A big picture of the workflow

how will different parts of the genre project communicate?
what is a "prediction"?

by Ted / May 24, 2013
primarily relevant to Mike, Boris, and Vijay

We're building an application that will gather training data, as well as a workflow that will produce predictions about genre membership. How will those two parts of the project communicate with each other?

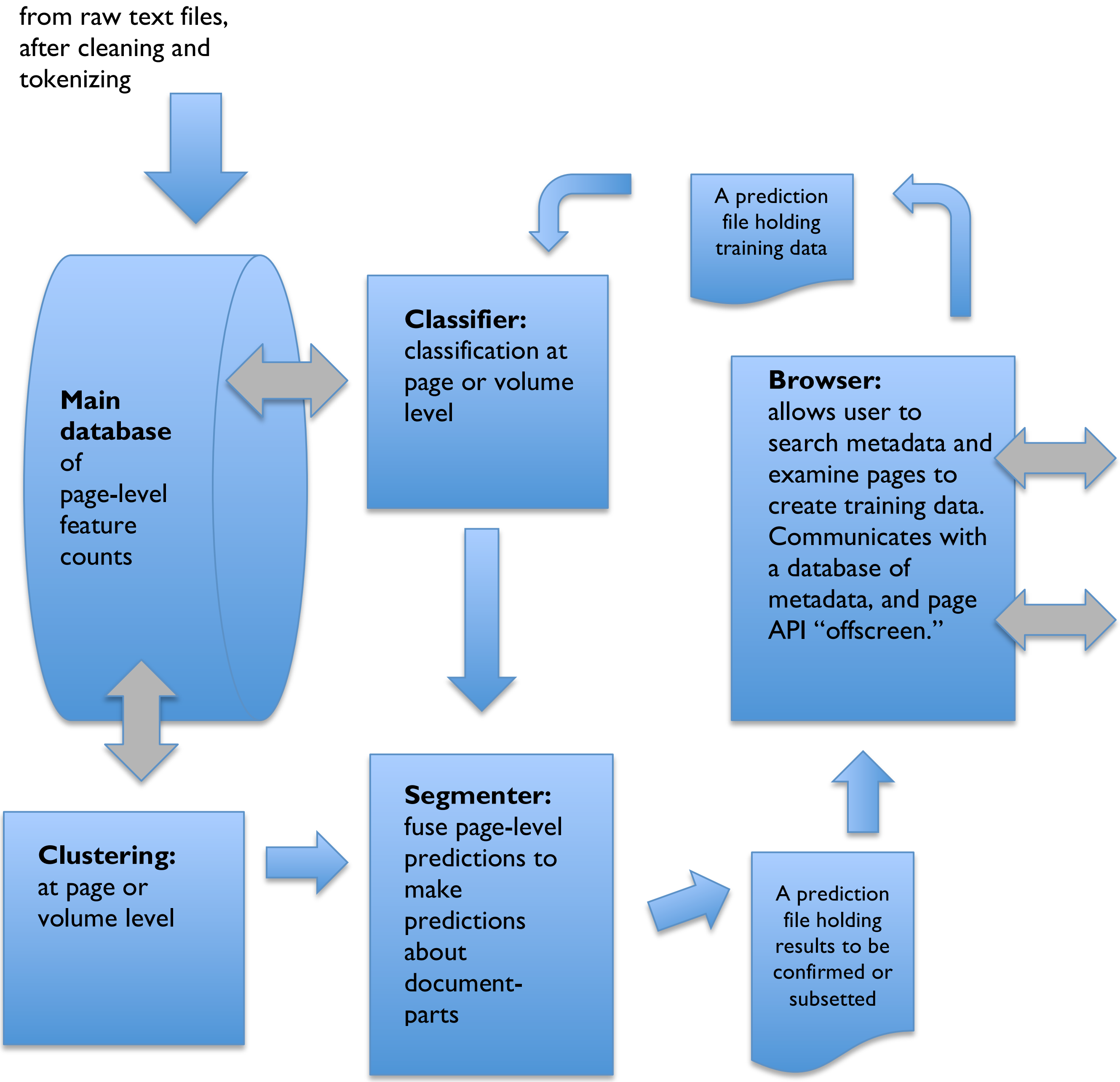
To define this, I'd like to suggest a standard format for a data object I want to call a "prediction." Because there's been talk of using ARFF in HathiTrust Research Center, I think I'm going to base this format on the ARFF standard.

A "prediction" characterizes a set of data objects (for our purpose, volumes or page ranges from volumes) by associating each data object with a predicted degree of confidence about its membership in a specific class. When we create training data for a classifier to use, we're creating a prediction. When a classifier uses this model to identify a longer list of document parts that belong to the same class, it's also creating a "prediction."

A prediction has three key elements: 1) it names a class, 2) it lists data objects (documents or document-parts), and 3) for each data object, it declares a degree of confidence that the object belongs to the class in (1).

The classifying/clustering part of our workflow will communicate with the training-data part by exchanging predictions. E.g., a user working with Mike's browser might manually tag volumes, or page ranges, as "drama." In this case, all the predictions will have 100% (1.0) confidence — because of our high level of trust in human readers.

Then we might use that prediction to train a classifier. It would go to work on the whole dataset in order to identify pages that seem to match this model of "drama." Once it has made a prediction about individual pages, another segmenting algorithm will go to work, fusing runs of similar pages in order to divide the document into document-parts, which again can be saved as a prediction.



from raw text files, after cleaning and tokenizing

Main database of page-level feature counts

Classifier: classification at page or volume level

A prediction file holding training data

Browser: allows user to search metadata and examine pages to create training data. Communicates with a database of metadata, and page API "offscreen."

Clustering: at page or volume level

Segmenter: fuse page-level predictions to make predictions about document-parts

A prediction file holding results to be confirmed or subsetted

Even a clustering algorithm could be used to create a "prediction" file. Because clustering is an unsupervised process, we won't know the name of the class a given cluster instantiates, but we could give it an arbitrary name. The set of objects included in that cluster (pages, page ranges, or volumes) can be saved as a prediction about membership in a particular class. (All of the predictions produced by clustering will have 100% confidence, unless it's a fuzzy clustering algorithm.)

Because predictions are so central to our workflow, it's going to be important to establish a standard for them, both as Java objects and as data files. I'm going to suggest that we use the ARFF file format developed for the Weka machine-learning toolkit: each prediction will be stored in a separate ARFF file. An alternate possibility would be make this a .json object. But it won't be at all hard to convert back and forth between those formats if we need to.

Here's documentation for the ARFF format:
http://www.cs.waikato.ac.nz/ml/weka/arff.html

And here's a link to Java code we can use if we want to read and write ARFFs:
http://weka.wikispaces.com/Creating+an+ARFF+file

Basically, to make a long story short, an ARFF begins with comment fields:

```
% 1. Title: Predicts Membership in class ‘gothic’
%
% 2. Sources:
% is-narrative.arff,0.75
% gothic-model-1.json
%
```

Those comments tell us that this prediction was produced by running gothic-model-1 on a workset defined by the prediction is-narrative, including everything that was above 75% probability in that workset. This is followed by a series of column declarations:

```
@RELATION gothic

@ATTRIBUTE htid         STRING
@ATTRIBUTE startpg      NUMERIC
@ATTRIBUTE endpg        NUMERIC
@ATTRIBUTE startpgpart  NUMERIC
@ATTRIBUTE endpgpart    NUMERIC
@ATTRIBUTE probability  NUMERIC
```

That is followed by essentially, a .csv file where the columns line up with the previous declarations.

```
uc2.ark:/13960/t9x061f8,1,1,120,0,0,0.52
uva.x000900036,6,366,0,0,0.81
yale.39002006534466,4,244,-1,-1,0.83
ien.35558005397654,3,302,0,28,0.6
mdp.39015021323699,4,330,0,0,0.7
loc.ark:/13960/t6639wj0p,-1,-1,
```

Let's call this whole file **`is-gothic.arff`**. We're eventually going to have multi-class predictions, and we may name those differently. But for right now let's focus on predictions that only describe data objects and their probabilities of membership in a single class. We'll prefix those with "is-."

We don't need to have all five "document part" columns in every prediction file. We could have just **`htid`**: the HathiTrust volume ID for a given volume.

Or we could have **`htid + startpg + endpg`**, which is a pretty tranparent description of a page range in a volume, including by default everything on both pages.

As we get more sophisticated, we may want to use **`startpgpart`** and **`endpgpart,`** which allow us to segment pages.

The meaning of 0 is "everything on the page." So

startpg	endpg	startpgpart	endpgpart
2	20	0	0

means everything between pages 2 and 20, including everything on both of those pages themselves — exactly the same meaning as if we had omitted the pagepart columns.

The meaning of a positive integer is “everything after/before line n.” In the case of **`startpgpart`**, obviously, it means “after”; **`endpgpart`** means “before.” So

startpg	endpg	startpgpart	endpgpart
2	20	12	18

Means “everything from page 2, line 12 through page 20, line 18.” Negative numbers are reserved for special purposes. -1 means “only the prose part”; -2 means “only the part that involves sequences of five or more lines with initial capitals (this may be verse, or drama, or an index). In all likelihood, we’ll only actually use these codes to distinguish page parts in cases where verse and prose are mixed on the same page for a long sequence of pages (e.g. a long poem with prose footnotes). If a negative number is used, **`startpgpart`** must always == **`endpgpart.`**

So:

startpg	endpg	startpgpart	endpgpart
2	20	-2	-2

Means only the verse (initial-capitalized) portion of pages 2-20 inclusive.

It’s true that lines of 20th-century poetry aren’t always capitalized. But generally, 20c volumes don’t present this odd problem where there’s verse at the top of the page and running prose footnotes at the bottom, over a long sequence of pages. We’ll only need negative “pagepart” codes for that peculiar sort of problem, and capitalization will in practice work to resolve it.

The **`probability`** column describes the probability that the data object in row *i* belongs to the class named in the “relation” declarations at the beginning of the file.

* * *

So there you have it: a tentative description of a “prediction” data object. Weka is open-source and includes Java classes that can be used to write and read ARFF files. We can extend this data definition as needed.