The Idea

Because Ali, and potentially other people, need to do word-based analysis, we need to try to separate out words without actually knowing what the letters are or what the words say. I.e. do word separation solely on spatial separation, without any lexical information.

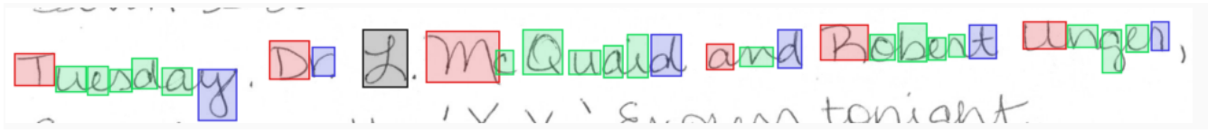
The Workflow

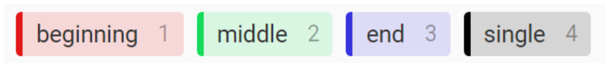
I am currently doing this using a Random Forest model with 4 values

So, our current workflow to do this can be broken into two parts: building the model (only need to do once) and consulting the model to do word separation (do everytime we run a document through **processHandwriting()**)

Building the Model

1. Get documents that we want to use. Currently 15 are in use but there are 20 total from 20 writers in *inst/extdata/word_splitting_data/documents_to_split*. There are even more (from the same 20 writers) in *…/all_documents*
2. Split the lines with a cropping/screenshotting tool and store in *…/all_split_lines*
3. Upload these lines into a program called **Label Studio**
    a. I'll put another section on how to set this up if you need it below.
4. Label each graph (I am using my best approximation of a letter) with one of four labels: [beginning, middle, end, single]





5. Export the labels as a JSON files
6. Use *WordModelMaker.R* inside of *handwriter/R* which takes the following steps:
    a. *Line 18* | Load the file as JSON
    b. *Lines 20 - 31* | Create a DF with the following information per character: line number, line height, line width, character height, character width, left-most x-value of the character, label)
    c. *Line 34* | Sort left to right and top to bottom in case it isn't done already
    d. *Lines 43 - 57* | Create four new rows of the DF that we will use to train our model:
        i. *height_prop* | Height of the character as a proportion of the line height
        ii. *width_prop* | Width of the character as a proportion of the line width
        iii. *to_right_prop* | distance from right side of character to left side of next character (as a proportion of the line width)
        iv. *to_left_prop* | distance from left side of character to right side of last character (as a proportion of the line width)
    e. Lines 59 - 73 | Setting up for creating Random Forest Model
    f. Lines 75 - 81 | Model testing loop to try different variable numbers
    g. Line 87-89 | Train the actual Random Forest and save it as *wordModelNew*
        i. This gets saved in *handwriter/data/WordModelNew*

Using the Model

1. Inside of **processHandwriting()** on line 615 of *JunctionDetection.R* exists **add_character_features()** (line 815 of *JunctionDetection.R*) which has the function **add_word_info()** (line 332 of *ExtractFeatures.R*)
2. The first part of this function sets up a DF of the incoming documents with the same info as the model [height_prop, width_prop, to_right_prop, to_left_prop]
3. Then on line 427 we actually use the model to get new predictions, as well as some additional code for some clean up like on 430 - 432. This is probably where I would say any sort of 'spaghetti code' begins
4. The loop from line 446 starts at wordCount = 1 and goes through each character putting the characters together in a (somewhat intelligent) way. When we think there is a word boundary the wordCount will be increased.
5. The result of this is that it adds another character feature, wordCount, and this is then used later when stitching words together in **create_words()** in *CreateWords.R*

The Problem

Well, sometimes if words are closer together than normal or letters are further apart than normal it will separate too soon or too late sometimes.

How I Would Attack This

I would look starting at line 430 and continuing through the loop starting at line 446. I don't believe that the character predictions can get that much better. But what can certainly be improved is the logic in which characters are stitched together to create words.

Certain steps are being taken right now if it is believed that predictions are wrong - but are there more steps or better steps to take instead?

What else is in this Document?

Directory of Stuff | Where all the files are located

Label Studio | A guide to setting up and using LabelStudio with our data

Directory of Stuff

Inside the **handwriter/inst/extdata** folder is external data…

word_splitting_data | documents and data relating to word separation model

- **/all_documents** | documents from 20 writers (450 total documents)
- **/all_split_lines** | split lines (manually, by me) of one document from each writer
- **/documents_to_split** | documents to be split (1 chosen from each 20)
- **/labelled_json** | labeled data as json objects.
  - 15W_140L means it consists of 15 writers and 140 lines
  - Keeping smaller datasets to track model performance

word_splitting_problems | documents that were causing Ali problems

Inside the **handwriter/R** folder is the following source code…

/WordModelMaker.R - Creates Random Forest model with labelled dataset

/ExtractFeatures.R

- line 313 | wordModelNew - data documentation for Random Forest model
- line 321 | add_word_info() - Uses model and logic to determine word separation

Label Studio

Label Studio is a Python-based data labeling program.
Basic information is here: https://labelstud.io/

Getting Label Studio

You can get label studio using **pip install -U label-studio** if you have Python 3.0+
Then use command **label-studio** to start it up

The account is local so just make a new one and remember the user/pass

Importing a Project

Create Project > Name and description doesn't matter
Data Import > Import the most recent JSON (15W_140L.json)
Labeling Setup > Object Detection with Bounding Boxes
    Click on 'code' in top right of the left hand panel and drop this in:

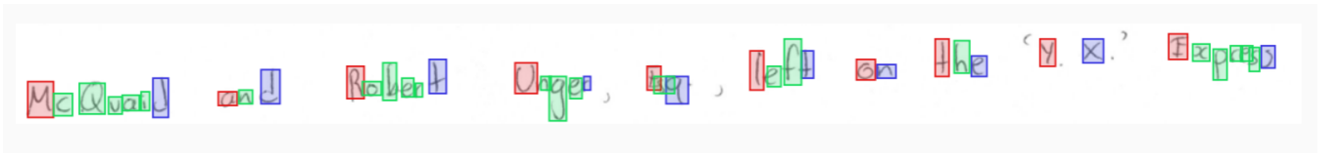
```
<View>
  <Image name="image" value="$image"/>
  <RectangleLabels name="label" toName="image">
    <Label value="beginning" background="#e21818"/>
    <Label value="middle" background="#14db5a"/>
    <Label value="end" background="#3734df"/>
    <Label value="single" background="#050505"/>
  </RectangleLabels>
</View>
```

> Save

When you look at the labels you won't actually be able to see the images. I'm sure there is a way to put the corresponding images up but I think you can just ignore it. The actual annotations are there, and you can add more images.

Labeling

Import a folder of new images to label
Click on each and label and label like so:



Exporting Data

When done, export as JSON and re-run WordModelMaker with new JSON object