**Daylan Vig**

**Project Proposal – NLP Application**

**Introduction**

Oftentimes, incomplete fragments of text are found. These might be text fragments from books that were damaged, as would be found by an archaeologist, or might just be part of a word puzzle. Regardless of the intent behind them, humans are not always able to deduce the most appropriate way to complete a sentence – we simply do not maintain the required vocabulary such that it is always readily accessible.

One solution to this problem is to use computers to attempt to complete details of text fragments – letters can be determined for word fragments, words can be determined for sentence fragments. The challenge behind this is that natural languages are challenging to process. Sentences have varied structures, duplicate words do not necessarily mean the same thing as they rely on context, and conversely different words might have identical meanings. The intention here is to design a system which will allow users to submit fragments of text so that they may be completed with reasonable accuracy and efficiency.

**Project Summary**

The project will support the completion of fragmented text for remote users through a web portal. Text fragments will be formatted through an integrated text editor and processed by a neural network before being returned in completed form to the user’s browser.

**Project Details**



Figure 1. High level dataflow diagram for the system.



Figure 2. High level structure of the planned modules.

Front end/user interface:

The system will be designed to be accessed by end users through the web browsers with the largest market share – Chrome, Internet Explorer, Firefox, Edge, and Safari. UX testing will be performed natively on Windows 10, and through a virtual machine running Linux Mint. At this time, mobile device support is not explicitly planned to be supported but will be subject to time availability during implementation.

Through the front end of the system, users will have access to a text editor (on the main page), a submission portal to contribute training files, an account management page with will allow for changes and account history viewing. These pages will require users to be registered and logged in, with the exception of the main text editor page, which will be publicly accessible.

Below is a sample, first draft prototype of what the registration page might look like. The top menu bar will be present on every page of the web portal. Not pictured is a bottom footer which will provide general information for users such as contact information, information on how the system functions, etc.

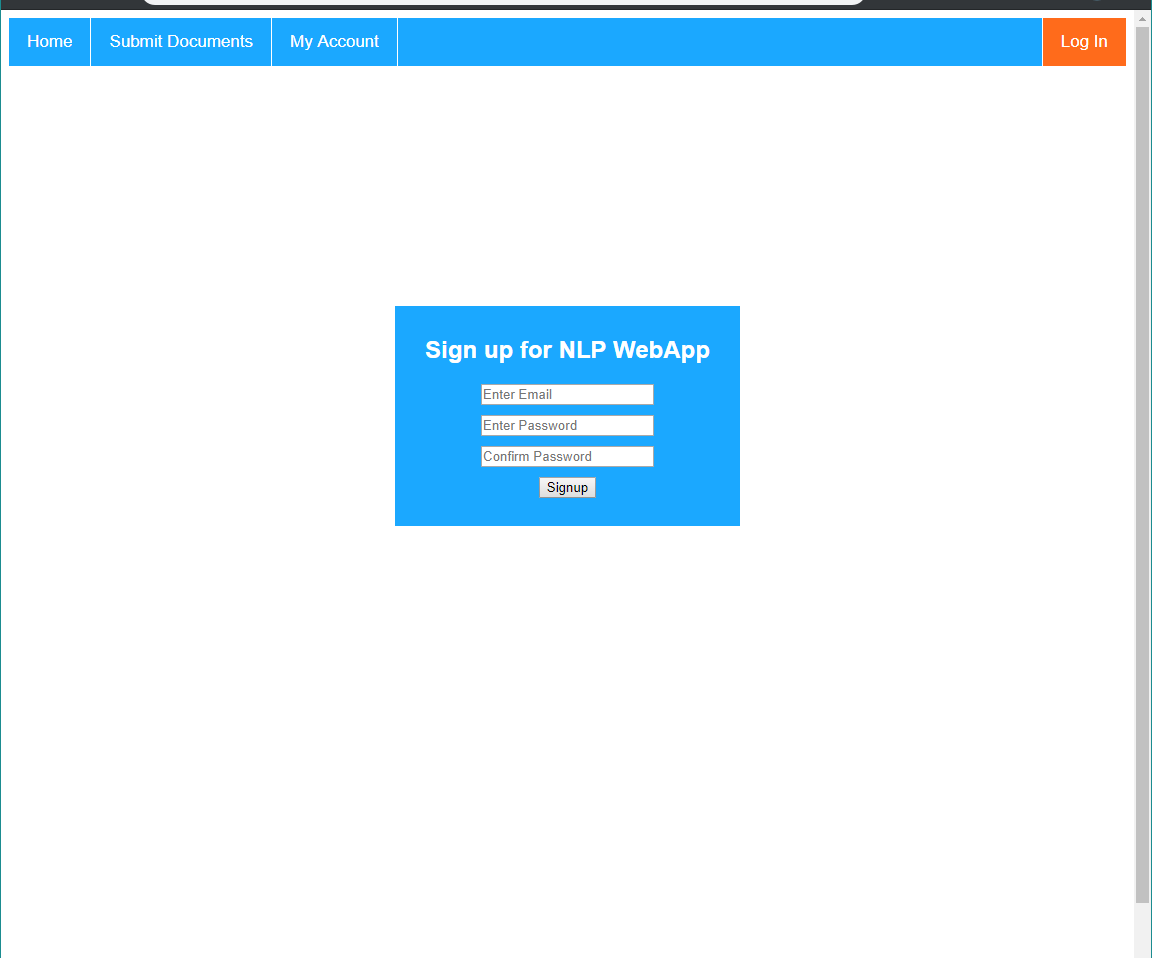


Figure 3. Sample registration page for the web portal

The colour scheme will be consistent throughout. Highlighted items on the top menu bar will darken, the currently selected page’s button will be blue, as will the log in/out button.

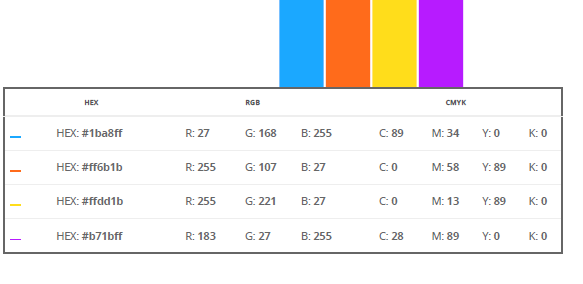


Figure 4. Colour scheme to be used throughout the web portal.

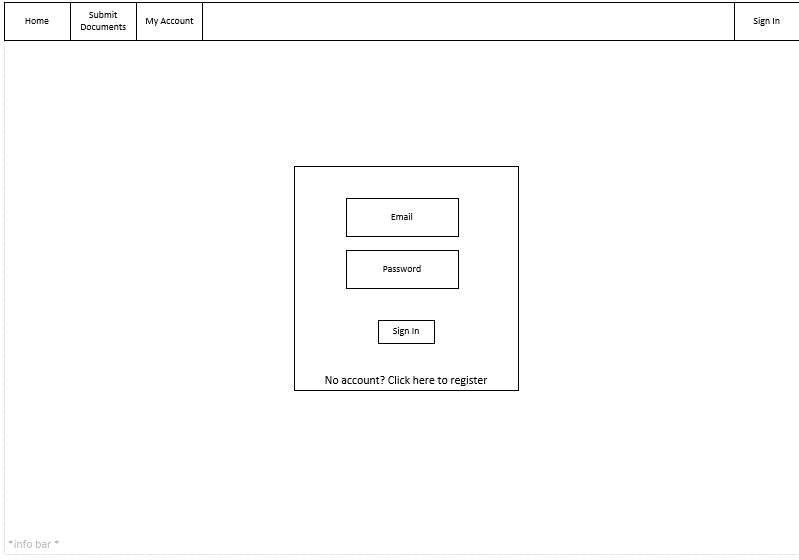


Figure 3. Sign in page and text fragment submission page



Figure 4. Training file upload page

The web portal will operate using Python’s Flask microframework. This will allow for the easiest integration into the back end of the system, which will be Python based for reasons discussed below. Some extensions supported by Flask will allow for the development of a secure, stable system which will support the intended usage (ie. low to medium volume processing).



Figure 5. Module for webpage routing

Back end system:

As mentioned, the back end of the system will be developed using Python due to the significant value that can be extracted from certain packages. Individual packages that will be used are discussed in the materials section.

The back end of the system will consist of a preprocessor and an interpreter for language processing. Training files which are uploaded by users will be stored on the server in the user’s directory where they will be cleaned using Python’s NLTK. The cleaned files will be used by the preprocessor to develop a language model.



Figure 6. Level one data flow diagram for the file cleaning process. Complete cleaning procedure to be further assessed as implementation proceeds.

The preprocessor will build a language model by creating weights from the training files using NumPy and Keras. It will also store the words it has encountered in a dictionary database, as well as the frequency with which they’ve been encountered. These weights will be stored for each user’s private access and will also integrated into a shared language model accessible by all users. Users will not be able to access other user’s models individually, only their personal ones or the entire communal model. This will be set to run periodically.



Figure 7. Preprocessing system which will be set to periodically query for new files to generate weights, which will be stored until needed.



Figure 8. Preprocessor helper methods to scan for new files, generate the weights, and merge by user.

When users submit text fragments through the web portal (through AJAX) they will be received by the interpreter. The fragments will contain tags to denote missing words and missing letters so that they can be analyzed by this aspect of the system. Once analyzed, the most probable completion will be substituted for the tagged areas and returned to the client in a user-friendly format.



Figure 9. Level one data flow diagram for the text analysis procedure to make a prediction on the most probably solution, which will be formatted and returned to the User. Results to be stored in the query database so they can be accessed in user history.



Figure 10. Language processor module with helper methods.

User account sessions will be managed through Flask-Login and SQLAlchemy. The information collected for user accounts will include the user’s email (for username and password reset purposes), and password, which will be salted and hashed. The account creation date and last login date will be stored so that the database might be periodically cleansed of inactive users.



Figure 11. Level one dataflow diagram showing the account creation process. Implementation to be integrated using Flask’s login extension with required security protocols.

  
Figure 12. Level one dataflow diagram showing the procedure of signing a user into the system.



Figure 13. Level one dataflow diagram showing the flow of information as a user updates their account information (email, password).



Figure 14. Initial class design for User. Functions as required by Flask login.

Table 1. Database schema

|  |  |
| --- | --- |
| **Data Store** | **Attributes** |
| User | **userID,** email, password, creationDate, lastActiveDate |
| TrainingFile | **fileID,** userID, fileName  \*File: store as /userID/training/fileID |
| Dictionary | **wordID,** frequency |
| TextSubmission | **textID,** userID,submissionDate**,** databaseSelected, contents, completedContents |
| ModelWeight | **userID, creationDate**  \*File: store as /userID/models/\*date\*.h5 |
| TestResults | **testID,** testQueryID, testDate, predicted, accuracy |
| TestQuery | **testQueryID**, textFragment, textComplete |

Indices: User(email), dictionary(frequency)

Testing and Reporting System:

A set of test cases to be developed as listed in milestones. These test cases will be run after the model is initially trained and will be set to automatically run after iterations of model training to ensure that standards are maintained. A report will be generated after each iteration listing the files that were added and the results of the unit tests.



Figure 15. Level one dataflow diagram showing the automated testing reports that will be periodically generated by the system as the model changes.

**Materials/Tools**

MySQL – the systems relational databases (such as the User account database) will be managed using MySQL.

VMWare – will be used to test the interface on a Linux distribution to ensure cross operating system compatibility.

Linux Mint Distro – will be used for ensuring cross operating system compatibility.

Python Packages:

Flask – web framework to allow for smooth integration into the backend systems as they are also Python based.

Flask Login – to simplify multiple user session management and authentication procedures.

SQLAlchemy – will provide support for Flask to be able to access the MySQL database.

NLTK – this toolkit will initially be used to clean files so that they may be integrated into the neural network more efficiently.

Keras – will be used as the base for building the recurrent neural network.

NumPy – certain statistics tools will be used to help develop the neural network for the language model.

Visual Studio Code – will be the environment used to develop the system.

Microsoft Azure\* - potentially to be used to train the machine learning model subject to feasibility testing.

Literature to be referenced as needed:

Neural Network Methods for Natural Language Processing – Goldberg, Yoav

Generating Text with Recurrent Neural Networks – Sutskever, Martens, Hinton

Natural Language Processing of Semitic Languages – Zitouni, Imed

Deep Learning for NLP: Lecture Notes Part 1 – Socher

**Milestones:**

Attached in “TimelineOne.xlsx” with specific details. Each milestone will be completed with ongoing integration testing as the component is completed. As the intention is to compress the project into a shorter time period, reports will be generated approximately twice a week rather than every two weeks.

Summarizing attached file:

Preliminary unit tests will be completed on 2018-11-23

User interface (web portal) to be completed on 2018-11-25

Account management system to be completed on 2018-11-30

File management system to be completed on 2018-12-05

Language processing/modeling system to be completed on 2018-12-12

Text submission/editing system to be completed on 2018-12-15

Testing system to be completed on 2018-12-16

All usage documentation to be completed on 2018-12-17

Reports to be completed on:

2018-11-27

2018-12-01

2018-12-06

2018-12-10

2018-12-14

**Deliverables:**

Web portal – a web portal will be created which allows for users to remotely access the system. They will be able to upload files, submit queries for fragmented text, manage their account, and view information about how the system functions.

Account management system – the account management system will allow users to sign in and sign out, create an account, reset their password, delete their account, and support private database building. Included will be security requirements for password storage and session key generation.

File management system – a file management system will be in place to allow users to upload training files to the system where they will be stored until they are processed by the preprocessor.

Preprocessor system – files will be cleaned to ensure the model is able to be generated in an optimized way (lower case, removal of special characters, stemming, etc). These cleaned files will periodically be used to generate weights for the language model.

Main language processor – the main language processor will allow users to select weights (private, shared) that were created by the preprocessor and provide a text seed (through the web interface) so that missing information can be determined. The intention is to use a long short term memory model to complete this, but the accuracy will be assessed and changes made depending on the required computing power of potential load of the system.

Testing and report system – a system will be in place to automatically run a set of predetermined test queries periodically so that a report can be generated to compare how the accuracy of system changes over time. The report will be saved in a text format (tsv or csv) so there is room for future integration into other visualization tools if desired.