Text Fingerprinting

30th September 2020

OVERVIEW OF PRODUCT

Our final product will be a web application that interfaces with a back end server, which handles all the machine learning. The program will be given training data, and given a piece of text it will be able to correctly classify which author from a given set wrote a specific passage with a high success rate.

The program will work by taking training data from a preset list of authors and learn identifying traits. Then, given a passage written by one of those authors, it should be able to correctly identify which of the authors from the training data wrote the specific passage.

GOALS

1. At least an 85% accuracy rate
2. Have a complete UI that can interface with the backend.
3. Product should function properly as intended.

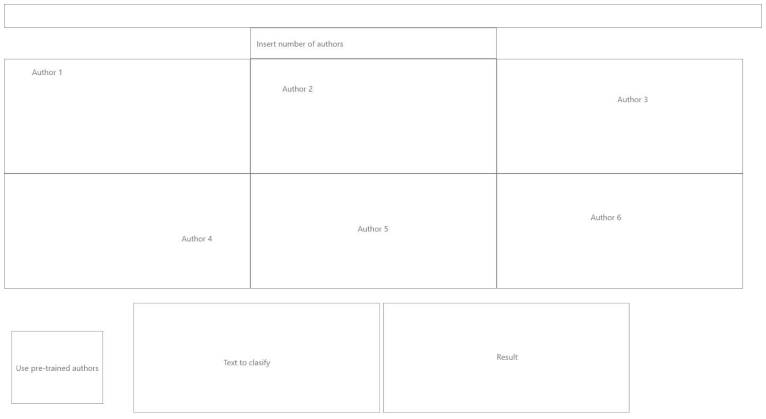
FEATURE LIST

- Web interface to upload text and see results
- Ability to choose which authors to use in the prediction
    - Ability to enter their own training data instead of preset training data
- Display percentage accuracy
- Display model loss graph
- Allow user to input lots of their own writing samples (if time)

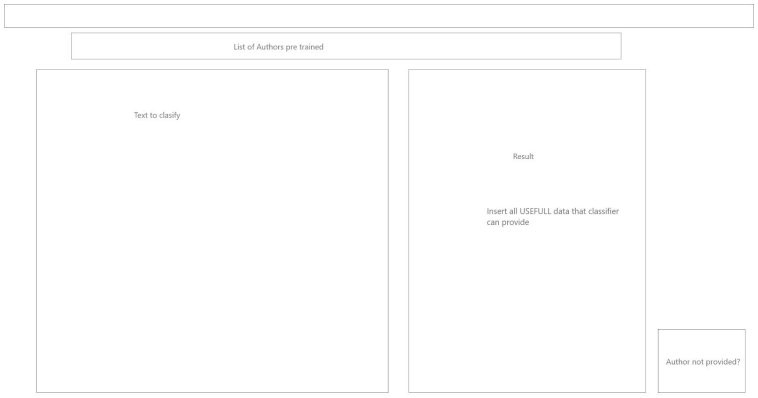
SPECIFICATIONS

UI

- Looks
    - Insert writing samples for different writers
    - Insert testing sample
    - Button to submit
    - Possible writers list and possible percentage
- Methods
    - ButtonClicked
        - Go to next screen
    - ReceiveTrainingText
        - Take txt file/string and put into file for backend to handle
        - Organize different authors
    - ReceiveSampleText
        - Take txt file/string and provide to backend
    - ProvideResults
        - Parse result from backend and provide user with the final answer/percent chances



User has author not trained



User has text with trained author

Back-end

- Directory structure:
    - Input (the input files for training and validating our ML model)
        - training.txt (author/writing training data)
        - validation.txt (author/writing validation data)
    - Feature.py (class for adding features to a feature vector)
        - Methods (sample features that we could feed into our model):
            - average_word_length(): returns the average length of a string
            - average_syllables_per_word(): returns the average syllables per word in a string
            - average_sentence_length(): returns the average length of a sentence
            - freq_of_function_words(): counts the frequency of function words (the, and, of, he, etc)
        - Fields:
            - text (string): a string of the training instance, which is around a paragraph of text written by a single author
    - DataUtils.py (utils for preparing training data)
        - Methods
            - get_html(url): returns html as string of a given url
            - bs_preprocess(text): removes unnecessary characters from the input text
    - classify.py (script to run classifier models and train the data)
        - spilt_data(X, y, test_size=0.33, random_state=42): splits the data into training and validation
        - grid_search(classifier, parameters, cv=5, n_jobs=-1): performs parallel grid search to find the optimal classifier algorithm with highest validation accuracy
        - accuracy(predicted, actual): returns the percentage of correct answers given predicted and actual vectors
    - Server.py (web server to interact with the front-end)
        - /predict
            - To output predictions to the front-end
            - GET parameters:
                - text (string): the text to classify from our pre-trained model
            - Output:
                - Returns [[author1, percentage], [author2, percentage]...]

        - /train
            - For the front-end text sample inputs to be fed into the classifier
            - POST parameters:
                - training (array): [[author, text]]
                - test (string)
            - Returns [[author1, percentage], [author2, percentage]...]
- Notes about directory structure:
    - Each line in training.txt and validation.txt is formatted like “author text”, which represents a single data instance (for example, one line might be “Jay The alarming measures of the British Parliament...”)

TESTING

Use Cases

- Measure common metrics of written works
    - Word Count
    - Number of Paragraphs
    - Number of Sentences
    - Average Sentence Length
    - Average Word Length
    - Etc
- Classify Works
    - Federalist papers were written by Hamilton, Jay, and Madison. Using this technique, we could figure out who wrote what.
- Keep your “writerprint” anonymous
    - Every person has a unique style of writing, unconsciously developed over time. This could compromise their anonymity if they were to reveal to someone their writing.
    - By understanding how well their writing can be classified by a computer, users (especially those with particular concerns about privacy, like whistleblowers) can alter their wording to lower their “writerprint”.

Edge Cases

General Test Cases

- Write tests for the Feature class, making sure that each of the methods works properly

    - For example, we could have a test for the average syllables per word function: avg_syllables_per_word("the quick brown fox") = 1.0
    - We plan to use python's doctests module for writing unit tests
- In the beginning, we will come up with a list of test cases for every method, and tweak our code until the test case passes (test-driven development)

PROPOSED TIMELINE

October 1st -7th

Working on code and development of UI and main Application

Working on writing test cases

October 7th -14th

Working on code and development of UI and main Application

Working on writing test cases

October 14

Alpha version of code for UI and main application should be done

Test cases should be done

October 14 - 21st

Testing and bug fixing as necessary

October 21st

Target date for finishing the main application code as well as UI - (Including testing)

October 21st-26th

Minor bug fixing, and polishing

October 26th

Entire project complete.