Text Classification for Hindi Documents using Supervised Machine learning Methods

Kiran Karandikar

Abstract

In recent times, there has been an increase in spam text messages (SMS) that contain regional Indian languages, in particular Hindi. However, there are limited resources that have classification of Hindi language. This is even though Hindi is a widely spoken language not only in India but also around the world; it is the 4th most spoken language across the world. As such, this premise showed me the need for an automatic text-classifier, that facilitates the creation of a document classifier for Hindi Spam texts or SMS. Thus, in the present work, I use supervised learning methods using Naive Bayes Classifier to facilitate text classification for Hindi Text and enable the identification for Hindi Spam text messages. Doing so, I hope to contribute towards improved spam Hindi filtration when the text is in Hindi, and thus reduce the risk posed by spam SMS.

1 Introduction

There is a rise in multilingual spam SMS, particularly Indian languages. India has over 121 languages, out of which 22 are separate official languages (*The Languages of India*, n.d.). This poses a challenge in identifying Hindi spam SMS due to the linguistic variance and language barriers. Furthermore, there are millions of documents in India, written in regional languages. To classify these manually would be a time-intensive and expensive process thus presents an opportunity to leverage automatic classification. Automatic classification can provide better management and retrieval of these documents.

After reviewing the literature on spam categorization, I learnt that there are many classifiers available for many Indian languages (e.g.,*Automatic Text Categorization*, n.d.; Bolaj & Govilkar, 2016; Calorx Teachers University et al., 2019). However, limited work is done for the classification of Hindi language, despite it being the 4th most widely spoken language in the world of today. Approximately 310 million people speak Hindi language in India. Given this background, in the present work my goal is to focus on document classifier for Hindi Spam SMS categorization. To do so, I first intend to create text classification for Hindi text. Text classification is a process of dividing a given set of documents into one or more predefined classes and is usually done automatically e.g., (*Automatic Text Categorization*, n.d.). Two main types of machine learning techniques are used for automatic text classification - supervised and unsupervised learning methods. Supervised learning methods assign predefined class labels to the testing documents using classification algorithms whereas in unsupervised learning methods grouping of testing documents are done using techniques like clustering. In the present work, the proposed system is a text classification for Hindi Text using supervised learning methods (using naive bayes classifier).

2 Key Concepts

In this section, I define key concepts and the Naïve Bayes classifier I use in the paper to achieve the goal of spam classification for Hindi SMS.

2.1 Naive Bayes Classifier

As stated in the introduction, the proposed system is a text classification for Hindi Text. I use supervised learning methods by leveraging the Naive Bayes Classifier. In statistics, Naive Bayes classifiers are a family of simple "probabilistic classifiers" based on Bayes' theorem. It has strong (naïve) independence assumptions between the features e.g., (Calorx Teachers University et al., 2019). Naïve Bayes classifiers are highly scalable.

They require several parameters linear in the number of variables (inputs/features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers (e.g., logistic regression). In the statistics literature, naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes.

2.2 Stemming and Lemmatization

In documents and textual materials, words can take on various forms of the word. For example – structure, structured, structuring etc. Thus, “the goal of stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form” (*Stemming and Lemmatization*, n.d.). The following example from (*Stemming and Lemmatization*, n.d.) illustrates an example:

$$am, are, is \Rightarrow be$$

Stemming is a crude heuristic process that often involves removing the derivational affix. Lemmatization on the other hand, uses proper vocabulary and morphological analysis of the words to derive the common base form.

3 Methodology

In this section, I describe the model flow and methodology I use. The system takes input as dataset of Hindi language sentences/SMS. The dataset then undergoes preprocessing steps which include text preprocessing, stop word removal, tokenization, stemming and adding parts of speech information. Then the features are extracted from preprocessed tokens. Finally supervised machine learning methods are applied to get output as classified Hindi sentence as spam or not spam. The supervised machine learning methods includes Naïve Bayes (NB).

3.1 Preprocessing

The main objective of pre-processing phase in sms/sentence/text classification is to enhance the influence between word and label of document. Steps of pre-processing for sms/sentence/text classification are as follows:

- **Text preprocessing:**
- **Tokenization:** Each row of the Hindi text will be divided into multiple tokens, and any unnecessary punctuation will be removed if required.
- **Stop Word Elimination:** To remove stop words from the given text, Github public dataset (*Stopwords Hindi (HI)*, 2016/2022)
- **Stemming:** As explained in the key concepts section, stemming is necessary to identify derivationally related forms of a word and reduce that to a common base form. For Hindi language, there is no library/package available to create a stemmed words list from a dataset or corpus. I therefore will be using manually crafted Hindi suffix list in order create a list of stemmed words.

3.2 Feature Extraction

Feature Extraction involves the process of selection in which the most relevant key words are selected based on how frequently they occur in the document and what their contribution(weight) is in the document. In this paper I use two types of feature selection techniques:

- **TF-IDF (Term Frequency, inverse document frequency):** In this the most relevant key words from the given text will be selected, based on its frequency and contribution (weight) in the text. TF (Term Frequency) measures how frequently a word appears in document) IDF (Inversed document frequency) measures how particular term is important for document.
- **Count Vectorizer:** A count vectorizer converts a collection of text documents to a sparse matrix of token counts. This is necessary to make model fitting possible.

3.3 Supervised Learning Methods

The key aspect of this step is Hyper-parameter Tuning. For this purpose, we will be using Grid Search. Grid Search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of alogarithm parameters specified in a grid. In addition to this, we will also be using the cross-fold evaluation (cv=5).

3.4 Expected Output

The expected output will be a classified text/sentence/sms.

4 DISCUSSION

In this section, I elaborate on the execution and application of the process delineated in the methodology section.

4.1 Data Source

For purposes of this project, I use the UCI SMS spam collection dataset (*UCI Machine Learning Repository: SMS Spam Collection Data Set*, n.d.) The dataset comprises of a collection of 5754 English messages (SMS). These are further tagged as ham (legitimate) or spam based on the classification.

4.2 Data Translation

The dataset I use however is entirely in English language. Therefore I had to apply translations to make it application to Hindi text message classification. I used the google translate service (*Google Translate*, n.d.) to translate the English words into Hindi words.

4.3 Data Pre-processing

4.3.1 Exploratory Data Analysis

In the next step, I conducted an exploratory data analysis to evaluate the basic statistical information about the data. The data is highly skewed towards ham label. Lot of rows contains Unicode characters which were not par sable.

4.3.2 Data cleaning

Since exploratory data analysis revealed that the data is highly skewed, the next step was to clean the data. During the data cleaning all punctuation marks, trailing white spaces, unrecognizable characters and non -Hindi (numeric or English words) were removed.

4.3.3 Stop word Elimination

To remove stop words from the given text, I used the Github public dataset (Stopwords Hindi (HI), 2016/2022)

4.4 Data Split

After this the data was split into three categories:

1. Training Data: Used to train the model
2. Validation Data: Used for performance testing.
3. Test Data.

4.5 Baseline

The baseline Pipeline is created using the above pre-processed data. This was done by using TfidfVectorizer from sklearn package to convert the text data into vectors. This was then passed to the `BernoulliNB` naïve baiyes classifier.

The roc_auc_score of this baseline was 0.5

4.6 Fine-tuning

The next step was to finetune the above baseline model. Because the language under consideration is Hindi, the goal was to find a tokenizer best suited for Hindi Language. Based on my initial research, the two libraries, the `indic-nlp-library` and `inltk` seemed relevant. However, after working on the sample data, both libraries failed to produce any meaning full tokens and further investigation is required in that regard.

After doing some more research, I discovered the indic bert library (*Ai4bharat/Indic-Bert · Hugging Face*, n.d.; *GitHub - AI4Bharat/Indic-Bert: BERT-Based Multilingual Model for Indian Languages*, n.d.; Kakwani et al., 2020).

According to (*GitHub - AI4Bharat/Indic-Bert: BERT-Based Multilingual Model for Indian Languages*, n.d.) indic bert is a multilingual ALBERT model. It covers12 major Indian languages. It is pre-trained on a corpus of around 9 billion tokens and evaluated on a set of multiple and varied tasks. Indic-bert has around 10x fewer parameters than other popular publicly available multilingual models while it also achieves a performance on-par or better than these models."

Thus, in order to tokenize the data, the pre-train model `'ai4bharat/indic-bert' referenced above was used as a baseline tokenizer.

The next step was to explore different combinations of hyper parameters for the TfidfVectorizer, BernoulliNB. The GridSearchCV with cross validation was used for this purpose. It was found that the inclusion of tokenization significantly improved the model performance jumping the roc_auc_score close to ~0.89.

The next step then was to perform Stemming and Lemmatization. In order to do that for the Hindi language, I referenced the work of (Anand et al., 2019; Pande et al., 2014; *Stemming and Lemmatization*, n.d.) to manually define the common suffixes and removing them from given word:

```
1: [u"ो",u"े",u"ू",u"ु",u"ी",u"ि",u"ा"],
2: [u"कर",u"ाओ",u"िए",u"ाई",u"ाए",u"ने",u"नी
",u"ना",u"ते",u"ीं",u"ती",u"ता",u"ाँ",u"ां",u"
ों",u"ें"],
3: [u"ाकर",u"ाइए",u"ाईं",u"ाया",u"ेगी",u"ेगा",
u"ोगी",u"ोगे",u"ाने",u"ाना",u"ाते",u"ाती",u"ाता
",u"तीं",u"ाओं",u"ाएं",u"ुओं",u"ुएं",u"ुआं"],
4: [u"ाएगी",u"ाएगा",u"ाओगी",u"ाओगे",u"एंगी",u"
ेंगी",u"एंगे",u"ेंगे",u"ूंगी",u"ूंगा",u"ातीं",u"ना
ओं",u"नाएं",u"ताओं",u"ताएं",u"ियाँ",u"ियों",u"ियां"]
,
 5: [u"ाएंगी",u"ाएंगे",u"ाऊंगी",u"ाऊंगा",u"ाइयाँ",
u"ाइयों",u"ाइयां"],
```

The words were stripped of the above suffixes and resultant words were then passed to tokenizer defined in previous step.

I also attempted applying CountVectorizer instead of TfidfVectorizer and in that case, the model performance was still around 0.98.

In the next step of the experiment I used the nltk libaray for the 'hindi.pos' corpus to investigate the impact of adding parts of speech to the existing sentence. Adding POS speech further improved the roc_auc_score to 0.98. Since we have a small-data size and it is highly skewed, the model was borderline overfitting.

This thus concludes the model hyperparameter tuning and feature enrichment. In the next section we offer commentary on the results.

5 RESULTS

We can obtain the accuracy of 0.97 (or even 0.98) on test data. However, this accuracy is because the model is overfitting due to small and highly skewed data set. Overfitting means that the model performed extremely well in test settings, but in real word settings, it will perform poorly, because the model is excessively adapted to the training data.

Some ways in which this can be improved in the future is by further investigating how word tokenization, stemming and lemmatization process can be refined. This is because the linguistics of Hindi language primarily act as a limiting factor here.

On the model side, the performance could be improved by trying different vectorization processes and machine learning models.

6 CONCLUSION

Here, I have presented the project report for the document classifier and spam-SMS Classifier for Hindi Language using Naïve Bayes Model. Because this was done within the limited scope of a class project, it can benefit from refinement and further improvement to enhance the accuracy and roc_auc_score to be eligible for real world application.

Acknowledgments

I thank the professors and TAs of the class that helped me learn the concepts I use in the paper.

References

Ai4bharat/indic-bert · Hugging Face. (n.d.).Retrieved December 12, 2022, from https://huggingface.co/ai4bharat/indic-bert

Anand, A., Chatterji, S., & Bhattacharya, S. (2019). Semi-supervised Approach for Hindi Stemming. *8th International Conference on Natural Language Processing (NLP 2019)*, 35–42. https://doi.org/10.5121/csit.2019.91204

Automatic text categorization: Marathi documents. (n.d.). Retrieved October 17, 2022, from https://ieeexplore.ieee.org/document/7503438/

Bolaj, P., & Govilkar, S. (2016). Text Classification for Marathi Documents using Supervised Learning Methods. *International Journal of Computer Applications*, *155*(8), 6–10.

Calorx Teachers University, Tamhankar, I., & Chaturvedi, A. (2019). Classification of Spam Categorization on Hindi Documents using Bayesian Classifier. *International Journal of Computer Trends and Technology*, *66*(1), 8–13. https://doi.org/10.14445/22312803/IJCTT-V66P102

GitHub—AI4Bharat/indic-bert: BERT-based Multilingual Model for Indian Languages. (n.d.). Retrieved December 12, 2022, from https://github.com/AI4Bharat/indic-bert#introduction

Google Translate. (n.d.). Retrieved December 12, 2022, from https://translate.google.com/?sl=en&tl=hi&op=docs.

Kakwani, D., Kunchukuttan, A., Golla, S., N.C., G., Bhattacharyya, A., Khapra, M. M., & Kumar, P. (2020). IndicNLPSuite: Monolingual Corpora, Evaluation Benchmarks and Pre-trained Multilingual Language Models for Indian Languages. *Findings of the Association for Computational Linguistics: EMNLP 2020*, 4948–4961. https://doi.org/10.18653/v1/2020.findings-emnlp.445

Pande, B. P., Tamta, P., & Dhami, H. S. (2014). A Devanagari script based stemmer. *International Journal of Computational Linguistics Research*, *5*(4), 119–130.

Stemming and lemmatization. (n.d.). Retrieved December 12, 2022, from https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html

Stopwords Hindi (HI). (2022). Stopwords ISO. https://github.com/stopwords-iso/stopwords-hi/blob/fae0a092c0fb8058f9c2470005467ee697b4763b/stopwords-hi.json (Original work published 2016)

The Languages of India: What Languages are Spoken in India? (n.d.). Berlitz. Retrieved October 17, 2022, from https://www.berlitz.com/blog/indian-languages-spoken-list

UCI Machine Learning Repository: SMS Spam Collection Data Set. (n.d.). Retrieved December 12, 2022, from https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection