Automated Text Tagging and Summarization Tool Final Report

Robert Rash
Stefan Popov
L.J. Brown

https://github.com/andy-rash/CSE4345

December 1, 2017

Contents

1 Executive Summary

This report outlines the requirements, design, development, testing and implementation process for a system, intended for filtering and tagging of text data. Our system is a software tool that utilizes natural language processing and machine learning techniques geared predominantly to aid social media

publishers and newspaper editors in quicker turnarounds in the areas of content posting and online publications. Our system identifies keywords within a text and ranks them based on their relevance. We aim to also provide an abstract containing the most relevant information from the input data. Our product includes a python library, a web API and a web front-end with a graphical user interface for the use of summarization, analytics and tagging of text documents.

The system is focused on providing online newspapers, referred to as primary clients, and publishers, referred to as customers and users, automation tools to aid in sorting and tagging text data. Although our ultimate goal is to expand to a wide variety of clients, this initial design is focused predominantly on that subset of our final intended client base. The initial focus was on the automated tagging system which will suggest the main subjects and keywords based on an analysis of raw text. The future goal is to implement a summarization feature which will reduce the raw text input to a user specified length.

2 Requirements

Objectives The objective will be to provide automation tools to aid in sorting, tagging and summarizing text data in the form of a python library, web API, and web front-end.

Business Objectives The project aims to reduce costs, increase profits and provide our primary clients with more efficient tools to publish content. The hope is that our API will enable writers and publishers to track and search for desired content more efficiently, by reducing our users workload and increasing the efficiency of reference-related activities and report generation. A working prototype is developed and deployed and will be made publicly available by the end of the 2017 calendar year. A thorough analysis on the ways these benefits will be provided is included in later sections

2.1 Preliminary Requirements

Functional Requirements The application's initial specification aimed to take in as input either raw English text or a TXT file containing text in English. The milestones for the output would be 1. the main keywords in the

inputted text and 2. a summarization of the contents of the text in the form of 3-4 short sentences. A user interface was also planned to be included to exhibit the functionality of the application programming interface which will be made available to the client for internal integration. The initial functional requirements were therefore as follows:

1. Support uploading of large text files
2. Ability to establish a REST communication between front and back-end
3. Support transfer of large JSON objects
4. Ability to parse large amounts of text
5. Ability to highlight keywords based on an algorithm that will be discussed in later sections
6. Ability to extract key information from the piece of text based on the algorithm and form a short summary of the contents

Optional Features The development team was hoping to increase the initially proposed functionality of the system by attempting to implement further functionality, if there was interest from the client and time permits. The algorithms used for text summarization and tagging could have been expanded to serve one or more of the following purposes:

1. Automated tagging of news articles
2. Automated posting of content to social media, news outlets and forums
3. Spam filtering
4. Analysis of Social Media

2.2 Final Requirements

The achieved requirements for the final deliverable of the project were all of the listed Functional requirements in the previous subsection. There were also initial plans to implement support for upload of PDF documents, but that milestone was not achieved, as the development team had faced issues with the requirements. None of the Optional Features listed was implemented due to lack of time and complexity of the problems involved.

3 Scope

The project aims to ease the internal and external search engines used by local online newspaper websites by automatically providing keywords that could be used for keyword tagging of content that is to be published to the website. It can also be used as a summarization tool to create abstracts for the same content. The project provides an application programming interface and a graphical user interface in the form of a web application to interact with the API. The interface is able to accept raw text input, and TXT files and parse them returning both the keywords and a short summarization. The project is aimed entirely towards text written in English and therefore online newspapers that post their content in English. A known limitation of the text summarization functionality is its inability to handle quotations in the texts.

The client, customers and users will be provided with an application programming interface (API) that will allow them to interact with the back-end algorithms performing the summarization and keyword identification to be implemented by the development team into the current system in place. This API will come with extensive documentation of its functionality and capabilities and will allow for some customization of aspects such as number of keywords. A simplified user interface was developed for the purposes of demonstrating the product in front of stakeholders. The user interface is not be able to customize the number of keywords to get back, but would rather have a constant number of 5.

4 Benefits

With the amount of content that is generated today in online newspaper media portals, said portals compete for traffic between themselves. Local online newspapers have less of a client base because of the lack of resources to provide an automatic tool for increasing searchability through Search Engine Optimization of their content. The most crucial approach to increase SEO is tagging as it provides for more visibility and traffic. Furthermore, studies show that millennials tend to skim text and ignore content that is longer than a paragraph. This project will aim to increase the popularity and traffic among the younger generations, which are a growing percentage of Internet users.

Using our product, the primary client can increase content author utilization and increase online visibility through SEO. By automating the process of tag generation, article authors, editors, and publishers alike will not have to spend precious minutes extracting tags out of a mass of copy, and instead they can spend their time on other useful tasks. In addition, the algorithm can extract the most meaningful words from the text, meaning search engine results can be tailor-made to bring the content to those for whom it matters most.

By providing a summarized text, the media platform can be more accessible and more appealing to readers, as it will allow them to quickly choose which articles they want to focus on and which to ignore.

5 Initial Risk Analysis

The text summarization algorithm upon which our product is based may not be able to adequately translate to keyword selection. In such a case, we would need to consider an alternative algorithm, or algorithms, upon which to base the product.

Delays in spinning up or properly configuring our deployment infrastructure for testing and production could hinder our development cycle and potential live demos. Additionally, unexpected or uncommunicated changes to configuration could pose greater threats to our time management in the midst of the project. We believe through proper communication and employment of a change management process we can mitigate this risk considerably. However, in such event a robust local development environment should be available for development to continue. Furthermore, this environment should replicate the specifications and behaviors of our production infrastructure as faithfully as possible.

6 Cost

The anticipated and final cost of this product was $0.99, which was made possible by the fact that only open-source technologies were used for the development of the web application, application programming interface and Python library. The only actual cost came from registering the domain name atts.me, which was a one time purchase for the purposes of hosting the

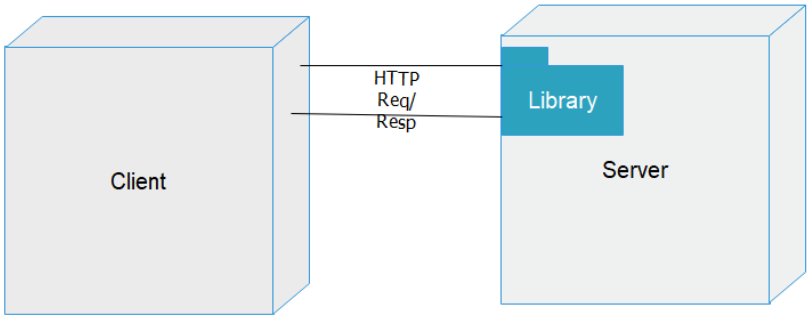

Figure 1: Client Server Architecture

API. This cost excludes the personal machines each team member had and any costs to power them.

In terms of man hours, each team member spent roughly about 25 hours total over the course of the semester, resulting in a total of 75 hours for development, research, testing and documenting.

7 Design

7.1 System Design

The system was built using a Client Server architecture, with the user interface web application deploying code on the client side for display and interacting with the server side that hosted the API. The server also housed the Python library package that was developed to perform the summarization and keyword extraction.The API interacted with the library to call the required functions and retrieve the needed data based on input. Figure 1. shows a diagram of the basic structure of the architecture.

The Client Server architecture was chosen for several reasons, the main one being the need for abstraction of the business logic that is being

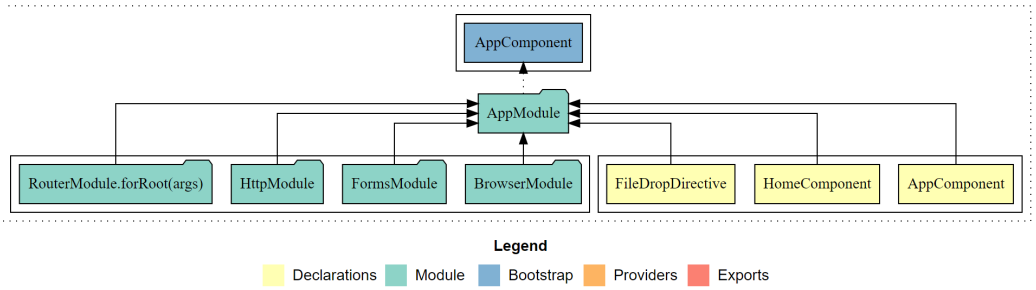

Figure 2: Classes and dependencies in Angular application

generated on the back-end from the user in the form of an application programming interface that simply interacts with the Python package. It also allowed for separating concerns between the three tiers that were created and ensured low coupling between the components, which is essential for the scalability of the overall system. The scalability will be further analyzed in the Maintenance section later on in the report.

7.2 Program Design

A lot of the structure of the program design was carried out by the programming tools used to develop the system that will be described in the Implementation section of this report. The classes and dependencies for the User Interface are displayed in Figure 2. and are standard for the functionality provided by an Angular 2 application. The component that acted as the pinpoint for the program is the HomeComponent, however, it implemented a lot of readily available modules, such as the Forms, Http and Browser modules that allowed it to provide the range of functionality. Describing the architecture of an Angular 2 application is beyond the scope of this report.

The application programming interface was intended to be fairly simple and act as a gateway between the web interface or user and the library package. A UML Diagram of the classes that were used to do this is shown in Figure 3. Please note the UML Diagram is an abstraction and the actual functionality and specifics of the API will be described in a separate document. A short description of how the server was deployed will be provided, but again this information is largely outside the scope of this report.

Figure 3: UML Class Diagram for the API

Lastly, the library package that performed the majority of the summarization was mostly functional and involved procedural calling of methods, which are shown in the Deployment diagram in Figure 4. The description of the design of those methods is described in the separate Word Embedding paper provided with this report.

8 Approach

The development team intended to follow an Agile approach, consisting of three 3-week sprints, each of which should result in deployable code and be used to refine requirements based on feedback from the client and users. The Agile approach was chosen to ensure that the basic functionality can be delivered to the client within a short time frame and solve the initial problem. It would also provide for more transparency for the API that will be provided for the client and can be adjusted accordingly as needed, as well as enable for changes in the initially proposed requirements and deliverables. During each sprint, all team members will perform stand-ups to share information of their progress on the assigned tasks. The stand-ups will be scheduled for Tuesdays and Thursdays.

Although solving the initial task was achieved within a reasonable time frame, the team's following of the Agile approach was not as successful as intended and certain road blocks detained from successful execution of the initially planned 3 Sprint milestones, that will be described below, with information of how the Sprint was altered and achieved in the end.

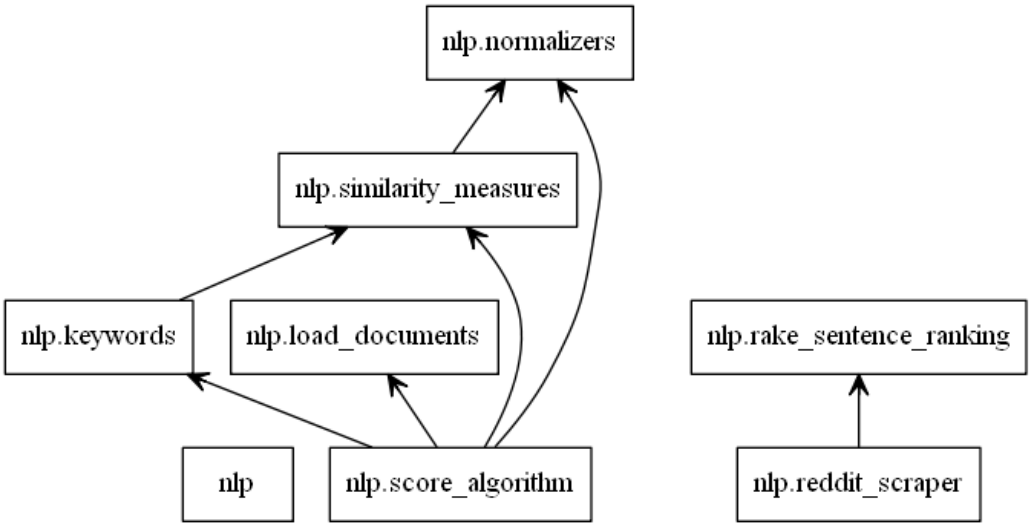

Figure 4: Deployment diagram of the Library package

Sprint 1 Initial Deliverables For Sprint 1 the development team wanted to focus on meeting the initial requirements of analyzing and extracting keywords from raw text as input. The team was going to focus on creating the basic workflow of the application for that feature by building a simplified web application that through HTTP requests will be able to interact with an API. The web application would be coded in TypeScript using the Angular 2 platform, while the API will be developed in Python, using the Falcon framework.
The finished product at the end of the first three week period will allow users to input text in a text box from a graphical user interface. The text boxs contents will be sent inside of a JSON object to the API. The API will invoke a script which will perform the algorithm on the input and return the keywords inside of a JSON object to the frontend that will be able to display them in a meaningful manner.
The duration of this sprint was three weeks, with a delivery date of Oct 19 2017.

Sprint 1 Achieved Deliverables During Sprint 1 the development team had issues with setting up the development environment and had to delay actual implementation as planned. The team ran into problems when setting

up both the web application and API, and chose to delay those milestones for Sprint 2, focusing more on research activities for keyword extraction and summarization. The keyword extraction algorithm was delivered on time and tested with a small input size to ensure correctness as much as possible, a method that will be described in the Testing section of this report. Unfortunately, Sprint 1 did not deliver a thoroughly finished product as intended.

Sprint 2 Initial Deliverables Sprint 2 was going to consist of expanding on the initial requirements and implementing an algorithm that will provide a short abstract of the parsed text. The API platform was planned to also support uploading of TXT files as extra functionality on top of the simple text box from Sprint 1. The product from Sprint 1 was to be shown to a client and the requirements modified again based on their feedback. Should changes need to be implemented to the workflow of the product, they would be accommodated for, otherwise, the workflow will remain the same. The duration of Sprint 2 was until Nov 9 2017.

Sprint 2 Achieved Deliverables After the initially planned Sprint 1 was not completely delivered, Sprint 2 had to be adjusted accordingly. Sprint 2 now entailed developing a simple web application that would be used to demonstrate the functionality of the product, with support for only parsing raw text in a text box. The API was developed and instantiated on a development server, with a domain name purchased, to host the API. It was able to leverage the functionality of the Python library to extract keywords from the input raw text that the front-end had sent and return a specific number of keywords. As described, the number of keywords requested by the front-end was a constant 5, while the API allowed for choosing a desired number. No user testing or client acceptance testing was done in this Sprint, with the former being scheduled for Sprint 3 and the functionality to upload files was also pushed back to Sprint 3. The text summarization implementation was also delayed for Sprint 3.

Sprint 3 Initial Deliverables Sprint 3 was to focus on refining the previously developed product based on update of requirements from the client and adding functionality, should those requirements be updated from the initial ones. The development team wanted to also include support for PDF file parsing. The delivery date of this Sprint is Nov 30 2017.

Sprint 3 Achieved Deliverables Sprint 3 was tasked with implementing the deliverables from Sprint 2 and the development team managed to do it successfully. The API and web application were expanded to support uploading of TXT files and returning keywords and a short summary of the text, whether from a file or from a raw text box input. A small sample of users, all college students, were asked to identify any potential usability issues with the web application and adjustments were made accordingly based on their feedback. The product, with all corresponding reports and documentation, was delivered on Dec 1 2017.

9 Technical Implementation

For the web application, the Angular 2 framework was used, written in TypeScript. It was chosen mainly for the ease of development and hosting of an application and the large number of readily available modules and libraries that were used to develop this. It allowed for a simplistic and stylish front-end that did not distract the user with overloaded appearance and required less than 3 clicks to achieve any task related to getting the needed information. The implementation comes with a script that allows to easily install all dependencies, including NodeJS, which Angular 2 is based off.

The API was developed in Python using the Falcon framework. Falcon is a Python framework designed for speed and reliability. It compiles using Cython when it can, so it has low latency compared to other Python web frameworks. It was chosen due to its portability and the fact that it is specifically designed to be used in REST communication, one of the requirements for this project. Falcon uses very simple syntax, allowing for endpoints to be written quickly while maintaining code quality. The open-source nature of the Falcon project allowed for ease in handling a CORS issue that the team was encountering.

Gunicorn is used to daemonize the Falcon app. The Gunicorn server is made further reliable through the use of a SystemD service. The server was instantiated with 4 worker processes that would run concurrently to serve multiple customer requests without causing flooding of the server.

Requests are served to the Falcon app through a Caddy webserver. Caddy allows for incredibly simple, yet powerful webserver configuration. It enables HTTPS by default for secure communications, but the configuration is significantly simpler than other choices such as Nginx or Apache. For our

use, the Caddy server has been made into a SystemD service in order to run continously. The API is hosted on a Digital Ocean droplet. Digital Ocean allows for very quick turnaround and complete customization when setting up hosting.

A quick summary of the methods implemented in the Python summarization and extraction library were as follows:

1. **Sentence Boundary Disambiguation**
   Operating on individual sentences is useful for a number of our natural language processing methods. Unfortunately, locating the boundaries of sentences in natural language is a challenge because of the ambiguity of punctuation marks and our implementation does not always handle those very well.
2. **Stop Word Removal**
   Analysis of core content in text is often easier when words of little value are removed. "Stop words" usually refer to the most common words in a natural language, however the list of words that fall under this category is ambiguous. Our implementation used a pre-set list we acquired through research.
3. **Word Stemming**
   In order to perform operations such as word frequency distribution, it is often useful to reducing inflected or derived words to their root (stem).
4. **Parts of Speech Identification**
   It is useful for many methods of natural language analysis to group words into lexical categories. Some of these categories could include parts of speech. Our implementation does that using an open-source Python library.
5. **Term FrequencyInverse Document Frequency**
   TF-IDF is a numerical statistic intended to reflect the importance a word relative to a document. This statistic is often used as a weighting factor for search, user modeling, information retrieval, and text mining. The value is proportional to the number of frequency of a word in a particular document, and inversely proportional to the frequency of the word in the corpus.

6. **Sentence Ranking**
   The need for solid sentence ranking algorithms based on importance to a document or search query perhaps one of highest importance in modern natural language processing. Extractive methods tend to rank sentences in a document based on keyword identification from metrics such as TF-IDF.
7. **Word Embedding**
   Word Embedding refers to a set of abstractive natural language modeling and feature learning techniques where words and or phrases are mapped to a vector space. Word Embedding can be used to measure word similarities and is a major focus of our project, with more information provided in the separately linked GitHub repository.
8. **Text Summarization**
   The process of shortening a text document in order to create a summary with the major points of the original text document. Automated text summarization is generally abstractive and is involves elements of machine learning and data mining. The main goal is to find a subset of data which contains the most core information of the entire set. Our implementation managed to reduce text to about 20% of its original length.

Further analysis and details on the topic of the keyword extraction and word summarization can be found in our separate paper that described the design as well as implementation of these methods.

10 Security

Being that our product is a hosted web application, there are a wide variety of potential attack surfaces. These may include, but are not limited to cross-site scripting, cross-site request forgery, man-in-the-middle attacks, injection attacks, and denial-of-service attacks. These are all well-known attack vectors, and therefore have wide support across software stacks in terms of mitigation.

Attacks such as denial-of-service, man-in-the-middle, and cross-site scripting, and cross-site request forgery are, for the most part, mitigated through server configuration. In our case, the risk of denial-of-service has been mitigated through use of a strict firewall which blocks requests on any port

outside of ports 22, 80, and 443 (SSH, HTTP, and HTTPS) on top of Cloudflare DNS services, which do even more to prevent such attacks. The risk of cross-site request forgery has mostly been mitigated through disabling CORS on the server. The risk of man-in-the-middle attacks and cross-site scripting attacks has mostly been mitigated through the use of HTTPS, which greatly reduces the attack surface for such threats.

On the other hand, injection attacks are typically performed through improperly sanitized inputs. Since our web app only has inputs, this is a potential attack surface. However, this risk has been mitigated through server side validation and sanitation built-in to the Falcon framework.

At the end of the day, there is no way to secure a web app against all attacks. However, there are some simple steps to eliminate many, if not, most attacks that may occur in the real word. In our case, these precautions have been taken, and it would take a serious effort to break through our defenses.

11 Testing

The main testing approach that the development team took was formal specification and verification. By writing down the requirements thoroughly, once implementation was done we were able to go back and formally verify the team's work to ensure correctness and lack of deviation from the as-intended requirements and design. The two formal verifications that were done for Sprint 2 and Sprint 3 showed that the design implementation was carefully followed and no defects were identified.

The development team also attempted to perform load testing on the API, as it was interesting to see how many requests it could handle. Unfortunately, there was an issue when using the free trial of the tool loader.io and all the requests returned failure, although successfully redirected. The team was not able to look further into the issue due to time constraints with delivery and privatization of other testing activities. A visualization of the report provided can be seen in Figure 5.

Testing was probably the most difficult task for the keyword extraction methods, as we were trying to gage which method performed best. The approach that was taken involved implementing a Reddit scraper, that would scrape news articles and apply all keyword extraction methods implemented and compare the keywords that were generated to the words in the title.

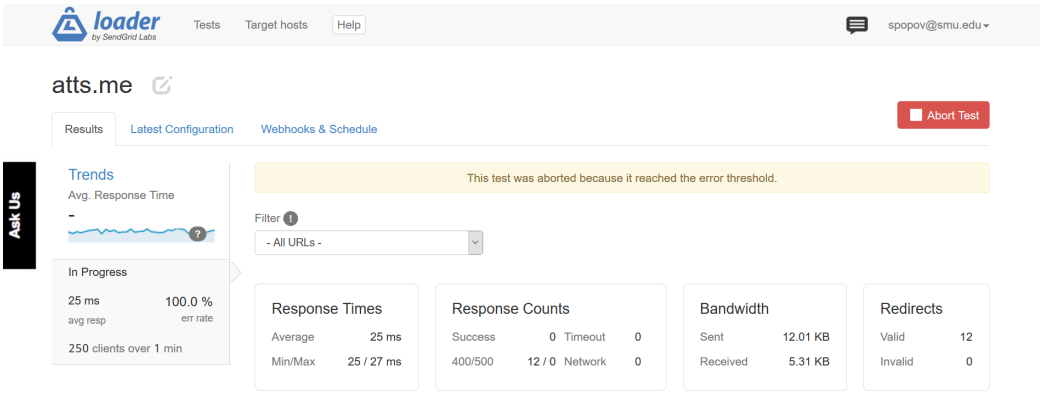

Figure 5: Load test results provided by loader.io

This testing approach was made on the assumption that news articles from trusted sources have enough relevant information for us to be able to compare our approaches accurately. The test cases, data, results and reports for this section can be found at the specifically created testing project and repository `https://github.com/browlm13/nlp_project`.

12 Maintenance and Evolution

As the Client Server architecture chosen for the development of this project allowed for separation of concerns and ease of scalability, maintaining the project should be fairly simple, for the development team. Furthermore, the fact that we have extensively documented our system allows us to verify it against the requirements and fix any issues, or update the documentation as needed.

If we had more time to work on this project and possibly when working on this project outside of the coursework of the Fall 2017 semester, we would focus our attention on improving the verification technique for the text keyword extraction and summarization functionality, as well as adding the intended functionality from Sprint 3 of the initially planned deliverables.

13 Conclusion

At its core, our product is designed to optimize the content creation and management workflow. We developed an open source project with wide-ranging community support from other open source authors as the foundation of our product and ended up with a powerful tool to add value to the work of media outlets. Although our initial plans were not all successfully met, we hope that in the future we will be able to expand on this project and provide the functionality we intended initially, as well as add more to it.