# Automatic Detection of Section Titles and Prose Text in Web Pages

Instruction Manual

Aug 24, 2018

# Contents

# Revision History

| Version | Date | Name | Description |
|---|---|---|---|
| 1 | 05/12/2018 | Abhijith | Initial document |
| 2 | 05/13/2018 | Abhijith | Added details on web services |
| 3 | 08/24/2018 | Abhijith | Final version |

# 1 Introduction

This document is intended for people/parties interested in running the code of the "Automatic Web Page Section Detector". This document contains detailed instructions for accessing, installing and running the application.

# 2 Prerequisites

We have tested our system on a Windows machine. We highly recommend using a system which has Windows as its operating system. Additionally, below are the prerequisites.

## 2.1 Hardware Requirements

| | |
|---|---|
| RAM | 16 GB or higher |
| CPU | 2.5 GHz or higher |
| GPU | Recommended |
| Free Space | 8 GB or higher |

## 2.2 Java Requirements

We use Java code for the following stages:

- Preprocessing

- Feature extraction for domain independent approach

- Generation of final (simple) HTML version of both the approaches

**Java version required**: **1.8.0_151**

**Stanford CoreNLP (3.8.0) jars:**

ejml-0.23.jar

javax.json.jar

joda-time.jar

jollyday.jar

protobuf.jar

slf4j-api.jar

slf4j-simple.jar

stanford-corenlp-3.8.0-models.jar

stanford-corenlp-3.8.0.jar

stanford-english-corenlp-2017-06-09-models.jar

stanford-ner-3.8.0.jar

stanford-ner.jar

stanford-postagger-3.8.0.jar

stanford-postagger.jar

xom-1.2.10-src.jar

xom.jar

**Other jars:**

weka.jar (version 3.8)

jsoup-1.11.2.jar

## 2.3 Python Requirements

We use Python code for the following things:

- Building word embeddings out of sections titles and prose text

- Training of neural network on the embeddings

- Classification of text into title/prose/junk in domain dependent model

- Performing k-means clustering for domain independent model

**Python version required – 3.6.1 :: Anaconda 4.4.0 (64 bit)**

Additionally, the following packages are required:

| Package | Version |
|---|---|
| beautifulsoup4 | 4.6.0 |
| bottle | 0.12.13 |
| gensim | 3.4.0 |
| Matplotlib | 2.0.2 |
| nltk | 3.2.5 |
| numpy | 1.14.1 |
| pandas | 0.20.1 |
| pandas-data-reader | 0.6.0 |
| scikit-learn | 0.19.1 |
| seaborn | 0.7.1 |
| tensorflow | 1.5.0 |

# 3 Running the Code

Before proceeding with this section, please read the Dataset documentation provided along with this document. This section contains instructions for running the code. All the necessary code is present in folders named “Java Code” and “Python Code”. Detailed Java code documentation is available in the folder “Javadoc”. Python code is composed using Jupyter notebook with inline documentation.

## 3.1 Domain Independent Model

### Python

The domain-independent model performs k-means clustering to create two clusters. Clustering is implemented at python end as a web service, and Java code invokes this to complete clustering and classification.

Run the `Lab_Web_Service_KMeans.ipynb` file. Please note that the file uses the port "12007" to host the service. If the port is unavailable in your system, change the code to accommodate the new port. Once this is running, you will get the below prompt:

```
Bottle v0.12.13 server starting up (using WSGIRefServer())...
Listening on http://localhost:12007/
Hit Ctrl-C to quit.
```

### Java

The class `com.lab.model.DomainIndependentModel` is responsible for generating the sanitized version of the input by detecting segments using unsupervised learning. Please read the Javadoc of this class for more information. You may need to change the following parameters according to your local environment.

| Variable | Purpose |
|---|---|
| strTemp_File_Features | Full path of the temporary file which will be used to store features of text. Note: Need to read the same file in Lab_Web_Service_KMneans.ipynb |
| model | Full file path of Stanford CoreNLP' s CONLL classifier model |
| strParentFolder | Base folder of the dataset |
| strTestFileName | File name to be tested. This will be "priv.html" for privacy policies, "TOS.html" for terms of service and "Misc.html" for miscellaneous documents. |
| strOutputFileName | "dom_ind.html" (we recommend leaving it unchanged). This is the output file. |

| | |
|---|---|
| strWebService_Label | Web service URL to perform classification using already built k-means model. If you have changed the port in the python step, then change here as well. |
| strWebService_Clustering | Web service URL to perform k-means clustering. If you have changed the port in the python step, then change here as well. |

Run the class `com.lab.model.DomainIndependentModel`. The system generates the output file.

## 3.2 Domain Dependent Model

### Python

The domain-dependent model creates word embeddings out of section title and section prose text. To achieve this, we need a collection of title text and prose text segregated and ready to train. The file `Lab_Build_WordEmdedding_Title_Prose.ipynb` creates these embeddings.

To prepare data required for training the neural network, we need to calculate the semantic distance/score between text in question and title embedding. Similarly, we need the score between text and body/prose embedding. To achieve this, we have the file `Lab_Web_Service_Title_Prose_Distance.ipynb`. This file uses the port "12008" to host the service. If the port is unavailable in your system, change the code to accommodate the new port. Once this is running, you will get the below prompt:

```
Bottle v0.12.13 server starting up (using WSGIRefServer())...
Listening on http://localhost:12008/
Hit Ctrl-C to quit.
```

The training happens through the file `Lab_DNN_Classifer_Model.ipynb`. The trained model is saved under a sub directory called "tf_model" for classification purpose.

Run the `Lab_Web_Service_Predict_Title_Prose.ipynb` file. This file is responsible for using the pre- trained neural network classifier for classification of text into title, prose and junk. Please note that the file uses the port "12009" to host the service. If the port is unavailable in your system, change the code to accommodate the new port. Once this is running, you will get the below prompt:

```
Bottle v0.12.13 server starting up (using WSGIRefServer())...
Listening on http://localhost:12009/
Hit Ctrl-C to quit.
```

**Java**

The class `com.lab.train.Lab_CreateTraining_File` does the job of reading the HTML train set and creating the training file for the neural network to train upon.

The class `com.lab.model.DomainDependentModel` is responsible for generating the sanitized version of the input by detecting segments using supervised learning. Please read the Javadoc of this class for more information. You may need to change the following parameters according to your local environment.

| Variable | Purpose |
|---|---|
| strParentFolder | Base folder of the dataset |
| strTestFileName | File name to be tested. This will be "priv.html" for privacy policies |
| strOutputFileName | "dom_dep.html" (we recommend leaving it unchanged). This is the output file. |
| strWebServiceHeaderDistance | Web service URL to get the semantic distance between the current text and header/title word embedding. If you have changed the port in the python step, then change here as well. |
| strWebServiceBodyDistance | Web service URL to get the semantic distance between the current text and body/prose word embedding. If you have changed the port in the python step, then change here as well. |

| | |
|---|---|
| strWebServiceClassifyText | Web service URL to perform classification using the neural network. |

Run the class `com.lab.model.DomainDependentModel.` The system generates the output file.

# 4 Evaluation

We evaluate our system in two ways:

## 4.1 Precision, Recall and F-1 scores of Segment Detection

After generating the output files from 3rd section, run the Jupyter file `Lab_Calculate_Precision_Recall_Title_Prose.ipynb.` This file compares the gold version with our output. It calculates and writes the precision, recall and F-1 scores into a html file. This file is displayed automatically in the default web browser.

## 4.2 Coverage

Run the java class `com.lab.test.TestCoverage` to get the coverage values of all the web documents passed. This class compares the text from the gold version with the text from our output version. The results are stored and displayed automatically in the default web browser.