Natural Language Processing Tool Tutorial

Node.js’ Natural Library

Information Retrieval, CAP6776
Justin Johnson, Z23136514
Fall 2017

Table of Contents

I) Introduction

Natural[1] is an open source JavaScript library. The library provides various natural language processing utilities to Node.js applications, similar to NLTK[2] for Python. Unlike NLTK, Natural is still a relatively new library and is not yet a complete one stop shop for natural language processing. Some of the functionality provided by Natural includes:

- Tokenizers
- String distance
- Tf-Idf
- Classification
- N-grams
- WordNet Integration
- Phonetics
- Inflectors
- Spellcheck
- Shortest/Longest Path Trees

II) Installation

Installation and set up with Natural is greatly simplified by the Node.js package manager, NPM. NPM is installed alongside Node.js, as the default package manager. In order to get started with Natural, Node.js must first be installed.

1. Install Node.js and NPM

Traditionally, JavaScript runs in the browser to create dynamic web sites. Node.js[3] is a JavaScript runtime environment that utilizes Google's V8 JavaScript engine to execute JavaScript outside the browser. With Node.js we can run JavaScript on a local machine or on a server. The Node.js installer can be download at https://nodejs.org/en/ .

NPM[4] is the default package manager for Node.js, and it is installed with Node.js.

2. Test Node.js and NPM install

From terminal, check Node and NPM versions. If install was successful, version will be printed to terminal. Node.js and NPM must be installed to proceed.

```
$ node -v
v9.1.0
```

```
$ npm -v
5.5.1
```

3. Creating a project directory

```
$ mkdir NaturalDemo
$ cd NaturalDemo
```

4. Initializing Node.js project

From within the new project directory, execute below statement:

```
$ npm init
```

Upon execution, the terminal will prompt several questions to define your new project. Prompt answers may be left blank to use defaults. Initialization will create a package.json file, which is simply a project configuration file that helps manage project build scripts and dependency versioning.

Now that we have a Node.js project initialized, we can install the Natural library.

5. Install Natural library

```
$ npm install --save natural
```

Now that the library is available in the project, all that is left to do is import the library.

6. Import Natural to project

```
const natural = require('natural');
```

A reference to the Natural API is now available through the variable natural and our project is now equipped and ready to begin using Natural. Natural’s documentation can be found: https://github.com/NaturalNode/natural .

III) Calculating TF-IDF Values

Natural provides several convenient methods for calculating document-term TF-IDF values given a collection of a documents.

1. Tokenize Documents

There are 4 tokenizers made available by Natural:

- Word Tokenizer
- Word/Punctuation Tokenizer
- Treebank Tokenizer
- Regular Expression Tokenizer

Tokenizing a document with Natural is as simple as importing 1 of the 4 Natural tokenizers and calling the tokenizer’s tokenize method, passing the document to the tokenizer as a string.

```
const natural = require('natural');
const tokenizer = new natural.WordTokenizer();

const document = "This is an example document, let's see it tokenized";

const tokenizedDocument = tokenizer.tokenize(document);
```

The tokenize method returns an array of tokens, as expected.

2. Stemming

Both Lancaster and Porter stemmer algorithms are available for use with Natural. When given an array of tokens as its parameter, the stemmer will return an array of stemmed tokens. This example tokenizes a document, and then applies the Porter Stemmer’s stem method to each token.

```
const natural = require('natural');
const tokenizer = new natural.WordTokenizer();

const document = "This is an example document, let's see it tokenized";

const tokenizedStemmedDocument = tokenizer
  .tokenize(document)
  .map(term => natural.PorterStemmer.stem(term));
```

3. TF-IDF Calculations

Natural’s TfIdf object accepts documents in the form of a string or an array of tokens, and outputs document-term TF-IDF values. By allowing an arrays of tokens, users of Natural can apply their own text pre-processing techniques.

```
var natural = require('natural');
const tokenizer = new natural.WordTokenizer();
const tfidf = new natural.TfIdf();


const document1 = "cats and dogs are popular pets";
const document2 = "cats like to sleep a lot";
const document3 = "dogs like to play outside with other dogs"
const document4 = "fish are not much fun";

const collection = [document1, document2, document3, document4];

const tokenizedStemmedDocuments = [];

// Push tokenized/stemmed docs onto new array
collection.forEach(doc => {
  tokenizedStemmedDocuments.push(
    tokenizer.tokenize(doc)
    .map(term => natural.PorterStemmer.stem(term)));
});

// Add each tokenized/stemmed doc to tfidf object
tokenizedStemmedDocuments.forEach(doc => tfidf.addDocument(doc));
```

This example takes a collection of 4 documents, tokenizes and stems each document, and then adds each document to an instance of natural.TfIdf. The ability to add documents to the collection dynamically greatly simplifies collection growth.

IV) Text Classification

The Natural library currently supports Naïve Bayes and Logistic Regression classification algorithms. The Naïve Bayes classifier was tested using the 4 Universities Data Set[5]. Classification test accuracy was then compared to that of Weka[6] as a benchmark.

Classifying with Natural's Naïve Bayes requires just 3 steps:

1. Instantiate Naïve Bayes Classifier

Creating a Naïve Bayes classifier is as simple as importing Natural and calling it's Naïve Bayes constructor.

```
const natural = require('natural');
const nbClassifier = new natural.BayesClassifier();
```

The Naïve Bayes classifier accepts a Stemmer as an optional argument. This allows users to define custom stemmers that implement the stemAndTokenize method, and then pass the stemmer to the Naïve Bayes constructor.

2. Add Training Documents to Classifier

All training documents must be added to the classifier in the form of (document, label) pairs. Similar to the TfIdf object, documents can be added to the classifier as either strings or arrays of tokens.

```
nbClassifier.addDocument('this product was great, I\'m so happy', 'positive');
nbClassifier.addDocument('I was dissapointed with this poor product', 'negative');
nbClassifier.addDocument('This was bad purchase, poor customer service and overall
disatisfied', 'negative');
nbClassifier.addDocument('Excellent service, very satisified with good quality',
'positive');
```

3. Train Classifier

Once all documents have been added to the classifier, it can be trained by calling it's train method.

```
nbClassifier.train();
```

4. Make Predictions

The classifier is now trained and prepared to make predictions on new data. Documents can be passed to the classify method as strings or arrays of tokens:

```
let test1 = 'I am happy with this product';
let classification1 = nbClassifier.classify(test1);
console.log('\nTest 1: ' + test1);
console.log('Result: ' + classification1);
```

Outputs: ‘positive’ as expected.

```
let test2 = 'I am disatisfied with purchase';
let classification2 = nbClassifier.classify(test2);
console.log('\nTest 2: ' + test2);
console.log('Result: ' + classification2);
```

Outputs ‘negative’ as expected.

5. Saving & Loading Classifiers

Once a classifier has been trained, it can be saved for use at a later time. Natural allows the classifier to be written to a JSON file, or serialized to a string that can be stored in DB.

```
nbClassifier.save('classifier.json', function(err, classifier) {
   // handle callback
});
```

Classifiers can then be loaded for use:

```
natural.BayesClassifier.load('classifier.json', null, function(err, nbClassifier) {
  const classification = nbClassifier.classfy('Some new document to classify');
});
```

6. Naïve Bayes Classification Results

As previously stated, Natural’s Naïve Bayes classifier was trained and tested using the 4 Universities Data Set. The accuracy was compared to Weka’s Naïve Bayes classifier accuracy results on the same data set. Weka was used as a benchmark to rate Natural’s performance.

Tool	Classifier	Accuracy
Weka	Naive Bayes	87.32 %
NPM Natural	Naive Bayes	73.38 %

Natural’s Naïve Bayes classifier performed significantly poorer than Weka’s Naïve Bayes classifier. Weka provides many configurable parameters, from the pre-processing step through the training and classification. Natural, on the other hand, provides minimal options to customize pre-processing, training, and classification.

V) Conclusions

Natural is a trending NLP resource for the Node.js community that is actively used by the JavaScript community. The library provides many useful features that simplify text processing, information retrieval, and text classification. Node.js and the Natural library was effectively used to create a REST API[7] for managing a searchable collection of documents and making text classifications. Despite Natural's progress, it's goal of becoming a one stop shop for Natural Language Processing has not yet been achieved. Easy to implement NLP solutions with Natural are convenient, but may not always yield the best results, as demonstrated when comparing the Naïve Bayes classification results with that of Weka's. As a powerful open source library with a lot of potential, Natural presents itself as a great opportunity to individuals interested in Natural Language Processing, JavaScript, and contributing to open source software.

VI) References

1. https://www.npmjs.com/package/natural
2. http://www.nltk.org/
3. https://nodejs.org/en/about/
4. https://www.npmjs.com/
5. https://github.com/johnsonj561/Search-and-Classification-With-Natural/blob/master/classification/data/readme.md
6. https://www.cs.waikato.ac.nz/ml/weka/
7. https://github.com/johnsonj561/Search-and-Classification-With-Natural