A Part of Speech Tagger for Software Documentation

Design Document

Team 35
Ali Jannesari, Hung Phan
Joseph Naberhaus
James Taylor
Austin Boling
Ekene Okeke
Ahmad Alramahi
Ethan Ruchotzke
sd21-35@iastate.edu
https://sdmay21-35.sd.ece.iastate.edu/

Revised: 10/1/2020

Executive Summary

Development Standards & Practices Used

ISO-IEC: ISO 12207 - Software Life Cycle: Used to model our project plan.

ISO-IEC: ISO 9001 - Quality Management: Used in conjunction with Agile Software development to ensure our product meets the client's specifications.

ISO-IEC: 15504 - SPICE: Used to asses the software development processes we have and will setup

Summary of Requirements

Study elements of standard POS tagger

Study new elements of software documentation

Implementation of a new POS tagger

Evaluation of the new POS tagger

Applicable Courses from Iowa State University Curriculum

Com S 227 Object Oriented Programming

Com S 228 Introduction to Data Structures

Com S 311 Introduction to the Design and Analysis of Algorithms

Com S 472 Principles of Artificial Intelligence

Math 207 Linear Algebra

Stat 330 Probability and Statistics for Computer Science

New Skills/Knowledge acquired that was not taught in courses

Python

Web scraping

General NLP Theory (for most in group)

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Our group would like to acknowledge our TA Hung Phan and our professor Ali Jannesari for giving us this project, as well as pointing us in the right direction to begin work on it. We would also like to acknowledge the Stanford NLP Group, who worked on the NLP, making this project possible.

1.2 Problem and Project Statement

As it stands right now, there isn't a tagger software that can analyze software documentation well. While the base NLP can do this to some extent, it isn't up to the task, as it doesn't fully understand the context of software documentation.

Our solution is to use the learning tools provided by the Stanford NLP group to augment their existing NLP model to meet the requirements that we have laid out. This includes creating a plethora of training data to give the NLP examples of correctly tagged software documentation. Further details of how we will augment the NLP will be found below.

1.3 Operational Environment

According to the Stanford Natural Language Processor Group, running a trained model requires at least 100 MB of memory, but in some cases can require upwards of 7 GB of memory ("The Stanford NLP", 2020). Training a model requires at least 1 GB of memory, but in many situations will require significantly more. Based on our current experiences, having at least 8 GB of memory and a reasonably modern processor is recommended ("The Stanford NLP", 2020).

1.4 Requirements

Corpus of tagged software documentation

- Collect a variety of forms (> 4 types) of software documentation in large quantities (> 25 of each)
- Tag the data in the same schema as the treebank used by the Stanford NLP
- Ensure the data is usable for future works

Augmented Stanford NLP model for software documentation

- Improve accuracy of base Stanford NLP model when run against english within software documentation
- Expand tag set of base Stanford NLP to cover common elements of software documentation
- Build Java and Python APIs for the new model
- Be capable of running on a mid-range machine with 8 GB of Memory and a mid range processor

Paper covering our work

- Produce accuracy comparisons between our model and the base Stanford NLP (both with normal english text, and software documentation)
- Follow writing standards used by Computer Science academia

1.5 Intended Users and Uses

The Intended users of this product are for the people who want a software that understands Natural Language (Stanford NLP Speech Tagger).

Part-Of-Speech Tagger (POS Tagger) is to read text that is given as an input in a Natural language and assign parts of the speech to a word (for an example, noun, verb,and adjective). Some of the uses of this software are as follows:

- useful for building parse trees
- extracting the relationship between words
- important in building lemmatizers which are used to reduce word to its root form

1.6 Assumptions and Limitations

Assumptions:

The inputs to this NLP will be in plain text

It will only be used for English software documentation

The architecture of the existing Stanford NLP is a good base to build our tagger off of

The model will be run locally on a user's personal computers

Limitations:

The model must be runnable on an ordinary mid-range computer (to match performance of existing models)

We will not be augmenting other capabilities of the Stanford NLP model such as the dependency grapher, NER, ect… (per user's requirements)

Users must be familiar with Java or Python to make full use of our model (per user's requirements)

This model will be optimized specifically for software documentation, and not for general NLP (per user's requirements and to improve technical feasibility)

No financial expenditures are expected for this project

1.7 Expected End Product and Deliverables

Corpus of tagged software documentation (November 2020)

Unlike the other two end products, this one covers two of the deliverables given to us by our client:

- Study elements of standard POS tagger (October 2020)
- Study new elements of software documentation-(November 2020)

A collection of software documentation that has been scraped from the internet and manually tagged. All types of documentation deemed relevant and interesting for this project will be included. Additionally, the tagging scheme of the corpus complies with the treebank schema used by the Stanford NLP model.

Implementation of a new POS tagger (February 2021)

A NLP processor tuned for software documentation. It has an expanded tag set that covers the common elements of most programming languages (with some bias towards Java). Internally, it works similarly to the existing Stanford NLP model, but with slight modifications. It's performance will be evaluated and reported along with the release.

Evaluation of the new POS tagger (May 2021)

An academic paper covering the methodology, and results achieved in our project. The paper discusses the basic architecture of the Stanford NLP and what modifications we have made to it. It will also include information about how we gathered a corpus of data to train the data. Finally, it will conclude with the performance and accuracy of our model.

2 Project Plan

2.1 Task Decomposition

The tasks outlined below are the general process the team will be taking in order to complete an additional module to the Stanford NLP processor. The team is following an Agile methodology, and the milestones below are not necessarily linear. In the event of a dependency, it will always be backwards, however a milestone's performance will determine the next milestone undertaken. This is evident in the schedule portion of the design document and the descriptions of tasks.

Major Milestones

1. Documentation Collection and Testing
2. Decision on Methods for NLP Extension
3. Implementation of Methods
4. Evaluation and Acceptance Testing
5. Reporting

Milestone 0: Setup and Testing [1]

- ❏ Complete Installations of Java CoreNLP and Python's Stanza
- ❏ Compile a list of test phrases for initial testing (English)
- ❏ Perform pipeline testing on both Java and Python NLP
    - ❏ Generate comparable outputs in a CSV format
    - ❏ Compare outputs of the pipeline for both versions
    - ❏ Determine next steps from the amount of consistency between platforms
- ❏ Make a decision on whether to use Python or Java's processor
- ❏ Generate documentation related to the extension and understanding of NLP
    - ❏ Make demo documentation for the group's usage

Milestone 1: Determine Types of Software Documentation [1]

- ❏ Find multiple examples of software documentation (English, and code)
- ❏ Categorize software documentation based on similar qualities
- ❏ Generate at least 3 categories of software documentation

Milestone 2: Collect Software Documentation [1]

- ❏ Find at least 5 examples of software documentation in each category
- ❏ Design a Web Scraper to take software documentation and output it in an easily readable format
    - ❏ Output the raw HTML data
    - ❏ Output the documentation segments

Milestone 3: Clean and Pre-process Software Documentation [2]

- ❏ Write a parser to generate “blocked” code based on whether the data is in monospace font or not
    - ❏ “Blocked” code distinguishes between code snippets and English text
- ❏ Take scraped documentation and generate treebanks based on the data.
- ❏ Use Word2Vec to generate training data based on treebanks

Milestone 4: Complete initial analysis of Software Documentation [2]

- ❏ Pass the raw software documentation through the existing English NLP pipeline
- ❏ Analyze the data to find common errors which need to be fixed
    - ❏ Find common errors in *English* documentation
    - ❏ Analyze behavior of NLP on code snippets

Milestone 5: Strengthen and Deepen Tags Related to Software [3]

- ❏ Analyze common errors to determine additional tags which may benefit POS analysis
- ❏ Come up with use cases for the new tags, along with several examples of their usage

Milestone 6: Manually Tag Documentation with New Tags [3]

- ❏ Test out new tags using the corpus of software documentation
    - ❏ Iterate on tags as necessary until a good model is found
- ❏ Manually tag all software documentation with new tags
    - ❏ Initially tag documentation with Stanford NLP
    - ❏ Correct Stanford NLP errors using manual tagging

Milestone 7: Train a new Markov Model [3]

- ❏ Put together corpus of training data
  - ❏ Use Treebank format to convert raw documentation data
  - ❏ Use Word2Vec to generate training data
- ❏ Use Stanford's CoreNLP library to train a new Markov Model
  - ❏ The new markov model will be subject to acceptance testing
  - ❏ This stage will be iterative as necessary

Milestone 8: Acceptance Testing [4]

- ❏ Perform acceptance and accuracy testing on a separate set of data from the training data
- ❏ Does the model conform to the accuracy standards set out earlier? If not, revert to milestone 5 and iterate.

Milestone 9: Reporting [5]

- ❏ Compile data on accuracy of tested data.
- ❏ Participate in the creation of a research paper related to the project.

2.2 Risks And Risk Management/Mitigation

Because the team is undertaking the project with an Agile methodology, risk is a minor factor due to the fast iteration process. However, risk is still a factor. Below are some of the most important risk factors for the project, and our methods for mitigating the risk. In total, however, Agile development and rapid iteration will allow risk to be mitigated for this project.

Risk Factor 1: Inability to train a model which is acceptable [5-10%]

The inability to train a model which meets acceptance testing standards is the major risk factor associated with this project, however Agile development essentially mitigates this risk for the project. Development of an acceptable model is vital, but iterative. So, even if the model does not meet acceptable accuracy requirements on the first iteration, there will be plenty of sprints available to increase the accuracy.

Risk Factor 2: Inability to create a large corpus of tagged documentation [3%]

The most vital part of machine learning is the creation of a corpus of data. In this project's case, it is possible that the team will be unable to generate a large corpus of data. In the event this happens, the mitigation is simple - scrape for a broader base of documentation. While initially we are only requiring 3 types of documentation, increasing our number of documentation categories will allow more documentation to be available for training, meaning the corpus will be expanded.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

At least 4 different types of software documentation will be identified

At least 25 instances of each type of software documentation identified will be collected.

At least 25 instances of each type of software documentation will be cleaned.

At least 25 instances of each type of software documentation will be run through the standard POS Tagger to see discrepancies

List of new tags relating to Software Documentation will be finalized.

At least 25 instances of each type of software documentation will have their tags modified manually to match expected

Modified MEMM will be trained with at least 25 instances of each type of manually tagged software documentation

Modified StanfordNLP Tagger will tag Software documentation with additional tags at 90% accuracy

2.4 Project Timeline/Schedule

Figure 2.4.1 below shows the predicted project timeline. Please note that the details of implementation are unknown, so it is just one long 2 month block. Additionally, names of members are not included because we do not yet know who will exactly be doing what.

It can be seen that we plan to be doing mostly information and data gathering over this first semester. This is the most essential part of the project, since we cannot produce a good model without first having good data.

During the next semester, we will start producing models and evaluating them. This will be an iterative process until we are satisfied with the results. Finally, we will release our finalized model, and produce a written report about it.

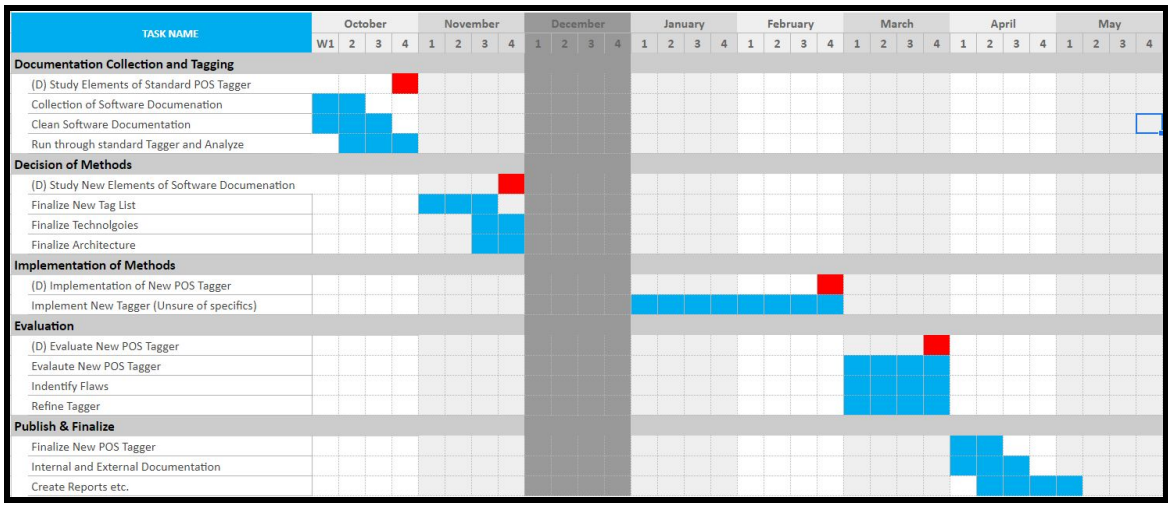

Figure 2.4.1 Project timeline (Deliverables shown with a red box)

2.5 Project Tracking Procedures

Tracking procedures/technologies:

Weekly Task Management (Sprint Boards) - **Trello**

Project Version Tracking - **git** repository hosted on **GitLab**

Daily Interaction, Sharing, Teamwork - **Discord**

2.6 Personnel Effort Requirements

Include a detailed estimate in the form of a table accompanied by a textual reference and explanation. This estimate shall be done on a task-by-task basis and should be the projected effort in total number of person-hours required to perform the task.

Task	Textual Reference	Time to complete	Explanation
Documentation Collection and Tagging	The task was distributed as follows: ● Study Elements.... ○ Java- Ahmad and Ekene ○ Python-Ethan and Joseph ● Collection of Software Documentation ○ James and Austin ● Clean Software Documentation ○ James and Austin	4 weeks	● Study Elements of Standard POS Tagger ● Collection of Software Documentation ● Clean Software Documentation ● Run through standard Tagger and Analyze
Decisions of Methods	TBD	4 weeks	● Study New Elements of Software Documentation ● Finalize New Tag List ● Finalize Technologies ● Finalize Architecture
Implementation of methods	TBD	8 weeks	● Implementation of New POS Tagger
Evaluation	TBD	4 weeks	● Evaluate New POS Tagger ● Identify Flaws ● Refine Tagger
Publish and Finalize	TBD	5 weeks	● Finalize New POS Tagger

			• Internal and External Documentation • Create Reports

Table 2.6.1 Personal effort requirements

2.7 Other Resource Requirements

To extend the Stanford NLP for software documentation tagging, our team requires relatively few resources other than machines and data that our team already possesses or can easily acquire. These resources are broken down into three categories: physical, virtual, and data.

Physical Resources:

- ❏ Mid-range machines for development (modern laptops)
  - ❏ Capable of running a program which may take up to approximately 1 GB of main memory
  - ❏ Most modern processors are more than capable of the development task

Virtual Resources:

- ❏ Mid-to-high-range machines for model training (university computing resources)
  - ❏ Capable of running a program which may take up to approximately 7 GB or more of main memory
  - ❏ Most modern processors are more than capable of the training task, but a high-end processor would drastically reduce the time needed to train
  - ❏ The Stanford NLP is able to use the GPU for training, so a powerful GPU would also drastically reduce the time needed to train

Data:

- ❏ Software documentation of various types from various sources for training
  - ❏ Documentation includes but is not limited to:
    - ❏ Library and package documentation
    - ❏ Documents that mix both natural language, code, and/or pseudo code
    - ❏ Natural language that describes code
  - ❏ Must already be virtual
- ❏ NLP models, documentation, and research

2.8 Financial Requirements

There are no financial requirements necessary for this project.

3 Design

3.1 Previous Work And Literature

Include relevant background/literature review for the project

– If similar products exist in the market, describe what has already been done

– If you are following previous work, cite that and discuss the **advantages/shortcomings**

– Note that while you are not expected to "compete" with other existing products / research groups, you should be able to differentiate your project from what is available

Detail any similar products or research done on this topic previously. Please cite your sources and include them in your references. All figures must be captioned and referenced in your text.

3.2 Design Thinking

Detail any design thinking driven design "define" aspects that shape your design. Enumerate some of the other design choices that came up in your design thinking "ideate" phase.

3.3 Proposed Design

Include any/all possible methods of approach to solving the problem:

- Discuss what you have done so far – what have you tried/implemented/tested?
- Some discussion of how this design satisfies the **functional and non-functional requirements** of the project.
- If any **standards** are relevant to your project (e.g. IEEE standards, NIST standards) discuss the applicability of those standards here
- This design description should be in **sufficient detail** that another team of engineers can look through it and implement it.

3.4 Technology Considerations

Highlight the strengths, weakness, and trade-offs made in technology available.

Discuss possible solutions and design alternatives

3.5 Design Analysis

– Did your proposed design from 3.3 work? Why or why not?

– What are your observations, thoughts, and ideas to modify or iterate over the design?

3.6 Development Process

Discuss what development process you are following with a rationale for it – Waterfall, TDD, Agile. Note that this is not necessarily only for software projects. Development processes are applicable for all design projects.

3.7 Design Plan

Describe a design plan with respect to use-cases within the context of requirements, modules in your design (dependency/concurrency of modules through a module diagram, interfaces, architectural overview), module constraints tied to requirements.

4 Testing

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or software.

1. Define the needed types of tests (unit testing for modules, integrity testing for interfaces, user-study or acceptance testing for functional and non-functional requirements).
2. Define/identify the individual items/units and interfaces to be tested.
3. Define, design, and develop the actual test cases.
4. Determine the anticipated test results for each test case
5. Perform the actual tests.
6. Evaluate the actual test results.
7. Make the necessary changes to the product being tested
8. Perform any necessary retesting
9. Document the entire testing process and its results

Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you have determined.

4.1 Unit Testing

– Discuss any hardware/software units being tested in isolation

4.2 Interface Testing

– Discuss how the composition of two or more units (interfaces) are to be tested. Enumerate all the relevant interfaces in your design.

4.3 Acceptance Testing

How will you demonstrate that the design requirements, both functional and non-functional are being met? How would you involve your client in the acceptance testing?

4.4 Results

– List and explain any and all results obtained so far during the testing phase

- Include failures and successes
- Explain what you learned and how you are planning to change the design iteratively as you progress with your project
- If you are including figures, please include captions and cite it in the text

5 Implementation

Describe any (preliminary) implementation plan for the next semester for your proposed design in 3.3.

6 Closing Material

6.1 Conclusion

Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.

6.2 References

List technical references and related work / market survey references. Do professional citation style (ex. IEEE).

6.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar data that does not directly pertain to the problem but helps support it, include it here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc,. PCB testing issues etc., Software bugs etc.

Works Cited:

The Stanford Natural Language Processing Group. Nlp.stanford.edu. (2020). Retrieved 4 October 2020, from https://nlp.stanford.edu/software/lex-parser.shtml.