FIT3162 Computer Science Project 2
Code and Test Report

Keyword Extraction and Summarization of News Articles

Matthew Khoo
29270294
School of Information Technology
Monash University Malaysia

Samuel Gan Han Yoong
30112419
School of Information Technology
Monash University Malaysia

Nicholas Yeo Wei Ming
29458021
School of Information Technology
Monash University Malaysia

Supervisor
Dr. Soon Lay Ki
School of Information Technology

Monash University Malaysia

Part 1: Code Report

1. Introduction

This project is a research intensive project, where the objectives do not necessarily depend on the quality of the user interface as well as the software as a whole. The main objectives from this project is to compare and test popular natural language processing (NLP) techniques. These NLP tasks are divided into keyphrase extraction, named entity recognition (NER), and summarization. These tasks are part of the main software, where yielding the output from each module is the main task of the overall software.

The development and execution of the software are organized into the pipeline shown below. Aside from the NLP modules, the software is specific for the use of the New York Times Corpus dataset, where data preprocessing is specific towards the XML file type, as well as the annotation scheme of the dataset XML tags. Therefore, several limitations are imposed towards the main software. Each NLP module is developed independently and compiled into a main script which sets up the models or techniques used by the respective modules and displays a simple graphical user interface.

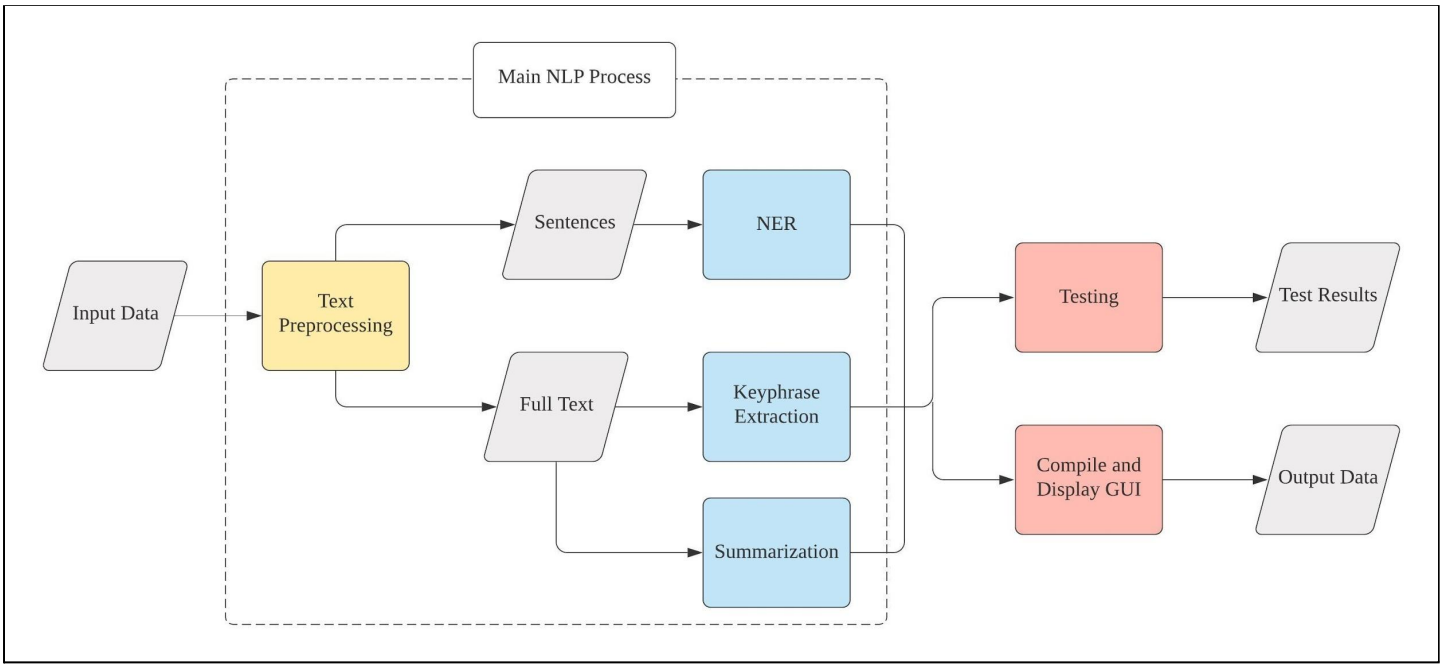


Figure 1.1 Main Software Pipeline

During the code demonstration, time is limited to explain the complete pipeline. Hence, software testing was removed from the demonstration, and the output from the NLP

modules became the main focus of the software. This report will cover all aspects of the software, such as software development procedures and principles, code quality, limitations, issues faced, potential improvements, further work, user guidance, and other matters that were not discussed in the code demonstration.

2. Source Code Overview

2.1 Deliverables

Since our project is a research project that intends to investigate and modify NLP algorithms to conduct keyphrase extraction, named entity recognition and text summarizations on the New York times dataset. We decided to produce separate modules that can be used independently for researchers to process the text. However, we recognized the need to cater to non-expert end users and thus provided a standalone application for said end users to apply the algorithms to the text using a simple gui.

Thus, we produced the following deliverables:

1. A standalone application that conducts keyphrase extraction, named entity recognition and text summarizations on the New York times dataset

   Located in dist

2. Separate NLP modules for Keyphrase Extraction (Pke_keyphrase.py), Named Entity Recognition(Spacy_NER.py) and Text Summarization (Nltk_sumarization.py)

   Located in the root directory

3. A series of preprocessing scripts including

   a. script that the researchers can use to filter the New York Times dataset for articles that fit their criteria

      Located in Preprocessing-> File_Filter_and_Copy.py

   b. a series of scripts that can be used to preprocess and prepare the files from the NYT dataset for further processing

      Located in Preprocessing

2.2 General Structure of Project

A general overview of the structure of the project. Descriptions of individual modules, class and function documentation, dependencies and installation instructions are found in the technical user guide and docstrings

Root Folder

Main scripts

- guiNews.py
- Nltk_summarization_nyt.py
- Spacy_NER.py
- Pke_keyphrase.py

Models

Justification of location of models is explored and explained further in the limitations section

- Kea-semeval2010.py3.pickle
- df-semeval2010.tsv.gz
- en_core_web_md
- en_core_web_md-2.3.1

Requirements

Explanation found in technical user guide

- Pipfile
- Pipfile.lock

Scripts to package application in standalone bundle

Explanation found in Code considerations ->portability -> standalone bundle section of this document and in technical user guide

- guiNews.spec
- hook-spacy.py

Preprocessing

Package to prepare texts for preprocessing

- File_Filter_and_Copy.py
- Text_Preprocessing.py
- XML_Parser.py
- XParser_Module.py

Models

- en_core_web_md-2.3.1
    - Model File. Justification is in software limitations section below

Tests

Description of tests as well as instructions on how to run tests found in test section

Data

- data
    - Data files for tests

Tests

- test_guiNews.py

- test_nltk_summarization.py
- test_Pke_keyphrase.py
- test_Spacy_NER.py
- test_Text_Preprocessing.py
- test_XParser_Module.py

Build

Folder created to store the temporary files for packaging a python application

dist

Location of standalone gui bundle

docs

Location of code and test documentation

2.3 Code Structure

The code is written as a compilation of scripts, where each script has independent context. This means that each script has only one purpose, which supports modularity of code in the software architecture. For example, the scripts for respective NLP modules are encapsulated into classes. Each class adheres object oriented principles such as reducing dependencies and encouraging reuse of code. The scripting style of the code supports the idea of simplicity through modularity. Dividing features into task-specific methods allows respective classes to reuse the code without worrying about side-effects.

The use of small modules allow compilation of code to be concise and easy to comprehend as each module has a complete set of methods to control all changes and returns within the object of respective classes. Each class does not rely on external functions or any piece of code outside its class to control the variables and data structures in the class. This practice of minimizing dependencies allows cohesion between modules, and supports data integrity within respective modules.

Immutability and integrity are strong concepts implemented in all modules, where variables from one script or class are immutable from any change from external code. The use of classes also allows the existence of multiple objects with varying instance variables. This is really useful to test the NLP modules with different configurations to compare and contrast. For example, the class of Spacy_NER_Model allows each object of the class to represent different models with different features. Another reason behind using classes is to isolate important attributes as instance variables, which allows the ease of tracking the state of attributes within each object. Hence, state management will be clear and organized, as less number of variables will be needed to manage the modules in the main driver code.

Furthermore, the methods in each class are structured in a way such that methods that return a value will always return the same value for the same parameter(s) passed. This supports the idea of pure functions which are more commonly found in functional programming. This concept allows the integrity between return value and parameters so that no arbitrary components are included in the function. This principle allows better software quality, as establishing pure functions makes debugging easier for developers, and guarantees a smoother process in software testing for input and output.

Once all modules are completed, everything is compiled into one main driver code that imports all of the modules as sub-routines into one script and constructs the software pipeline. Aside from compiling and structuring the modules, the driver code is also responsible for creating the simple GUI which will be utilized by the end user. Another important component includes an independent module which installs all the packages used by all modules into one directory to simplify the software setup.

2.4 Code Readability

The scripts have unique names to easily identify their respective uses. The names of classes are also relatively short and similar to the naming convention of the script. The scripts which serve as modules usually only have one class respectively to prevent confusion. The name of functions and methods are also fairly short and simple. The naming convention for classes, functions, methods and variables are either separated by underscore for each distinct word (e.g. “get_my_output”) or separated by uppercase letters for each word, where the first letter of the first word is in lowercase (e.g. getMyResults). On the other hand, script names are always separated by underscore for each word, and the first letter for each word is always uppercase (e.g. “Text_Preprocessing”).

The naming convention for variables is also short and meaningful, where it is easy for developers to identify what each variable represents. The number of variables is also kept to a minimum, where each variable references almost unique data. The use of temporary variables are also prevented as much as possible, as it causes issues with keeping track of the more meaningful variables, and temporary variables tend to have arbitrary names which may cause readability issues when someone else is reviewing the code.

Since all the code is written in Python, indents and whitespaces are consistent and structured cleanly, as Python is both indentation sensitive and whitespace sensitive. This means that Python will return an error if there are misaligned chunks of code, usually caused by the lack of or extra indentation and/or whitespace. To most developers, indents makes it easier to align code and identify the structure of loops and conditions within a portion of code. Nested loops and conditions tend to be more confusing with more levels, hence the use of functions to break long nested code is preferable.

Our code does not have many nested loops or conditions for each chunk, as we strive to create many smaller functions that are easier to read, as opposed to long pieces of code which contain many indentation levels. However, the use of “one-liner” codes are also prevented when it is unnecessary, because it is generally difficult to understand one long line of code which performs a lot of function calls or actions. Documentation will also become a problem for this type of code. This problem tends to arise in functional programming styled code, while our code adheres more of object-oriented principles and imperative style coding.

2.5 Code Documentation

All of the scripts are adequately documented to provide information regarding the overall script as well as classes, methods, and functions within the script. Classes are documented heavily to provide programmers information about important attributes (instance variables) and the use of each method. The documentation for classes rather summarizes the scheme of methods and attributes, where it relates back to the overall purpose of each class. The documentation for methods within each class focuses on the information of input and output where applicable, as well as the data used within respective methods.

In several classes, the documentation is not consistent on style and content, as the importance of displayed information varies. For example, the Spacy_NER_Model class, shown in the snippet of Appendix 1 shows detailed information about entity labels and how each label is processed, because that information is most important for NER. On the other hand, generic modules such as preprocessing has the common style of class documentation without any added information (ie. attributes and methods documentation).

Documentation for classes, methods and functions are considered high-level, and several lines or chunks of code usually need explanation, especially when an uncommon external library is widely used in those parts of code. Commonly, chunks of loops and conditions are well-commented to show what is being processed in a certain chunk. However, there are also cases where specific lines of code need comments due to the use of external libraries and/or to assert the importance of each of those lines.

For example, the chunk of code for sentence segmentation customization shown in Appendix 2 has multiple comments for small pieces of code. These comments are needed to show the additional rules on where the sentence should or should not break, because it is specific to the style of the text from the dataset. The heavy use of Spacy’s in-built functions and notations might be foreign to developers who have not read Spacy’s documentation. Hence, these comments will allow specific inspections towards the use case and features of those lines of code.

3. Code Considerations

3.1 Robustness

The application is made out of separate modules which functionality mainly depends on well maintained and extensively tested libraries. Thus, there is a low chance of a major bug occurring within the modules. Thus, the main vectors where failures could be introduced would be through faulty input and the interactions between modules. The following are the approaches that we took to reduce the risk of those occurrences happening.

Input Validation

Within the graphical user interface, warning messages will pop up if the user does not select the file and output directory before attempting to run the program. We have also restricted the type of files chosen to minimize user error.

We added checks within the code to verify the file type before the main program processes the files. This is done by verifying the extension of the files and by ensuring that at each step through the process, the files are the right type and of the right format.

Module independence

Each module is functionally pure and does not affect the execution of other modules. Thus, even if one module fails, other modules would be able to output data as usual.

3.2 Scalability

The modular design of our application means that our application can be easily scaled up to handle a vast amount of input. It would also be able to be integrated with other applications as standalone modules. We also have written extensive tests for each module, meaning that it would be possible to verify the modules when they become part of a larger or a different application.

The gui is currently not coded to accept more than a single article. However, there is no limit to the size of the article, the only constraint is time, as a longer article would take a much longer time to process.

3.3 Portability

The software is developed in MS-Windows and MAC OS operating systems. There seems to be no significant problem with migrating code between these OS and the packages used for each module also do not have any portability problems. Although there are no issues for these operating systems, there might be some possible minor issues that need to

be addressed when the software application is run on Linux OS as we did fully test the software on Linux.

The reason for the excellent portability of our code is because Python code is to a large extent operating system independent. This is because it is an interpreted language: the python code we write gets compiled to bytecode and it is run by the python virtual machine. As long as there is a python virtual machine implemented for a certain platform, our program would be able to be run on said platform. This is one of the main reasons why we choose to develop our code in python.

We also have made a point to avoid the use of operating system specific functions/ paths when writing our code. For example, instead of hardcoding our file paths, we used python's os.path.join to deal with the different ways that file paths are expressed in different operating systems, with Microsoft windows using backslash characters and Linux and Mac Os using forward characters.

One major weakness however, is that we make very extensive use of software libraries. Therefore, this is made less of an issue by the use of Python's pipenv, a tool that creates virtual environments from a single file of dependencies, called a pipfile. However, creation of virtual environments is still tedious, time consuming, and might take up a lot of space on the local machine.

Standalone Bundle

Our solution is to package all the necessary libraries into a standalone bundle with the use of Pyinstaller. This means that all a user would need is the standalone bundle packaged specifically for their operating system. This brings up the one issue with this method:

Issue	Description
Specialized compilation needed to different operating systems	We used Pyinstaller to compile our standalone application bundle. The advantage of this method is that Pyinstaller is only able to compile a bundle for the operating system that it is being run on. Our main development was conducted on a Macbook, thus we only provided a bundle for use on the Mac Os operating system. However, we would be able to produce bundles for different operating systems on request.

However, compared to even pipenv, the standalone bundle greatly increases the portability of our program.

4. Software Limitations

Module	Limitation
Data Preprocessing	- Filter only works for NYT Corpus dataset.

	- Limited filtering option. - Parser only for XML files. - Parser only for NYT Corpus annotations.
Keyphrase Extraction	- Unreliable evaluation against NYT Corpus descriptors.
Named Entity Recognition	- Entity labels limited to who, where, when. - Unreliable evaluation against NYT Corpus named entity annotations.
Summarization	- ROUGE scores unreliable against NYT Corpus abstracts.

4.1 NYT Corpus Dataset

As shown on the table above, the New York Times (NYT) Corpus Dataset causes several issues with the processes in NLP modules, particularly the problem with evaluation of statistical techniques and models. The NYT Corpus dataset is a general news article dataset with annotations. These annotations do contain useful data such as named descriptors, named entities, and abstract which align with the objectives of keyphrase extraction, NER, and summarization respectively. Unfortunately, most of these annotations are abstractive, which means that it doesn't use the words or phrases from the original full body text of the news article.

This is a big issue for the NLP modules since this project's scope is to use the more common extractive methods instead of abstractive to produce the output. Hence, the annotations that were seemingly useful cannot be utilized for validation data. This causes issues with testing and validating the techniques and models used by the NLP modules, which prevents yielding proper data for evaluation metrics such as precision, recall, and F1 score. Therefore, different datasets are needed for the evaluation of each NLP module to replace the unsuitable annotations.

4.2 Data Preprocessing

The preprocessing module also had several limitations as shown on Figure 4 above. The data preprocessing code which filters and copies the dataset according to the chosen tag(s) is specific for the structure and annotation format of the NYT Corpus dataset. Since the use of XML files is not as common to other file types (e.g. JSON), the data preprocessing module can only filter one specific dataset.

Another specific limitation would be the mechanism of filtering the files. The code allows the user to input an array of XML tags to allow the filter to search for files that have all

of the defined tags in the array, and copy them. However, the code is limited to only search with all the tags, instead of being able to choose other logical operators. For example, if the search list contains the tags “a”, b”, “c, the filter will only find the files with the tags under the expression of “a” and “b” and “c”. This is a distinct limitation in case the user wants to search with either tags in the search list or find the files without certain tags. In short, the filter code only supports the “and” logical expression by finding all tags in the search list, and cannot perform more flexible searches using the “or” and “not” logical operators.cv

The file parser also shares similar problems as the data preprocessor. The code for extraction of annotations is limited to XML tags with the NITF annotation format, which is not exactly common for publicly available news datasets. Hence, the parser cannot be used for other datasets when it is needed. Since this project is using other datasets for statistical evaluation purposes, the parser is not universal, which means that the NLP modules that use other datasets will need independent parsers for preprocessing. This limitation increases the workload of developers by the necessity to create independent parsers.

4.3 Code Structure

The models used by Spacy are placed in the main directory instead of in a specialized sub directory. Our code makes use of various functions that are called from various libraries that all require the models. Having the models in the main directory makes it straightforward for the libraries to locate the models.

The libraries look for the models in the follow locations:

1. The default location for spacy model files
2. The root directory

We have decided that it is a reasonable to tradeoff stringent code structure for ease of locating the models. This is because some libraries we use import and import spacy models directly, with no easy way to modify the default location of the models. The alternative is to rewrite the libraries, which we judged as not being worth the cost as modification to complex libraries would necessitate the creation of our custom test bed that would take up a huge amount of time, and would lead to deployment bugs that we might not be able to identify.

5. Potential Improvements

Further modification and customization of Graphical User Interface

Integration with other

6. Future Work

For the software, the only work left that needs to be done is testing and evaluating the techniques used for each NLP module. This includes yielding common evaluation metrics for statistical models such as F1 score, precision, and recall. Different modules use different models, hence there are specific evaluation metrics that can be used. For example, summarization uses the Text Rank method and ROUGE score is a specific evaluation for the module. Our results from evaluation will be used as data for visualizations, which are critical for the project research. As research is the main point of this project, evaluation of these models will be the most important part of the NLP modules, where it yields the information needed for the final research outcome of this project.

As stated under limitations, the NYT Corpus dataset is not suitable as test or validation data because of its abstractive annotations. Hence, each NLP module will need different datasets which are specifically annotated for respective tasks to perform evaluation on the techniques and models used by each module. Although there is no standard dataset for each NLP module, all of the datasets are from news corpora, so that the scope of this project is still within the domain of news articles. These datasets will provide test and validation data needed for proper evaluation of the NLP modules' approaches. Then the obtained evaluation data will be used for comparisons against other techniques proposed by researchers, and construct useful discussions from the results.

7. End User Guide

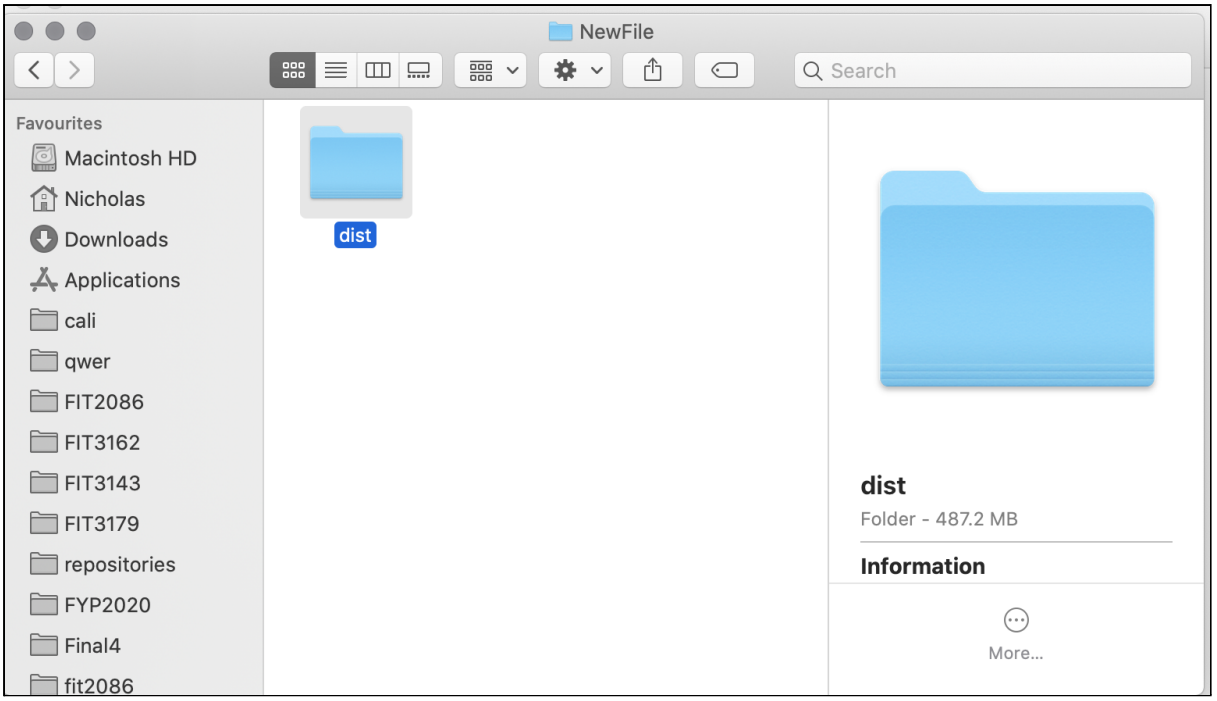
We Provide 2 main methods of using the graphical user interface.

I. By using the pre-bundled package

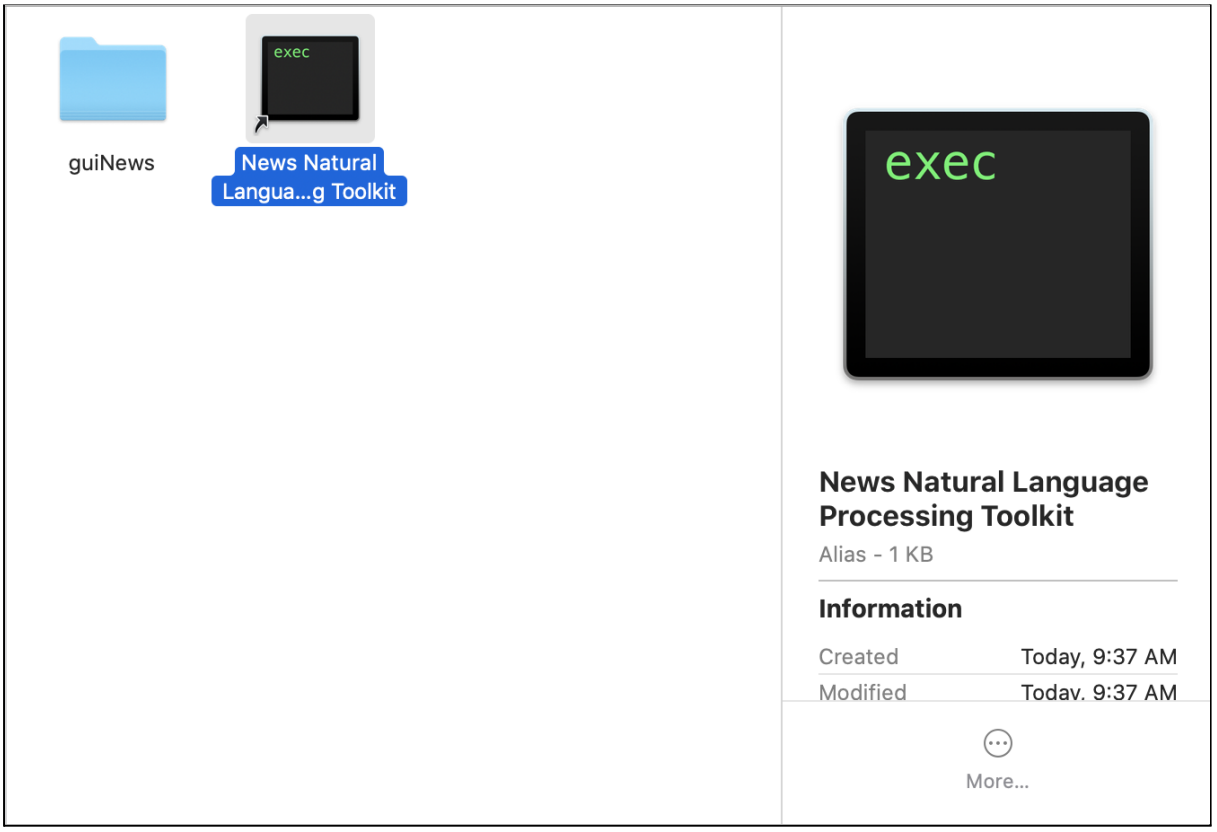
II. By installing the necessary packages through Pipenv through a pip file that we provide. The details of the second method would be expanded further in the technical user guide

Setup

1. Place code bundle into preferred directory. Code bundle is obtained from
    a. Root Directory-> dist

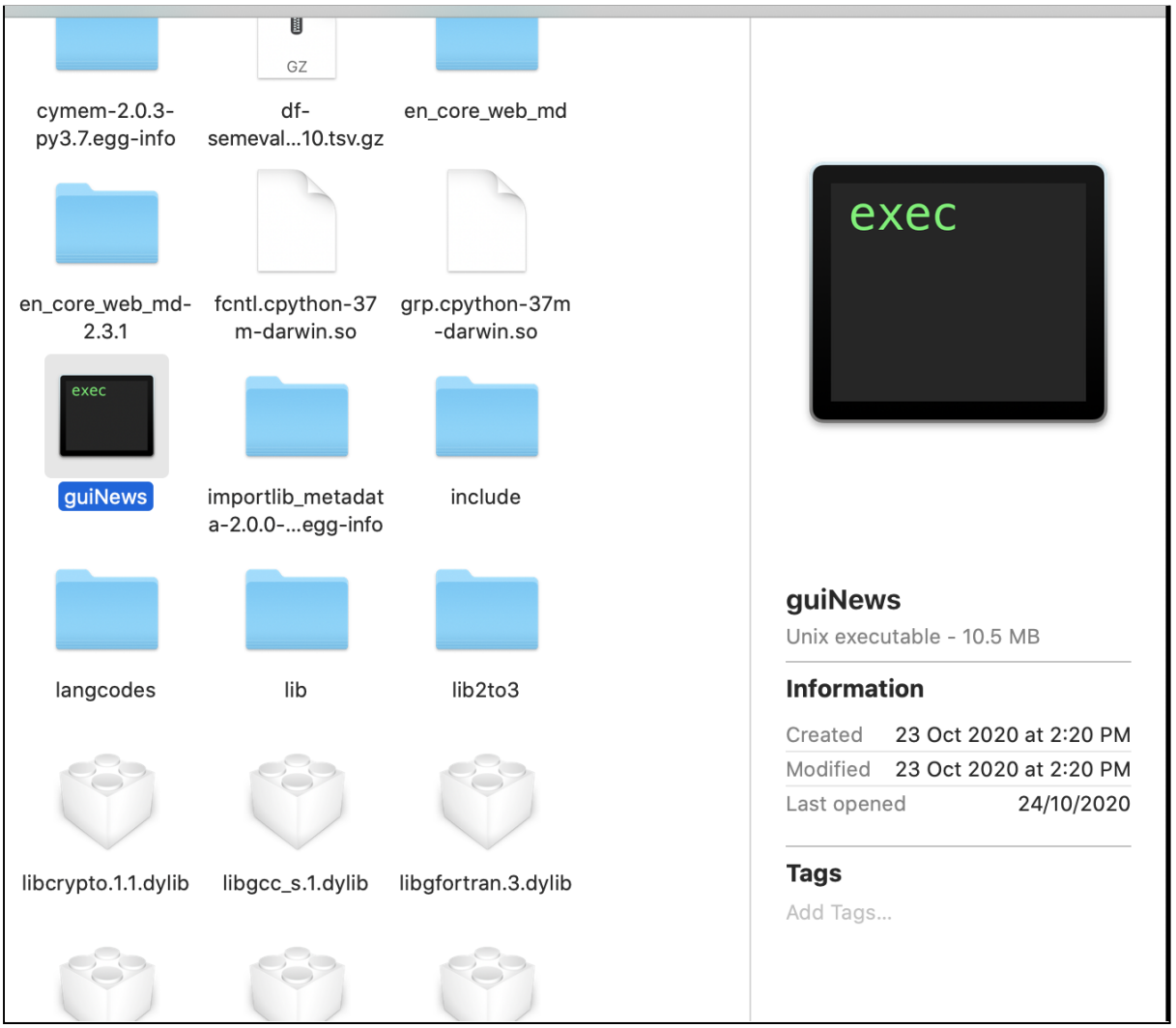


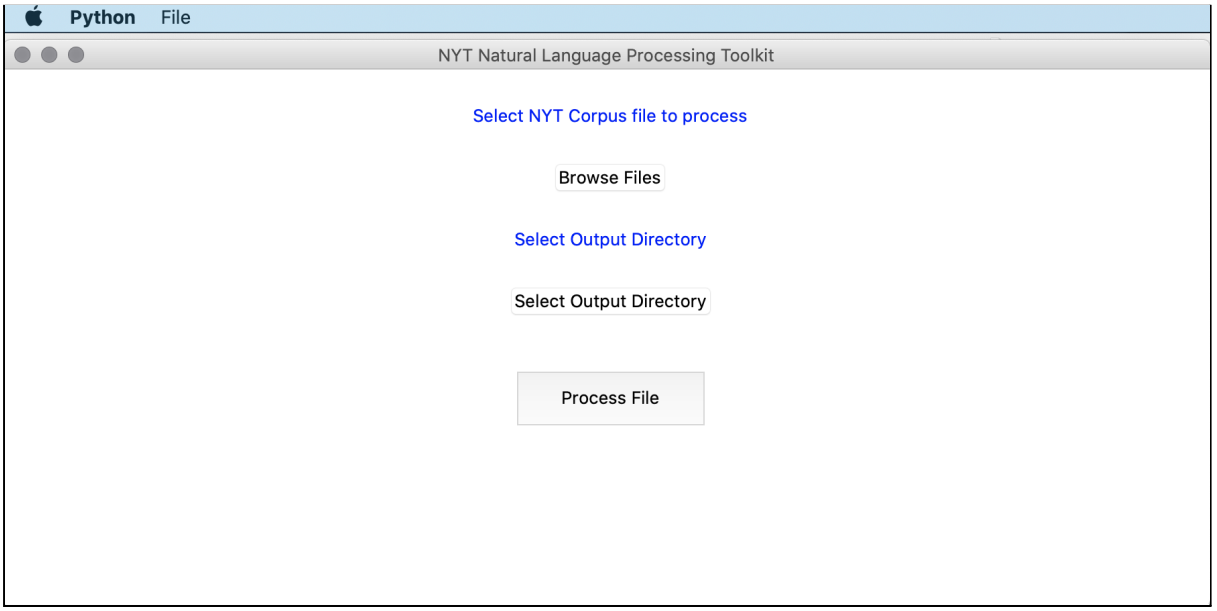
2. Either:
    a. Double click the exec file within the bundle named “guiNews”



    b. Or double click the shortcut

3. Wait until Graphical User Interface window loads

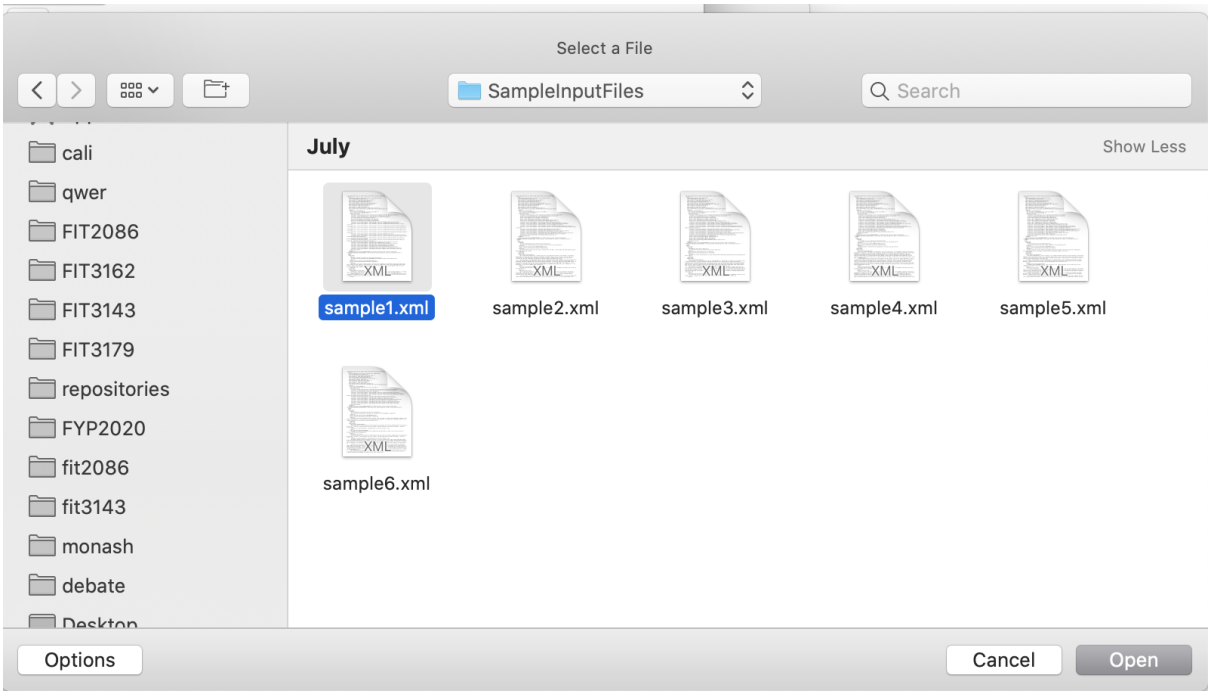




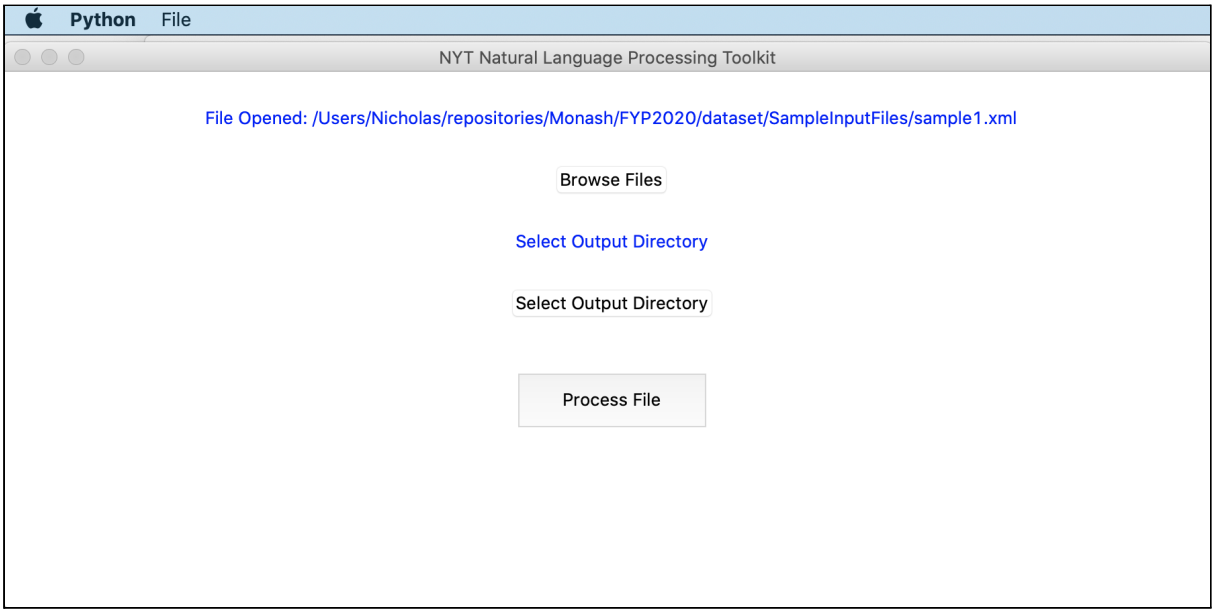
Basic Usage

4. Browse to NYT corpus file that you would like to process by either:

    a. Clicking on the browse files button
    b. Or or through the file menu: File->Open



5. The location of the selected file would be displayed in the window. Verify that the file selected is the correct file

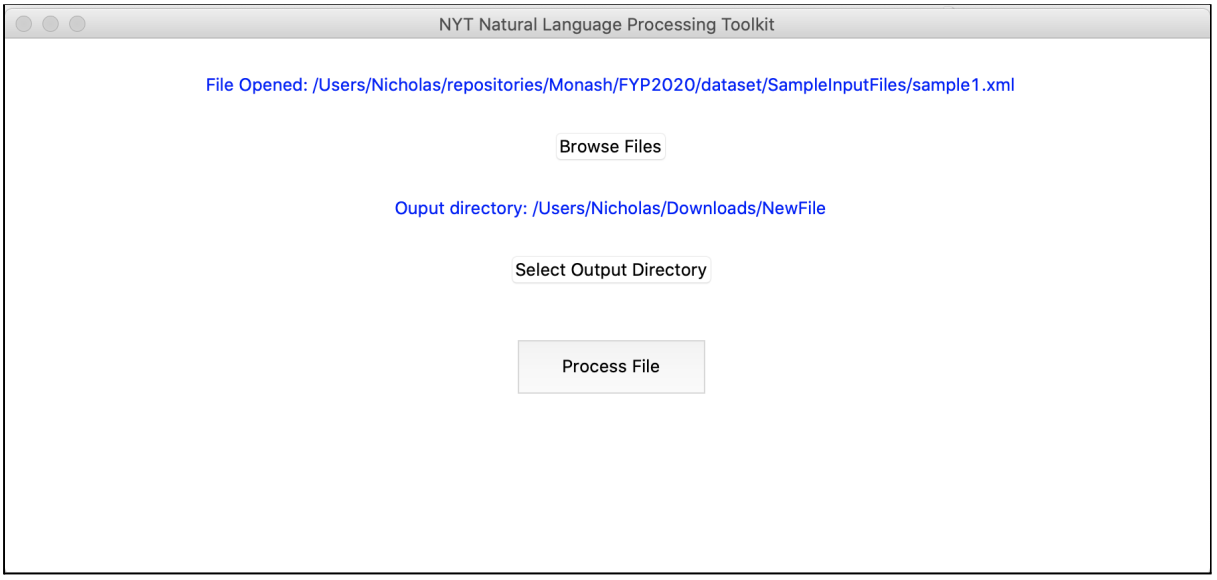


6. Browse to the output directory that you would like to save the output files to:

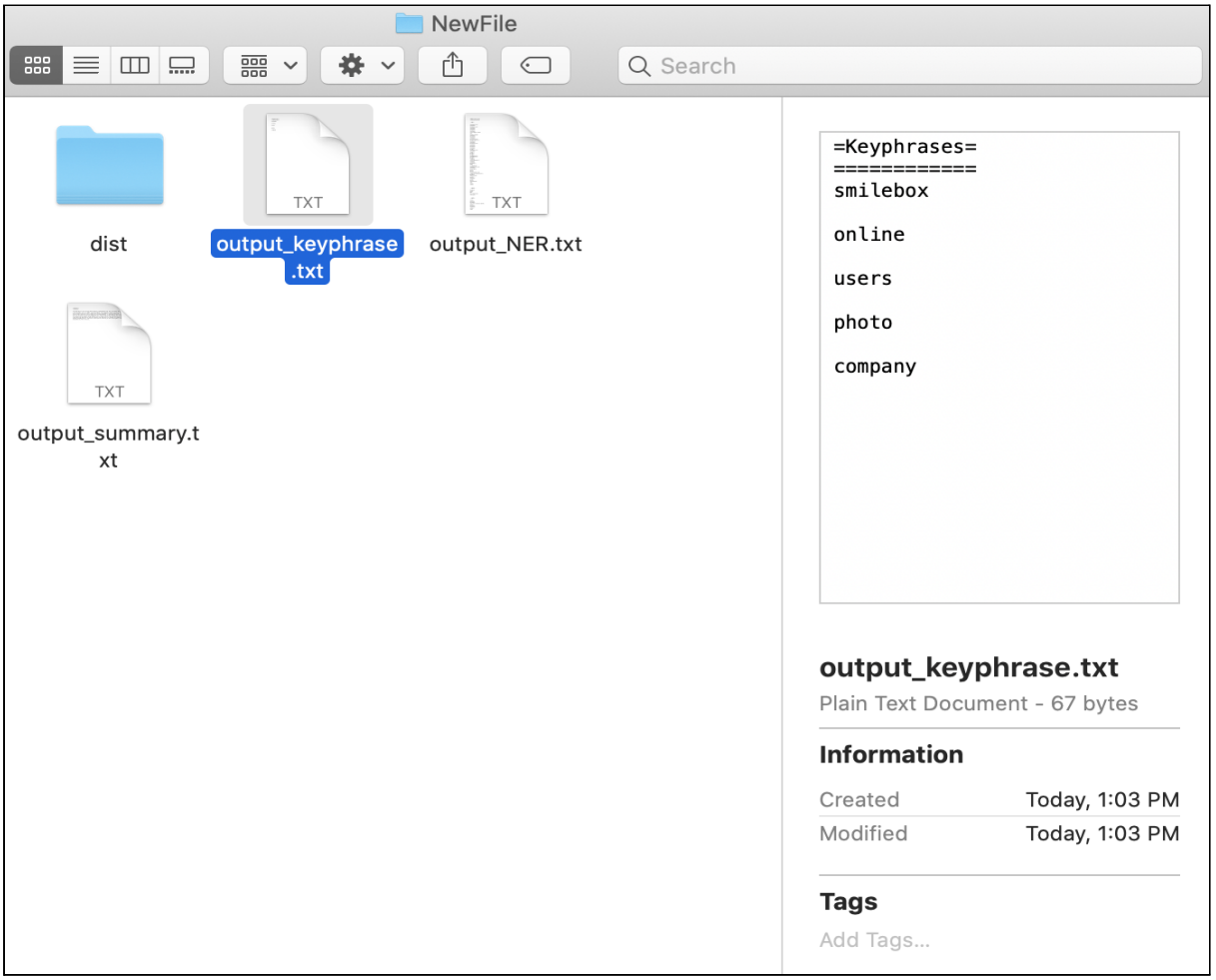
    a. Clicking on the Select Output Directory button

b. Or or through the file menu: File->Output Directory

7. The location of the selected directory would be displayed in the window. Verify that the selected directory is the desired directory



8. Click Process File
9. The 3 files would be outputted to the selected directory



Troubleshooting

1. If all else fails, contact the developers:
    a. **Instructions**: Copy the contents of the terminal output, Operating System Version, and the Current Version of the News Natural Language Processing Toolkit used and email to nyeo0001@student.monash.edu

8. Technical User Guide

8.1 Installation

Note the installation instructions were tested on a Mac, Windows and Linux operating system

8.1.1 Obtaining the files and dependencies

1. File can be obtained by cloning from github

    git clone https://git.infotech.monash.edu/bombompowfyp2020/projectDesign.git

2. Note: we use *Pipenv* to manage our dependencies. If you do not have *Pipenv* installed, install through pip by running the following command in terminal (Mac Os, Linux) or Powershell (Windows). Pip installation instructions can be found here: Installing Packages

    pip install pipenv

3. Our Graphical user interface requires a python3 version installed with tkinter. If you are unsure if your python3 supports tkinter, the easiest way is to install tkinter using binaries from https://www.python.org/. You can test if tkinter is properly installed by running

    python3

    To activate your python interpreter, then running the following command:

    import tkinter

    If it proceeds without error messages, you are good to go.

4. Once you have *Pipenv* installed, you can install the dependencies directly from the pipfile using:

pipenv install

5. Activate the environment using

   pipenv shell

6. Run the guiUsing:

   python3 guiNews.py

8.1.2 Standalone binary

Our package comes with a pre-packaged bundle for MacOs. If you would like to package such a file for your Operating system, run

pyinstaller guiNews.spec

Then locate the executable file guiNews.exc within the folder produced in the directory, that is dist->guiNews->guiNews.exc. Double clicking on this would run the application.

8.2 Individual Modules

Preprocessing:

1. File_Filter_and_Copy.py

Packages	Instructions	Main Usage
- ElementTree class from etree library of Python's XML API.	1. Import xml.etree.ElementTree	- Process the named entities quicker and more accurately by passing a stream of sentences instead of full paragraphs. - Identify named entities from a sentence by classification of the pretrained CNN model. - Label each named entity with WHO, WHERE, or WHEN labels.

2. XParser_Module.py

Packages	Instructions	Main Usage
- ElementTree class from etree library of Python's XML API.	1. Import xml.etree.ElementTree 2. Locate the zip parent folder. 3. Filter news articles that contains an abstract tag. 4. Copy the filtered documents into respective directories.	- Extract the contents based on important annotations

3. Text_Preprocessing.py

Packages	Instructions	Main Usage
- SpaCy parser - en_core_web_md	1. Import SpaCy 2. Load the en_core_web _md package as the preprocessing model 3. Disable the tagger and NER features of the model's pipeline. 4. Now, only the rule-based parser will be used to perform tokenization on the textual data.	Tokenizer: - Tokenize textual data. - Remove semantically useless tokens such as stop words and punctuations. Sentencizer: - Segmentize paragraphs into sentences. - Customize the sentence segmentation according to the delimiters of the text.

Natural Language Processing:

1. Nltk_summarization_nyt.py

Packages	Instructions	Main Usage
- nltk	1. Import nltk 2. Load the desired xml path 3. Extract the full text and abstract from the xml 4. Outputs summary and rouge scores	- Outputs a summary that are sentences made from the original article. - Outputs a rouge score that measures the unigram/bigram and longest common subsequence

2. Pke_keyphrase.py

Packages	Instructions	Main Usage
-pke	1. Import pke 2. Load the pke model 3. Get keyphrases from each xml articles	- Identify the top n-th keyphrases desired.

3. Spacy_NER.py

Packages	Instructions	Main Usage
- SpaCy NER - en_core_web_md	1. Import SpaCy 2. Load the en_core_web _md package as the NER model. 3. Disable the parser and tagger features of the model's pipeline. 4. Pass in an array of sentences into the outputSentenceEntites as a stream. 5. Get the named entities from each sentence along with appropriate labels.	- Process the named entities quicker and more accurately by passing a stream of sentences instead of full paragraphs. - Identify named entities from a sentence by classification of the pretrained CNN model. - Label each named entity with WHO, WHERE, or WHEN labels.

9. Appendix

Appendix 1: NER Class Documentation

```
class Spacy_NER_Model:
    """
    Attributes:
    self.model: Spacy model
        The loaded model used for all processes within this class.
        Default model is en_core_web_md pretrained model.

    Relevant Entity Categories:

    PERSON:         People, including fictional.

    NORP:           Nationalities or religious or political groups.

    FAC:            Buildings, airports, highways, bridges, etc.

    ORG:            Companies, agencies, institutions, etc.

    GPE:            Countries, cities, states.

    LOC:            Non-GPE locations, mountain ranges, bodies of water.

    DATE:           Absolute or relative dates or periods.

    TIME:           Times smaller than a day.


    Aggregated into the following labels:

    PERSON, ORG, NORP -> WHO

    LOC, GPE, FAC -> WHERE

    DATE, TIME -> WHEN
    """

    def __init__(self, model = None):
```

Appendix B: Comments on Sentence Segmentation Rules

```
for i, token in enumerate(doc[:-2]):
    # prevents breaking a token with a comma, colon, semi-colon or starts with lowercase character.
    if token.text in (",",";",":") or token.is_lower:
        doc[i].is_sent_start = False
    # prevents breaking after the start of a quotation
    elif i > 0 and token.text[0].isupper() and token.nbor(-1).text == "'":
        doc[i].is_sent_start = False
    # breaks quotation mark after a full stop followed by space, but don't break after quotation
    elif token.text == "'" and (token.nbor(-1).text == "." or "'")) and token.nbor(-1).whitespace_:
        doc[i].is_sent_start = True
        doc[i + 1].is_sent_start = False
```

Part 2: Test Report

1. Software Testing Approach

Software testing serves as the measurement of the software's robustness, scalability, and limitations. The testing procedure for this project involves a series of blackbox testing across all modules within the software pipeline. Blackbox testing is an easy, but powerful procedure which only tests the input and output data without any regards for the code within the function. If the functions pass the test with flying colours, there is a very small possibility that there are major bugs which cause errors.

Generally, XML files are used as input data, and the input data for the other modules along the pipeline is extracted from the contents of the XML file, most notably the full body text. The NLP modules deal with string objects or a series of strings to process and extract useful information from text. Hence, testing the preprocessing modules is critical to ensure that the NLP modules can perform their required tasks properly. Since the contents of the NYT Corpus are relatively consistent across all files with abstracts, not many files are needed to test the consistency of the functions and methods.

Once all independent modules in the pipeline are tested, the overall software testing will be complete with integration testing. In this stage, the driver code which compiles all the modules will be tested together with the GUI which displays the whole software. The integration test is meant to assess the overall software pipeline and detect any errors between the interaction of the pipeline modules.

For the GUI, a section on user test shows qualitative and quantitative feedback from several users on the user-friendliness of the GUI and opinions on the functionality and features of the software. There are also several viable suggestions that could be used to improve the overall system.

2. Unit Tests

To verify that each of our individual models do work, we will be performing unit testing. This process is to validate the smallest testable parts of our code to perform as designed. To help us accomplish this, we will be using the python unittest testing framework.

All tests are located in the folder "tests". To run a particular test, simply run "python3 <test name>" . Note all required libraries must be installed. Check the technical user guide above for more details about how to install libraries.

2.1 Preprocessing

Test	Description
Module Tested	File: XParser_Module.py Class: XParser Method: getIndexDesc, getFullText
Test Goal	- Check proper XML tag path finding. - Check the output format of each method
Test Procedure	- Ensure that the parser finds the tag path for annotations of the NYT Corpus dataset files (i.e. path finding for NITF annotations). - Check whether the data type of output from getIndexDesc is always a set of strings or an empty set. - Check whether the data type of output from getFullText is always a string. - Ensure that the output from getFullText is never an empty string.
Input	Sample XML file taken from the NYT Corpus Location: tests/data/1815742.xml
Expected Output	- getIndexDesc returns a set of non-empty strings or an empty set - getFullText returns a non-empty string.
Result	Actual output matches the expected output and test passed.

```
(testProduction) Nicholas@NYWM testProduction % python3 tests/test_XParser_Module.py
Test Xparser passed!
```

Image: Actual test output

```python
# Importing modules to test
from Preprocessing.XParser_Module import XParser

# Getting path to test files
testPath = os.path.join('tests', 'data', '1815742.xml')

class test_Xparser_Module:
    def __init__(self, testPath):
        # Initialising the XParser object
        self.obj = XParser(testPath)

    def test_getIndexDesc(self):
        indexDesc = self.obj.getIndexDesc()
        sampleIndexDesc = {'and', 'Photography', 'Computers', 'Internet', 'Softw
        # Black box testing
        assert sampleIndexDesc == indexDesc
        # Testing if the the elements are strings
        for desc in indexDesc:
            assert type(desc) == str

    def test_getFullText(self):
        sampleFullTestOutput = '''FOR years, Internet users have treated online
        fullText = self.obj.getFullText()
        # Black box testing
        assert fullText == sampleFullTestOutput
        # Testing if the output is a string
        assert type(fullText) == str

if __name__ == '__main__':
    test1 = test_Xparser_Module(testPath)
    test1.test_getIndexDesc()
    test1.test_getFullText()
    print("Test Xparser passed!")
```

Image: Actual code of the test

Test	Description
Module Tested	File: Text_Preprocessing.py Class: TextPrep Method: sentencizer
Test Goal	- Guarantee the output format of the method return. - Check for abnormalities in the attributes of the output.
Test Procedure	- Ensure that the method returns a non-empty list. - Ensure that each element in the array is non-empty. - Check whether the list elements are only string objects.
Input	Full body text of a news article from a sample XML file taken

	from the NYT Corpus. Location: tests/data/1815742.xml
Expected Output	- The returned list is not empty. - Each of the element in the list is not an empty string (i.e. “ “)
Result	Actual output matches the expected output and test passed.

Test	Description
Module Tested	File: Text_Preprocessing.py Class: TextPrep Method: default_clean
Test Goal	- Guarantee the output format of the method return. - Check whether each cleaning stage is properly done
Test Procedure	- Ensure that the output list is non-empty. - Ensure that elements of the list are non-empty strings. - Check whether common stop words are part of the output. (e.g. common verbs, pronouns, prepositions, determiners) - Check whether there are punctuations in the output list. - Check whether there are uppercase letters in each element of the output string.
Input	Full body text of a news article from a sample XML file taken from the NYT Corpus. Location: tests/data/1815742.xml
Expected Output	- The returned list is not empty. - Each of the element in the list is not an empty string (i.e. “ “) - No stop words and punctuations in the output list. - Each character for each token is in lowercase.
Result	Actual output matches the expected output and test passed.

```
(testProduction) Nicholas@NYWM testProduction % python3 tests/test_Text_Preprocessing.py
Test_Text_Preprocessing passed!
```

Image: Actual test output

```python
class test_Text_Preprocessing:
    def __init__(self, testPath):
        # Initialising the TextPreprocessing object and the required models
        self.nlpModel = spacy.load(("en_core_web_md-2.3.1"), disable = ["tagger", "ner"])
        self.obj = XParser(testPath)
        self.fullText = self.obj.getFullText()
        self.prep = TextPrep(self.fullText, self.nlpModel)
        with open(stopWords, "r") as stopWordFile:
            self.stopWords = [line.rstrip('\n') for line in stopWordFile]

    def test_sentencizer(self):
        self.sentences = self.prep.sentencizer()
        sampleSentences = ['FOR years, Internet users have treated online photo services like a batch of holiday snapshots -- checking them out for a w
        # Black box testing
        assert self.sentences == sampleSentences
        # Testing if the list is not empty and the elements are non-empty strings
        assert self.sentences != []
        for sentence in self.sentences:
            assert type(sentence) == str
            assert sentence != ''

    def test_default_clean(self):
        self.cleaned = self.prep.default_clean()
        sampleCleaned = ['years', 'internet', 'users', 'treated', 'online', 'photo', 'services', 'like', 'batch', 'holiday', 'snapshots', 'checking', '
        # Black box testing
        assert self.cleaned == sampleCleaned
        # Ensure that the output list is non-empty.
        assert self.cleaned != []

        for unit in self.cleaned:
            # Testing if the output is a string
            assert type(unit) == str
            # Check whether common stop words are part of the output. (e.g. common verbs, prepositions, determiners)
            assert unit not in self.stopWords
            # Check whether there are punctuations or symbols in the output list.
            punctuation = [i for i in string.punctuation if i !='$']
            assert unit not in punctuation
            # Check whether there are uppercase letters in each element of the output string.
            if unit != '$' and any(char.isdigit() for char in unit) != True:
                assert unit.islower() == True


if __name__ == '__main__':
    test1 = test_Text_Preprocessing(testPath)
    test1.test_sentencizer()
    test1.test_default_clean()
    print("Test_Text_Preprocessing passed!")
```

Image: Actual code of the test

2.2 Keyphrase Extraction

Test	Description
Module Tested	Pke_keyphrase
Test Goal	- To ensure that PKE_KEA_Module correctly executes the KEA algorithm on the NYT corpus - To guarantee the consistency of output format
Test Procedure	To perform the KEA algorithm on a file from the NYT corpus and to compare the output with a previously verified output. That is that the output from the file 1815742.xml matches the output below: [('smilebox', 0.04721038011946404), ('online', 0.019741538672329157), ('users', 0.016916023220632924), ('photo', 0.01632993581652423), ('company', 0.015016062009211)] The test is ran by running the following command in terminal from the root directory of the package:

	python3 tests/test_Pke_keyphrase.py
Input	Sample file taken from the NYT corpus Location: tests/data/1815742.xml
Expected Output	Test Passed!
Result	*See image below*

```
(testProduction) Nicholas@NYWM testProduction % python3 tests/test_Pke_keyphrase.py
WARNING:root:LoadFile._df_counts is hard coded to /Users/Nicholas/.local/share/virtualenvs/testProductio
n-Bo-Xqf7m/lib/python3.7/site-packages/pke/models/df-semeval2010.tsv.gz
/Users/Nicholas/.local/share/virtualenvs/testProduction-Bo-Xqf7m/lib/python3.7/site-packages/sklearn/bas
e.py:334: UserWarning: Trying to unpickle estimator MultinomialNB from version 0.20.0 when using version
 0.23.2. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
Test Pke_keyphrase passed!
```

Image: Actual test output

```python
# Get path to parent directory and append to PYTHONPATH
currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parentdir = os.path.dirname(currentdir)
sys.path.insert(0,parentdir)


# Importing modules to test
from Pke_keyphrase import PKE_KEA_Model
from Preprocessing.XParser_Module import XParser

# Getting path to test files
testPath = os.path.join('tests', 'data', '1815742.xml')

class test_PKE_KEA_Model:
    def __init__(self, testPath):
        # Initializing the necessary objects
        self.obj = XParser(testPath)
        self.fullText = self.obj.getFullText()
        self.pkeModel = PKE_KEA_Model()

    def test_get_keyphrases(self):
        # Verifying output
        self.actualKeyphrases = [('smilebox', 0.04721038011946404), ('online', 0.019741
        keyphrases = self.pkeModel.get_keyphrases(self.fullText)
        assert keyphrases == self.actualKeyphrases

if __name__ == '__main__':
    # Running the tests
    test1 = test_PKE_KEA_Model(testPath)
    test1.test_get_keyphrases()
    print("Test Passed")
```

Image: Actual code of the test

2.3 Named Entity Recognition (NER)

Test	Description
Module Tested	File: Spacy_NER.py Class: Spacy_NER_Model Method: outputSentenceEntities
Test Goal	Guarantee the consistency of output format
Test Procedure	- Check whether the output is always an array of tuples of (string, string) - Ensure that the entity labels are limited to WHO, WHEN, WHERE only. - For the same input data, the model consistently recognizes the exact texts as entities and the labels are the same.
Input	A list of sentences processed from the full body text of a sample file taken from the NYT corpus Location: tests/data/1815742.xml
Expected Output	The output array is a list of tuples of (text, label), where both data are string type, and all labels are either WHO, WHEN, WHERE, and no other label should exist. The recognized entities and labels are always consistent for every run.
Result	Actual output matches the expected output and test passed.

```
(testProduction) Nicholas@NYWM testProduction % python3 tests/test_Spacy_NER.py
Test_Spacy_NER_Model Passed!
```

Image: Actual test output

```python
# Importing modules to test
from Spacy_NER import Spacy_NER_Model
from Preprocessing.XParser_Module import XParser
from Preprocessing.Text_Preprocessing import TextPrep
import spacy

# Getting path to test files
testPath = os.path.join('tests', 'data', 'sample1.xml')

class test_Spacy_NER_Model:
    def __init__(self, testPath):
        #Prepping the model for testing
        self.nlpModel = spacy.load(("en_core_web_md-2.3.1"), disable = ["tagger", "ner"])
        self.nerModel = Spacy_NER_Model()
        self.obj = XParser(testPath)
        self.fullText = self.obj.getFullText()
        self.prep = TextPrep(self.fullText, self.nlpModel)
        self.sentences = self.prep.sentencizer()

    def test_get_NER(self):
        # Predetermined output
        self.actualNamedEntities = [('19th-century', 'WHEN'), ('British', 'WHO'), ('Richard Bur
        # Blackbox testing of output
        assert self.actualNamedEntities == self.nerModel.ouputSentenceEntities(self.sentences)
        # Testing of internal logic
        for entity in self.actualNamedEntities:
            # Outputs are either when, who or where, and are tuples containing strings
            assert entity[1] in ['WHEN', 'WHO', 'WHERE']
            assert type(entity[0]) == str
            assert len(entity) == 2

if __name__ == '__main__':
    test1 = test_Spacy_NER_Model(testPath)
    test1.test_get_NER()
    print("Test_Spacy_NER_Model Passed!")
```

Image: Actual code of the test

2.4 Summarization

Test	Description
Module Tested	File: Nltk_summarization_nyt.py Class: Text_Summarizer Method: get_Summary
Test Goal	Ensure that the summary generated are made out of strings, and is the expected output from the algorithm.
Test Procedure	- Check the output is made up of sentences from the original text.
Input	An xml file
Expected Output	Strings made up of sentences that are from the original text.

Result	The output summary matches the reference summary

```
(testProduction) Nicholas@NYWM testProduction % python3 tests/test_nltk_summarization.py
Summary test passed!
```

Image: Actual test output

```python
# Importing modules to test
from Nltk_summarization_nyt import *
from Preprocessing.XParser_Module import XParser


# Getting path to test files
testPath = os.path.join('tests', 'data', '1815742.xml')

class test_Nltk_summarization:
    def __init__(self, testPath):
        # Prepping the variable and objects for testing
        self.obj = XParser(testPath)
        self.fullText = self.obj.getFullText()
        self.nltkObj = Text_Summarizer(self.obj, self.fullText)


    def test_get_Summary(self):
        #Blackbox testing for the module
        self.summary = '''The darling of free online photo-sharing, P
        assert self.summary == self.nltkObj.get_Summary()

if __name__ == '__main__':
    test1 = test_Nltk_summarization(testPath)
    test1.test_get_Summary()
    print("Summary test passed!")

```

Image: Actual code of the test

3. Integration Test

Our program required the integration of 6 different modules. The output of each module that we call must match the input for a different module. Furthemore, we must ensure that there are no distortions to the data when passing information between different modules due to different networks or data protocols. Thus, integration testing is done in 3 ways:

1. Ensuring that every function that we use outputs data in a format and a structure that is predefined.
    a. The majority of this work is performed when testing each individual module (as seen above), where we ensure that the output matches predetermined criteria
2. Ensuring that every function that we use can handle all possible formats and structures of input data.
    a. The majority of this work is performed when testing each individual module (as seen above), where we ensure that the functions can handle inputs that match a predetermined criteria
3. A overall testing of the application ensuring that there are no errors when passing information between different modules
    a. Since our application does not make use of complex network pipelines and different communication protocols, there is unlikely to be major issues with passing data from one module to another. This is because the passing of data is handled through the passing of objects and variables at a language level--things that are unlikely to fail.

       Thus the goal of this test would be to find logical errors and typos in our code itself.

       We decided to use black box testing here because:
        1. Black box testing is the most suited to testing the project requirements and functionalities.
        2. We use this method to verify any contradictions present in the system and specifications.
        3. Able to quickly inform any part of the GUI work as expected and that the system allows compatibility of different file formats.

3.1 Graphical User Interface

We used black box testing here because we did not code our own gui from scratch but instead used tkinter, a library that is extensively tested and used.

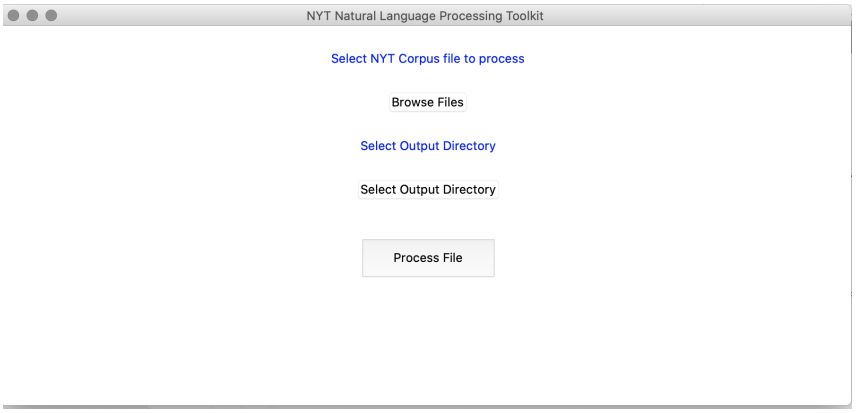
3.1.1 Black Box Testing

Testing Basic Functionality

Testing Main Window

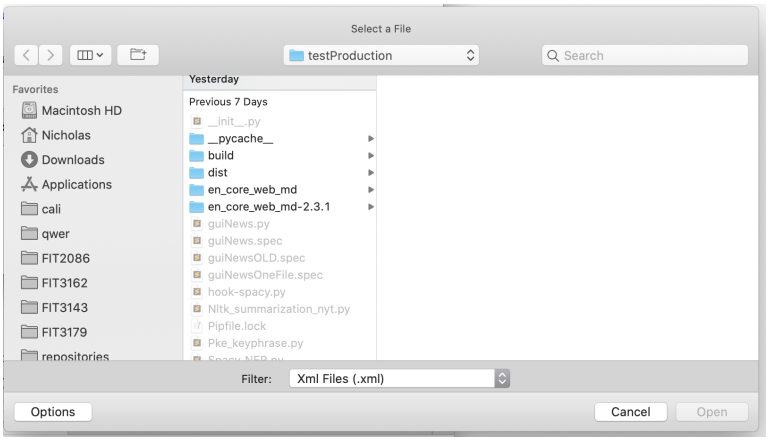
1. Window is produced when the application is launched
   Expected behaviour:
    - Window Launched

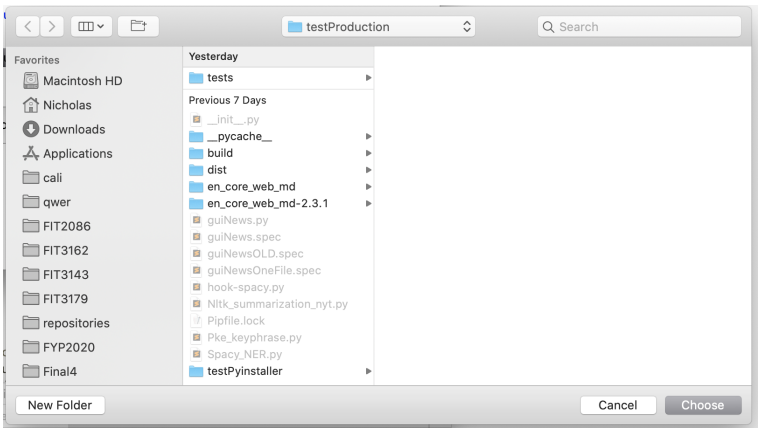
   Actual behaviour:
    - Window Launched



2. File and folder browsers are produced when browse files and select output directory buttons are clicked
   Expected behaviour:
   - File and folder browsers Launched

   Actual behaviour:
   - File and folder browsers Launched

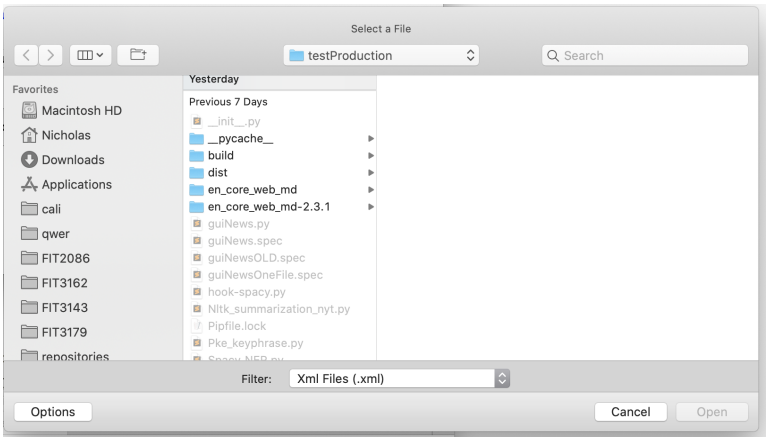




3. File browsers only allows selection of xml files
   Expected behaviour:
   - File browsers only allows selection of xml files

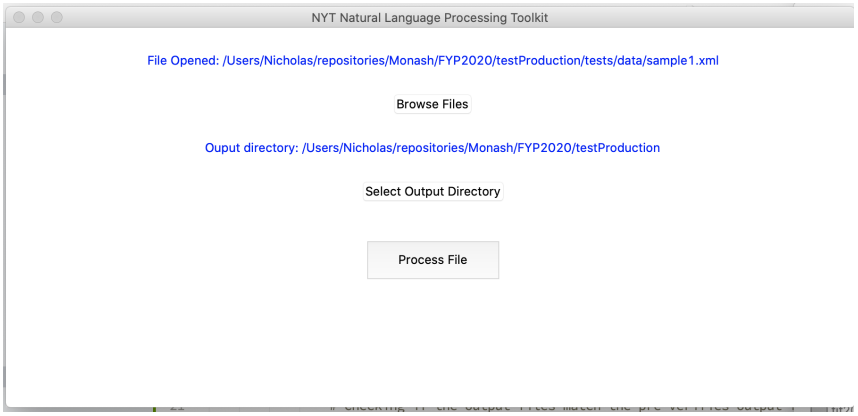
   Actual behaviour:
   - File browsers only allows selection of xml files

-



4. Path to file and folder selected show up in gui
   Expected behaviour:
   - Path to file and folder selected show up in gui

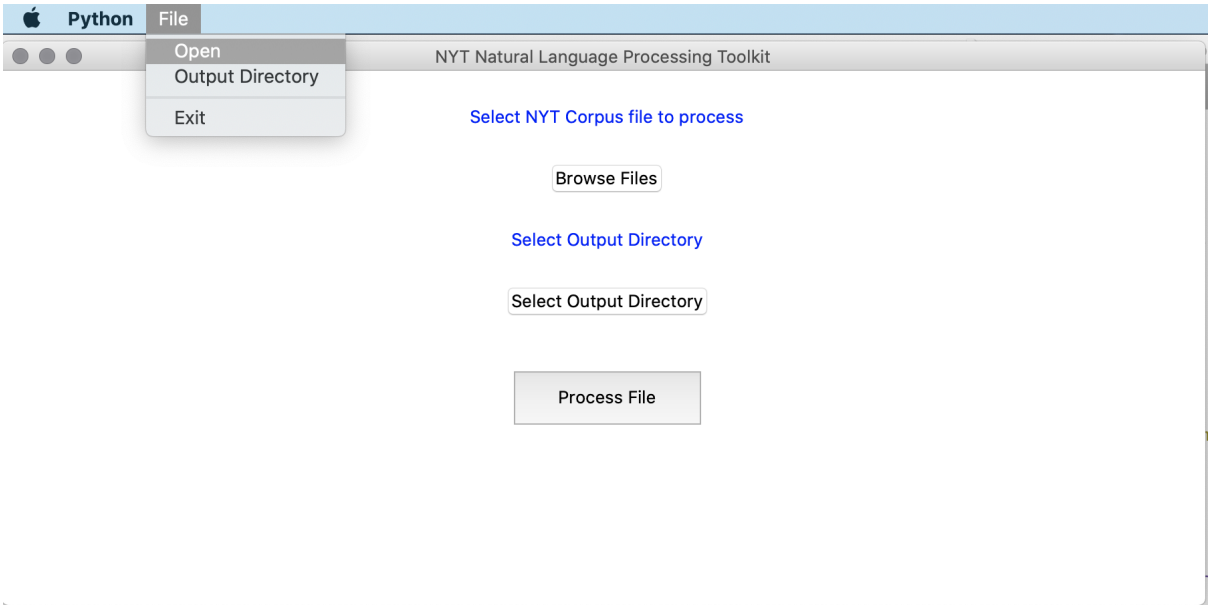
   Actual behaviour:
   - Path to file and folder selected show up in gui

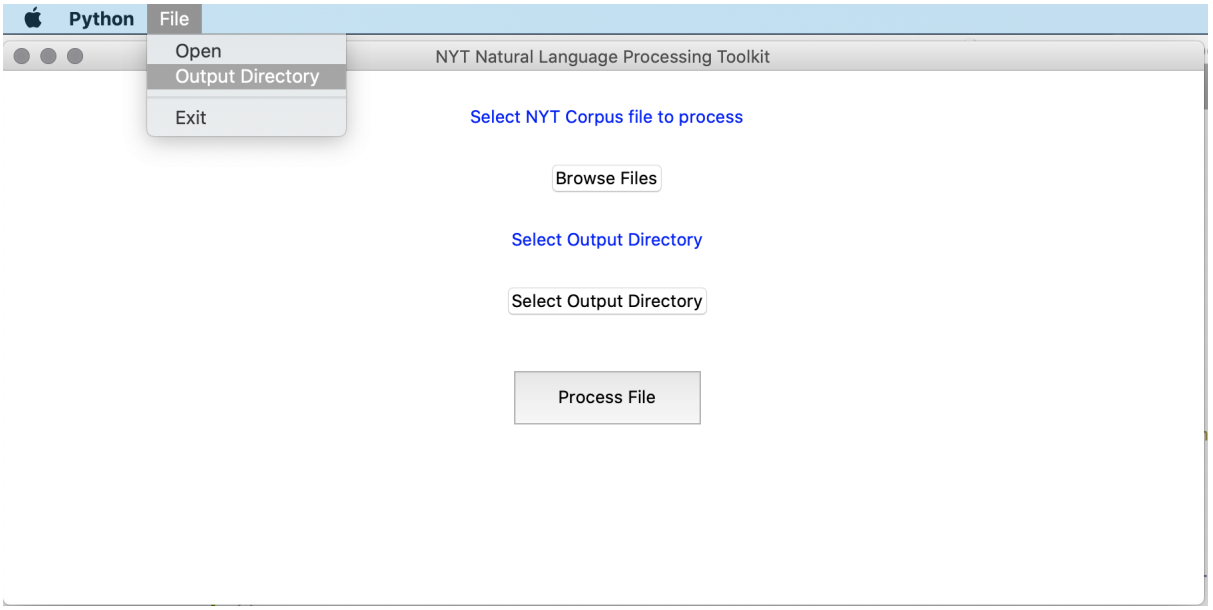


Testing Menu Bar

5. Path to file and folder selected show up in gui when File -> open and File -> output directory are clicked
   Expected behaviour:
   - Path to file and folder selected show up in gui

   Actual behaviour:
   - Path to file and folder selected show up in gui





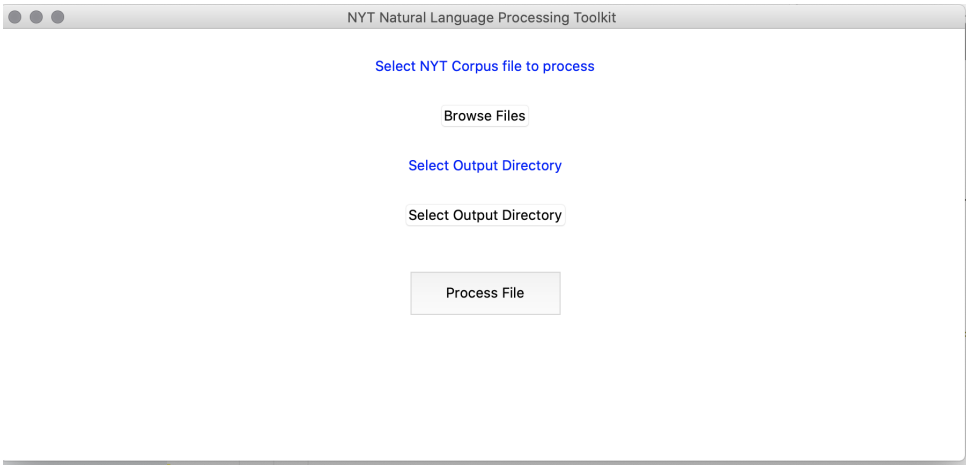
6. Window closes when file-> exit is clicked
   Expected behaviour:
   - window closes

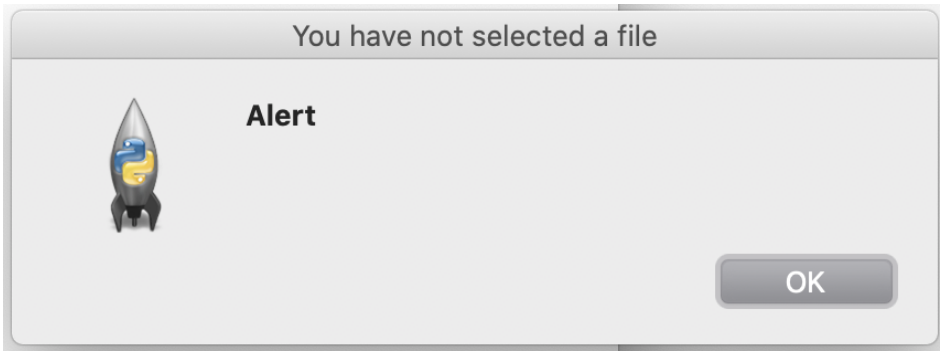
   Actual behaviour:
   - window closes

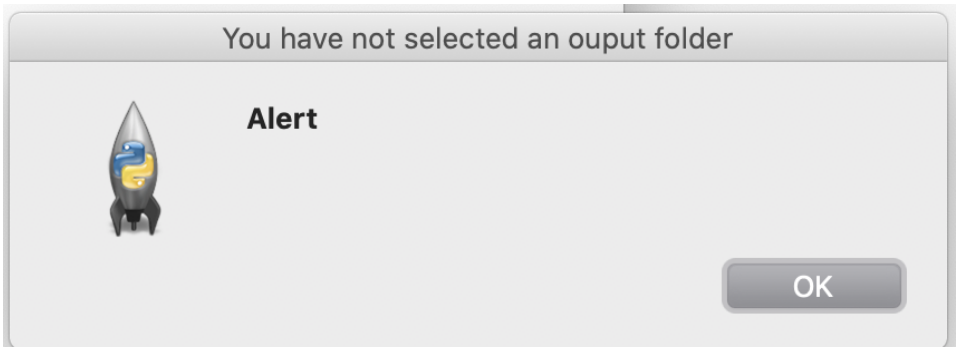
Testing Boundary Conditions

7. Message windows pops up when file and directory are not selected and Process File button is clicked
   Expected behaviour:
   - Warning window pops up

   Actual behaviour:
   - Warning window pops up

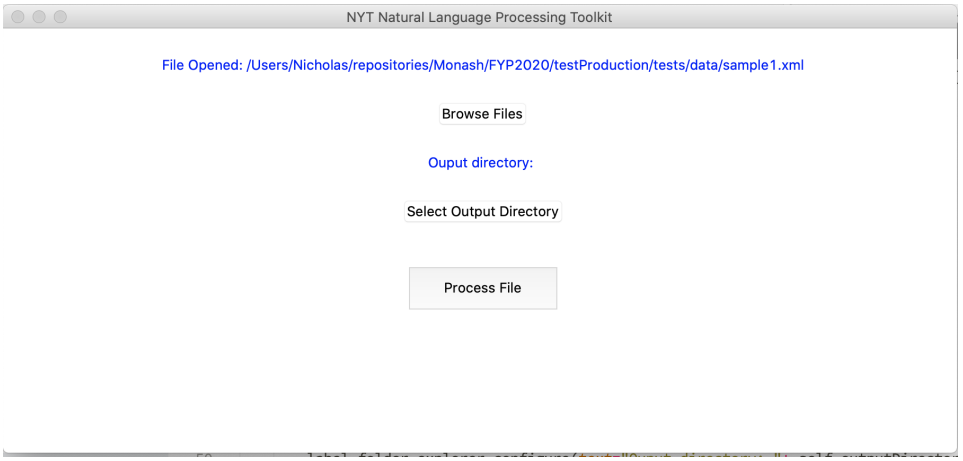


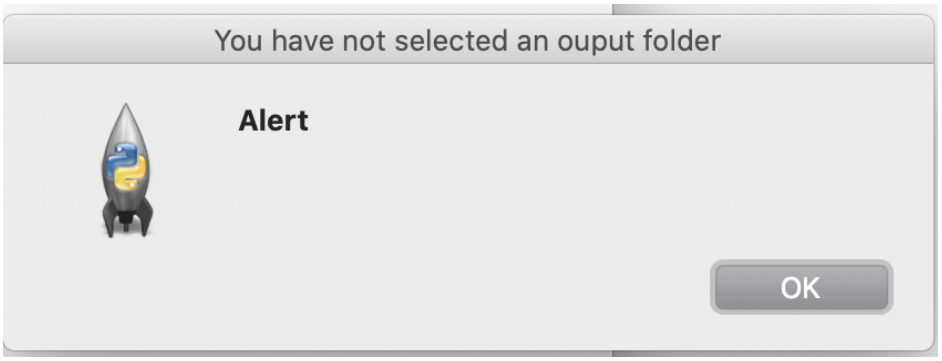




8. Message window pops up when directory is not selected and Process File button is clicked
   Expected behaviour:
   - Warning window pops up

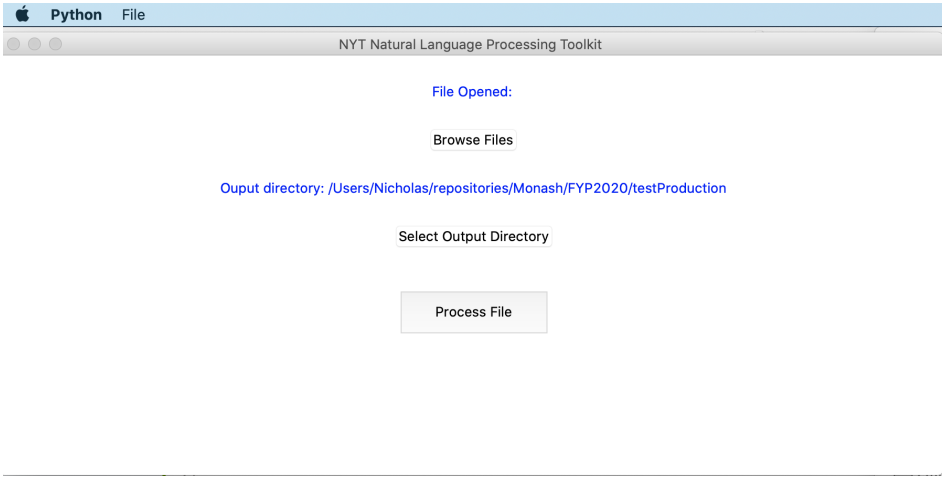
   Actual behaviour:
   - Warning window pops up

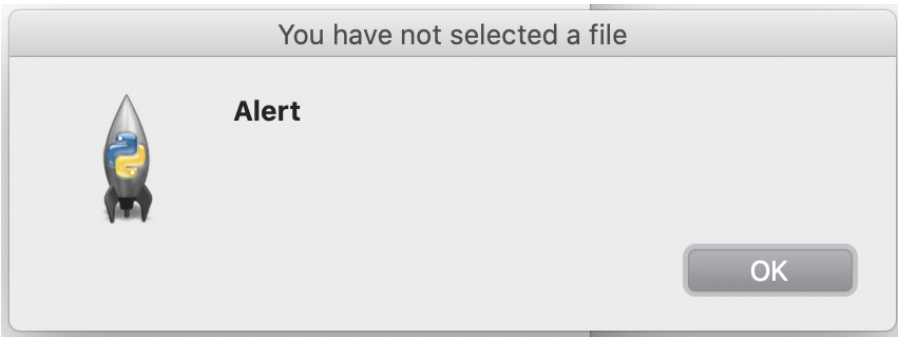




9. Message window pops up when file is not selected and Process File button is clicked
   Expected behaviour:
   - Warning window pops up

   Actual behaviour:
   - Warning window pops up





3.1.2 Output Testing

Test	Description
Module Tested	guiNews.py
Test Goal	- To ensure that the output of the overall program when provided with a predetermined input file matches pre-verified output files
Test Procedure	1. Run guiNews.py and select tests->data->sample1.xml as input file 2. Select root directory as output directory 3. Click process 4. From the test directory, run python3 test_guiNews.py 5. The test script with compare the outputs and remove the output files from the root folder
Input	Sample file taken from the NYT corpus Location: tests/data/1815742.xml
Expected Output	Test Passed!
Result	*See image below*

```
Nicholas@NYWM testProduction % python3 tests/test_guiNews.py
Gui Output Test Passed!
```

Image: Actual test output

```python
import os
import sys
import filecmp
import inspect

currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))
parentdir = os.path.dirname(currentdir)
sys.path.insert(0, parentdir)


class test_guiNews:
    def __init__(self):
        # Preparing the list of paths of the output files
        self.outputFileNames = ['output_keyphrase.txt', 'output_NER.txt', 'output_summary.txt']
        self.sampleFilePaths = [os.path.join('tests', 'data', "sample_" + i) for i in self.outputFileNames]

    def test_Output(self):
        for i in range(3):
            # Checking if file exist
            if os.path.exists(filename):
                # Checking if the output files match the pre-verifies output files
                assert filecmp.cmp(self.outputFileNames[i], self.sampleFilePaths[i]) == True
            else:
                print("File " + filename + " does not exist")


if __name__ == "__main__":
    test1 = test_guiNews()
    test1.test_Output()
    # Removes test files from root directory
    for filename in test1.outputFileNames:
        if os.path.exists(filename):
            os.remove(filename)
        else:
            print("File " + filename + " does not exist")
    print("Gui Output Test Passed!")
```

Image: Actual code of the test

4. User Test

As previously stated, we described most of the subjective opinions of our Alpha user in the potential improvements and future work section, thus, the following section is mostly focused on giving an overview of the impressions from our Alph users.

We conducted the usability test by having our friends and family use of the app., and recording their feedback in qualitative and quantitative forms. We first described the app to them

- Explain the program to them
    - Qualitative
    - Quantitative feedback

Qualitative Data	**Quantitative Data**
Success Rate	Observation using the GUI

Task time	Problems experienced
Satisfaction questionnaire	Comments/ Recommendation

- One evaluation for nyt dataset

Sample of summary from user based on 1851213.xml

In 1959, when she was 22, and known as Linda Riss, Mrs. Pugach opened the door of her Bronx apartment to a thug who claimed to have an engagement present for her and instead threw liquid lye in her face.
The overall vibe is morbidly entertaining, though something of a downer, partly because it's unclear if Mr. and Mrs. Pugach know that they are such sick puppies, partly because it's unclear if Mr. Klores cares that they are.

- Qualitative opinion on how good it is, and how to improve

Comments from users
‘’The output summary from the system are sentences from the articles. It’s different from how the ‘usual’ summary we are used to which is concise and abstract.”

“The time taken for the system to output results is quick.”

Test Limitation and Future Work

Method: Black-box Testing
Code access is not required.
Tester is not aware of the system underlying architecture and does not have access to the source code.
Limited coverage
Only a selected number of test scenarios is actually performed
User perspective vs Developer’s perspective.
Tester are unable to target specific area of the code that are prone to bugs or errors

Method: White-box Testing
Time constraint.
In some cases, testers are unable to discover all the bugs or defects found in the codes. The codes may have too many hidden errors. Also, testers might not find unimplemented or missing functionality.
Code access is required.
Testers require a high level of technical knowledge of the internals of the software under test.
Test suites created using this method are highly specific.

Continuous integration (CI) and Continuous delivery (CD)

Continuous integration allows code from various developers into one central branch of the repo multiple times a day. Each push is verified by an automated build allowing teams to detect problems early. Regular integration allows for earlier error detection and is able to locate them easily.

Continuous delivery enables team to implement small changes and check in code to version control repositories frequently. The objective of CI is to keep code in a deployable state at any given time. This means that the features that are made available are vetted, tested and debugged in a smaller sprint cycle.