} symbol of j^{th} training instance of n^{th} language;
#$t_j^n(k) \in \{a,...,z,\_\}$;

for $n \leftarrow 1$ **to** N **do**
  for $j \leftarrow 1$ **to** M_n **do**
    for $k \leftarrow 1$ **to** *Length*(t_j^n)-2 **do**
      $\mathcal{L}_n = \mathcal{L}_n + (\rho\rho H(t_j^n(k)) * \rho H(t_j^n(k+1)) * H(t_j^n(k+2)))$;
      `// * binding, + superposition`
    end
  end
end

Algorithm 2: Testing Algorithm

Input : Test Dataset, Language profile hypervectors
Output: Native Language Identification of Test Dataset
#R: Number of Test Records;
#Q: Query hypervector;
#$M^{'}$: Predicted Language;
#N : Number of Languages;
#$H : \{a,...,z,\_\} \rightarrow \{1,-1\}^{10000}$;
#$\mathcal{L}_n$: Profile hypervector of language n;
#$t_j(k)$: k^{th} symbol of j^{th} test record;
#$t_j(k) \in \{a,...,z,\_\}$;

for $j \leftarrow 1$ **to** R **do**
  for $k \leftarrow 1$ **to** *Length*(t_j) $-$ 2 **do**
    $Q = Q + (\rho\rho H(t_j(k)) * \rho H(t_j(k+1)) * H(t_j(k+2)))$;
  end
  for $i \leftarrow 1$ **to** N **do**
    Angle[i]=Cosine($\mathcal{L}_i, Q$) ; `// Angle[] stores angle between` $\mathcal{L}_i$ `and Q`
  end
  $M^{'}$ = Language(Min(Angle)) ; `// Language() returns predicted language`
  Write predicted language $M^{'}$ in file;
end

5 Results and Discussion

For validation of classification algorithm, we have considered 70% as training and 30% as validation instances for given training dataset. Fig. 2. shows the normalized confusion matrix corresponding validation dataset. Data is cross-validated 10 times and average accuracy reported is 65.13%.

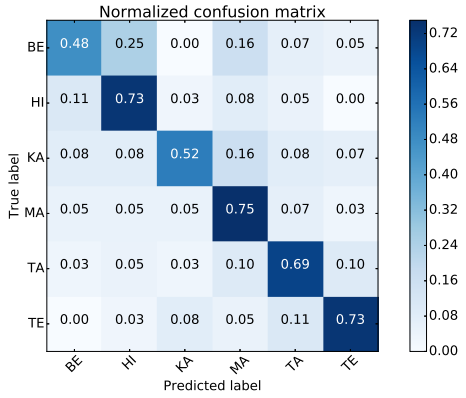

Fig. 2. Normalized confusion matrix of validation dataset

Run1 and Run2 results are corresponding to the bi-gram and the tri-gram language profile vectors respectively. The results are presented in Table 2 and Table 3 in which Table 2 and 3 represent result of TestSet-1 and TestSet-2. TestSet-1 is used for comparing results with previous year's shared task. In Run1 (bi-gram) 32.4% and Run2 (tri-gram) 32.1% accuracy is reported. For Run1 and Run2 for Testset-2 accuracy is 29.2% and 31.5% respectively. These experiments are conducted on 3.6 GHz Intel Core i7 PC with 4 GB of RAM. The python implementation of 3-gram training algorithm recorded 2.81667 minutes. The validation results differ from test results primarily due to the presence of unseen tri-gram (bi-gram) in the test data while training phase.

6 Conclusions

In this paper, we have proposed HDC approach for Indian Native Language Identification. The results are encouraging and can be enhanced further. Features selected for classification are tri (bi)-grams which are easy to aggregate. Since the efforts were put on proof of concept of HDC based classification, the other aspects of feature selection and efficient implementations were not addressed in this work. We believe that by deleting or rewriting lexical terms such as pls

Runs	Language	Precision	Recall	F1-score	Accuracy
Run1	BE	47.7%	55.7%	51.4%	32.4%
	HI	50.0%	7.6%	13.1%	
	KA	20.4%	40.5%	27.1%	
	MA	25.2%	58.7%	35.3%	
	TA	31.9%	29.0%	30.4%	
	TE	24.7%	23.5%	24.1%	
Run2	BE	45.1%	55.1%	49.6%	32.1%
	HI	52.0%	5.2%	9.4%	
	KA	22.4%	44.6%	29.9%	
	MA	24.7%	60.9%	35.1%	
	TA	31.0%	26.0%	28.3%	
	TE	28.4%	25.9%	27.1%	

Table 2. Results of Testset-1

Runs	Language	Precision	Recall	F1-score	Accuracy
Run1	BE	39.0%	27.5%	32.3%	29.2%
	HI	7.6%	3.6%	4.9%	
	KA	41.5%	38.0%	39.7%	
	MA	22.8%	51.5%	31.6%	
	TA	24.7%	30.0%	27.1%	
	TE	35.8%	17.6%	23.6%	
Run2	BE	45.2%	33.8%	38.7%	31.5%
	HI	8.8%	3.6%	5.1%	
	KA	43.5%	43.2%	43.4%	
	MA	23.9%	53.5%	33.0%	
	TA	28.1%	29.3%	28.7%	
	TE	32.1%	16.8%	22.0%	

Table 3. Results of Testset-2

(please), u (you), y (why), r (are), sry (sorry), fyi (for your information), etc. will improve the accuracy of the system. We also believe that punctuation marks may have a role to play in natural language identification. In future, we would like to include such features in HDC framework. We also would like to compare the proposed approach with state of the art methods in term of training effort, precision, recall and accuracy.

References

1. Anand Kumar, M., Barathi Ganesh, B., Soman K, P.: Overview of the INLI@FIRE-2018 Track on Indian Native Language Identification. In: In workshop proceedings of FIRE 2018, FIRE-2018, Gandhinagar, India, December 6-9,CEUR Workshop Proceedings.
2. Joel, T., Daniel, B., Aoife, C.: A report on the first native language identification shared task. In: In Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications. (2013)
3. Malmasi, S., Evanini, K., Aoife, C., Tetreault, J., Pugh, R., Hamill, C., Napolitano, D., Qian, Y.: A Report on the 2017 Native Language Identification Shared Task. In: Proceedings of the 12th Workshop on Building Educational Applications Using NLP, Copenhagen, Denmark, Association for Computational Linguistics (September 2017)
4. Anand Kumar, M., Barathi Ganesh, H., Singh, S., Soman, K., Rosso, P.: Overview of the INLI PAN at FIRE-2017 track on indian native language identification. CEUR Workshop Proceedings,2036,pp. 99-105 (2017)
5. Mathur, P., Misra, A., Budur, E.: LIDE: language identification from text documents. CoRR **abs/1701.03682** (2017)
6. Nicolai, G., Islam, M.A., Greiner, R.: Native language identification using probabilistic graphical models. In: 2013 International Conference on Electrical Information and Communication Technology (EICT). (2014) 1–6
7. Ulmer, B., Zhao, A., Walsh, N.: Native language identification from i-vectors and speech transcriptions
8. Sari, Y., Fatchurrahman, M.R., Dwiastuti, M.: A shallow neural network for native language identification with character n-grams. In: Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications. (2017) 249–254

9. Joshi, A., Halseth, J.T., Kanerva, P.: Language geometry using random indexing. In de Barros, J.A., Coecke, B., Pothos, E., eds.: Quantum Interaction, Cham, Springer International Publishing (2017) 265–274
10. Castro, L.N.D.: Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications. Chapman and Hall/CRC (2006)
11. Kahneman, D.: Thinking, Fast and Slow. FARRAR STRAUS & GIROUX (2012)
12. Vapnik, V.N.: Statistical Learning Theory. Wiley-Interscience (1998)
13. Plate, T.A.: Holographic reduced representations. IEEE Transactions on Neural Networks **6**(3) (May 1995) 623–641
14. Kanerva, P.: Binary spatter-coding of ordered k-tuples. In: Artificial Neural Networks — ICANN 96. Springer Berlin Heidelberg (1996) 869–873
15. Gayler, R.W.: Multiplicative binding, representation operators & analogy (workshop poster) (1998)
16. Gallant Stephen I., O.T.W.: Representing objects, relations, and sequences. Neural Comput. **25**(8) (August 2013) 2038–2078
17. Kanerva, P.: Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. Cognitive Computation **1**(2) (2009) 139–159