indicNLP

A text processing framework for Indian Languages

Nisarg Jhaveri (201302195)
Team number: 45
Project number: 2

Mentor: Pulkit Parikh

Table of contents

Problem definition - Excerpts from scope document

Problem Statement

The aim of the project was to create a basic text based NLP framework for Gujarati. The plan was to include Sentence Breaker, Tokenizer, Stop word detection, Stemmer and POS tagging and variation identifier in the framework. Also, Key concept identifier, NER and document classifier were to be added to the framework depending on resources available.

Applications

Gujarati is one of the major languages in India. And looking at the current growth of use of computer technologies in local languages, it is important to make basic text processing techniques available for many Indian languages, so that it can be used in various tasks impecting a lot of people using computer technologies in local languages.

Some of the major applications includes,

- Better information retrieval, benefitted by stop word identification, variation identification and key concept detection in documents.
- Making focused crawlers using document classification.
- Better text-to-speech and machine translation systems, benefitted by sentence breaker, tokenizer and POS tagging.

Some of the modules that are intended to be part of the framework are really basic building blocks of any system in the area of NLP and IRE.

Challenges

Indian languages are in general morphologically rich languages when compared to other western languages such as English. Additionally, Gujarati is a morphologically richer when compared with Hindi. The rich morphology adds a lot of challenges.

Also, Gujarati an extremely resource-poor language in terms of computational linguistics resources. Being a resource-poor language, it makes many tasks, that are considered trivial, a lot more difficult.

The Solution - indicNLP

What is indicNLP?

indicNLP is a generic collection of common solutions to basic text processing problems, for Indian Languages. In fact, many of the modules are generic to most languages of world, but some are specifically designed for Indian Languages. indicNLP provides a set of tools, that can be used for most Indian Languages out-of-the-box. The specification of these tools are provided in further sections of this document.

Motivation

While looking up literature and working on some basic tasks, we realized that many Indian Languages are similar in nature with some differences. If we can create a generic solution for multiple similar languages, it should save a lot of efforts and repetition. The solution should be generic, but at the same time, it should take care of differences between languages. indicNLP is an attempt to achieve this goal.

Known issues and challenges

Though making generic solutions for multiple languages is useful, it may not always perform very well across languages. There are vast variety of differences among Indian Languages, especially when we talk about Indo-Aryan languages and Dravidian Languages.

indicNLP - An Overview

indicNLP includes following tools,

- Tokenizer and Sentence breaker
- Stop word detection and removal
- Stemmer
- Part-Of-Speech tagger
- Variation identifier
- Document Classifier

All the modules are designed keeping in mind the reusability and simplicity of use. All the modules supports multiple languages and multiple models if applicable.

Goals

- Easy to use
- Should work for most Indian Languages
- Reusable and sustainable code

- Easily integratable and usable with external modules or projects

Code quality and Quality assurance

One of the goals of the framework is *"Reusable and sustainable code"*. To ensure the reusability, several measures were taken.

- Code should not be *highly* specific to problem in hand

  Efforts were made to ensure that the code is not specific to current scenario and can work in different common scenarios, to ensure that the code is reusable in future.

- Extendable code

  The structure of framework is designed keeping in mind the extensibility of it. Adding a new module to framework and integrating it with current modules is easily possible with the current design of the framework.

- Adherence to standard coding guidelines

  The code currently is fully compliant with *flake8* standard, which is combination of *pep8* style guide and *pyflakes* semantic checker. Both of them are highly recognized and used standards for python.

- Automated testing, unit tests

  The framework for automated testing is in place. Currently code coverage of the automated tests included is 41%.

- Continuous integration

  All the style checks and tests run automatically after each commit made to the project repository.

Dependencies

We tried to keep the external dependencies of the framework minimal. As a result, the entire framework has only one external dependency.

It needs to following package to be installed for POS tagger and NER to work.

- `python-crfsuite`

Tools included in indicNLP

Tokenizer and Sentence breaker

Tokenization is breaking the text into smaller meaningful units (words, punctuation, etc..) called tokens, so that these can be processed by further modules. Along with tokenization, boundaries of sentences in a larger text are also determined, it is called sentence breaking.

The tokenizer included in indicNLP is a rule based tokenizer and sentence breaker. It handles various indian language scripts and languages.

It also allows different lists of not breaking prefixes (words that are commonly followed by a dot, but do not end sentence) for different scripts and languages. Currently such list is included only for Gujarati and English. English prefixes are always used as most indian languages are commonly codemixed with English.

It also has ability to automatically detect the scripts and use rules specific to that script if language is not specified.

One can also disable sentence breaking while using the tokenizer depending on the use case.

Supported Languages

The tokenizer currently works for following languages out-of-the-box.

- **Hindi** (hin) with Devanagari Script (deva)
- **Marathi** (mar) with Devanagari Script (deva)
- **Nepali** (nep) with Devanagari Script (deva)
- **Tibetan** (bod) with Devanagari Script (deva)
- **Konkani** (kok) with Devanagari Script (deva)
- **Bengali** (ben) with Bengali Script (beng)
- **Assamese** (asm) with Bengali Script (beng)
- **Punjabi** (pan) with Gurmukhi Script (guru)
- **Gujarati** (guj) with Gujarati Script (gujr)
- **Oriya** (ori) with Oriya Script (orya)
- **Tamil** (tam) with Tamil Script (taml)
- **Telugu** (tel) with Telugu Script (telu)
- **Kannada** (kan) with Kannada Script (knda)
- **Malayalam** (mal) with Malayalam Script (mlym)

Stop word detection and removal

Stop words are the words that are far too commonly used in a language and does not contribute much to the meaning of a sentence. Stop words can also be seen as function words.

Many NLP and IRE application requires filtering of stop words, as stop words can be act as noise in data for some applications

indicNLP includes statistical stopwords detection and removal module. While training, it calculates and stores most frequently used words and uses it later to remove stop words from tokenized text.

Stop word removal also removes punctuations from the tokenized text given to it.

It is again a generic module and can be learnt for any language given enough data.

Stemmer

Stemming is a process to obtain base word (commonly known as stem) of any derived word removing the morphological or otherwise inflectional parts. It is even more important for morphologically rich languages to get the stems. Stemmers are used widely in IRE systems when linguistic root of a word is not required.

A lightweight rule based stemmer is included in indicNLP. It requires a list of inflections suffixes for each language. The list can be handcrafted or learnt. Currently a manual list of suffixes for Gujarati is added.

Issues and Experiments

The issue with the rule based stemmer is that we need to have a list of all the possible suffixes. Also, it suffers with over-stemming even if we can get an exhaustive list of suffixes.

To handle the first problem, we tried to automatically derive the list of possible suffixes by taking the most frequent suffixes from the ILCI corpus, for Gujarati. But by visual inspection, the list was no way near complete or accurate. One possible reason behind it might be lack of enough data, with enough diversity.

Part-Of-Speech tagger

POS tags are one of the most informative and most commonly used information for higher level text processing. POS taggers tag the given sequence of tokens with most appropriate POS tags.

indicNLP includes a CRF-based POS tagger tool. We can train a POS tagger with different features and manage multiple different models, for different languages.

Experiments

We experimented with different features to build and evaluate POS tagger models. The dataset used was ILCI corpus. Experiments with done primarily with Gujarati and Hindi. All the experiments were done separately on *health* and *tourism* corpus provided by ILCI. All the accuracy provided are in form (*health*, *tourism*).

A baseline model was created by using just tokens as features for CRF. Total of four different models using different features were trained and evaluated. The models and features used are listed below. New features introduced in each model is highlighted.

- baseline: [token]
- model1: [token, **length, position, relative_position, prefix_1, prefix_2, prefix_3, suffix_1, suffix_2, suffix_3**]
- model2: [token, length, position, relative_position, prefix_1, prefix_2, prefix_3, suffix_1, suffix_2, suffix_3, **token-1, token-2, token+1, token+2**]
- model2: [token, length, position, relative_position, prefix_1, prefix_2, prefix_3, suffix_1, suffix_2, suffix_3, **suffix_4, suffix_5**, token-1, token-2, token+1, token+2]

Here,

length: length of token

position: position of token in sentence

relative_position: position of token in sentence / sentence_length

prefix_n: first n characters of token

suffix_n: last n characters of token

token[-+]n: context token, token at position [-+]n from current token

Results

The train, development and test set consisted of 70%, 15% and 15% respectively of the original data.

For Gujarati (best 91.58%),

	dev_health	*test_health*	*dev_tourism*	*test_tourism*
baseline	87.00%	87.30%	83.96%	83.42%
model1	91.00%	91.09%	89.22%	88.61%
model2	91.00%	91.27%	89.69%	89.26%
model3	**91.32%**	**91.58%**	**90.16%**	**89.76%**

In Gujarati, the best accuracy achieved on health corpus was **91.58%** and on tourism corpus was **89.76%**, both with the *model3*.

For Hindi (best 89.43%),

	dev_health	*test_health*	*dev_tourism*	*test_tourism*
baseline	87.61%	87.24%	83.44%	83.25%
model1	89.38%	89.13%	86.45%	86.61%
model2	**89.53%**	89.37%	86.96%	87.16%
model3	89.51%	**89.43%**	**87.08%**	**87.25%**

In Hindi, the best accuracy achieved on health corpus was **89.43%** and on tourism corpus was **87.25%**, both with the *model3*.

We can also notice that there was not much difference between *model2* and *model3* in Hindi, unlike Gujarati. This might be the indication and result of Gujarati being morphologically richer than Hindi, as the only two features added in *model3* were suffixes of length 4 and 5.

Variation identifier

Manier times, in a given language there are multiple, slightly different ways of writing a single word. It is important to identify such variations and treat variations of same word similarly. It is useful and necessary in a variety of tasks including search engines and machine translation systems.

In indicNLP, a rule based framework for variation identification is included. We can add rules for each language separately. Currently such rules for Gujarati are included.

An alternate approach to this is to check phonetic similarity of two written words. But Indian Scripts being phonetic in nature, it is almost the same thing if we write rules directly in Indian Language Script or convert it to phonetic notation and then find similarity.

Document Classifier

Document Classification is one of the important tasks when it come to IRE systems. This includes tasks like spam-filter or news category determination etc.

Naive Bayes classifier are known to work well for document classification problem. indicNLP includes a document classifier module which used Naive Bayes classifier to classify the documents. We can have multiple models for multiple languages.

Experiments and results

Classifiers were trained for Hindi and Gujarati using a dataset containing categorised news articles. The dataset had 70990 samples for Gujarati and 114678 samples for Hindi belonging to various categories. Training and testing set were prepared by taking 80% and 20% respectively for both the sets.

A Naive Bayes classifier, using stop word removed tokens of the title and text of the article, achieved **93.99%** accuracy in Gujarati and **85.26%** accuracy for Hindi documents.

Other experiments and results

Key concept identification

The original scope document states that key concept identification might be added depending on the availability of data. Well, we could not find any available dataset for Gujarati that night help with this task. Nevertheless, we attempted some basic rule based techniques, but it was extremely painful to work without any proper evaluation method to follow. As a result, the module was dropped from the framework.

Named Entity Recognition

Due to lack of annotated data in Gujarati, I could not perform experiments for statistical NER. However, I also realized that if someone wants to use CRF for NER, POS tagger can be reused to work as an Named Entity Recognizer, and there is no need to simply duplicate the module.

References

Irshad Ahmad. "indic-tokenizer, Tokenizer for Indian Scripts and Roman Script". https://github.com/irshadbhat/indic-tokenizer/.

Juhi Ameta, Nisheeth Joshi, Iti Mathur. "A Lightweight Stemmer for Gujarati". http://arxiv.org/pdf/1210.5486v2.pdf

Naoaki Okazaki. 2007. "CRFsuite: a fast implementation of Conditional Random Fields (CRFs)". http://www.chokkan.org/software/crfsuite/

Wikipedia. "Naive Bayes classifier". https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Girish Nath Jha. 2012. "The TDIL program and the Indian Language Corpora Initiative". In Proceedings of Language Resources and Evaluation Conference.

External links and tags

GitHub repository: https://github.com/nisargjhaveri/indicNLP

Project homepage: http://nisargjhaveri.github.io/indicNLP

Project report: http://nisargjhaveri.github.io/indicNLP/report.pdf

YouTube (Presentation and Demo): https://youtu.be/Pwh1NYAF5Gw

SlideShare (Presentation):
http://www.slideshare.net/NisargJhaveri/indicnlp-a-text-processing-framework-for-indian-languages

DropBox:
https://www.dropbox.com/sh/uslvll3pzi7dbz9/AAC8nFKLcNzw39ZAe6PFpb9ia?dl=0

Tags

indicNLP, IRE, NLP, Indian Languages, Tokenizer, stopwords, POS tagger, Stemmer, NER, Document Classification, Categorization, Spelling Variation Identification, Writing Variation Identification, text processing.

Assamese, Bengali, Gujarati, Hindi, Kannada, Konkani, Malayalam, Marathi, Nepali, Oriya, Punjabi, Sindhi, Tamil, Telugu, Tibetan.

Information Retrieval and Extraction Course, Major Project, IIIT-H.