AvatolCV Developer Documentation

AvatolCV Internal Data Format

Morphobank Data Model

Morphobank uses a matrix-based repository. Each row pertains to a taxon, each column pertains to a particular morphologic character. When filled out, each cell (intersection of row and taxon) captures a particular character state for that character-taxon combination. For example, for an image of a skull, character “tooth xyz” might have states “present” or “absent”.

BisQue Data Model

Bisque uses an image-centric, rather than a matrix-centrix repository. Images have annotations that are key value pairs. A key for a leaf image might be “leaf apex angle”, a value might be one of “obtuse, acute, reflex, or straight”. In the context of character scoring, then, the character would be the key and the value would be the character state.

Normalized Data Model

In order to avoid having different code to deal with the two data formats, AvatolCV converts them both to a standard, normalized format, which is patterned as a key value pair. For BisQue data, each image is represented by a data file with a set of key value pairs. For Morphobank data, each cell in the matrix is represented by a data file with a set of key value pairs. Since it is possible for a single image to be represented in multiple cells for different characters, that image might be represented in a number of data files, where each one has the metadata for a particular cell.

To normalize the two worlds into a consistent form, each file has a number of entries that looke like

```
normalizedKey=normalizedValue
```

Both the normalizedKey and the normalizedValue have the same format which is

```
type:ID|name
```

A normalized key/value pair for Morphobank data might look like:

```
character:1847869|Upper I2 presence=characterState:4942258|I2 present
```

This allows the two notions of character and character state to be collapsed into a single key value pair, but preserving both the ID and value information for each. A normalized key/value pair for Bisque data might look like:

```
?:10932276|leaf apex curvature=?:?|acuminate
```

Since Bisque has no formalized notion of character and character state, the type isn't specified. Also, keys have an internal id, but values do not. The '?' character is used for "not specified".

The ID value is the ID value assigned internally by the BisQue repository or the Morphobank repository. For Morphobank, it's a row number from the database table. For Bisque, it's a unique hash-style ID.

This normalized data is stored in the directory called
`<installationRoot>/avatol_cv/sessions/<project_name>/normalized/imageInfo`

...where installationRoot is the folder in which AvatolCV has been installed (the parent folder of the avatol_cv folder), and project_name is the name of the project in either BisQue or Morphobank, and each file represents that data for a particular image.

```
10-249-195-126:imageInfo jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/Bat Skull Matrix/normalized/imageInfo
```

Files are named with the pattern

<imageID>_<oneUpNumber>.txt

...where imageID is the internal database ID of the image (ID within Morphobank or Bisque). oneUpNumber is needed because in Morphobank, a particular image might appear in more than one cell for a taxon, but the metadata for that image (character and character state values) will be different for each cell. So if there are four cells with that image, they filenames could be idX_1.txt, idX_2.txt, idX_3.txt, idX_4.txt.

The full contents of one of these metadata files for a Morphobank cell image is

```
10-249-195-126:imageInfo jedirvine$ cat 403274_1.txt
avcv_imageName=403274
avcv_annotation=point:90.8711340206186-40.9621212121212
?:?|taxon=?:778211|Artibeus jamaicensis
?:?|view=?:10477|Skull-ventral
character:1847869|Upper I2 presence=characterState:4942258|I2 present
```

Any key starting with avcv_ is reserved for special use. For Morphobank, notice how the notion of taxon and view are also held in the same normalized key value form. Next is an example of a Bisque normalized data file.

```
10-249-195-126:imageInfo jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/leafDev/normalized/imageInfo
```

Notice how the different format of internal ID used in Bisque makes for more complicated filenames.

```
10-249-195-126:imageInfo jedirvine$ cat 00-CeWjyaDiuKHN5g7JrhoLBA_1.txt
avcv_imageName=Smilax_herbacea_0195.jpg
avcv_timestamp=?:?|2015-05-01 10:14:17.678340
?:10932276|leaf apex curvature=?:?|acuminate
?:10932277|leaf base angle=?:?|reflex
?:10932278|leaf base curvature=?:?|convex
?:10932279|widest part of leaf lamina=?:?|base
?:10932280|leaf lamina length to width ratio=?:?|?
?:7231166|leaf apex angle=?:?|obtuse
```

AvatolCV Directory Structure

```
Top level:

avatol_cv/distro      // scripts for generating release bundles
        /docs         // documentation
        /images       // couple of images used by the UI
        /java         // the source code
        /license      // the license information for code that
avatolCV depends on
        /logs         // log files for sessions
        /modules      // algorithms that AvatolCV uses, and their
dependencies
        /sessions     // all the data for user sessions
        /sessionSummaries  //files that capture the essence of a run,
so that the results viewer can find results to show
```

Data for each project lives under
avatol_cv/sessions/<projectName>/

Data for each run lives under
avatol_cv/sessions/<projectName>/<runID>

If data was pulled from Bisque, it is stored prior to normalization under avatol_cv/sessions/<projectName>/bisque

If data was pulled from Morphobank, it is stored prior to normalization under avatol_cv/sessions/<projectName>/morphobank

Running Shipped Algorithms in Development Environment

The leaf pipeline, supported on Mac only, is

1. **basicSegmenter** (avatol_cv/modules/segmentation/yaoSeg)
2. **basicOrientation** (avatol_cv/modules/orientation/yaoOrient)
3. **shapeTextureScoring** (avatol_cv/modules/scoring/leafScore)

These dependencies are installed by AvatolCVInstaller, but if you are working with the AvatolCV source code and want to run these in your development environment, you will need to install some libraries that the basicSegmenter depends on. Refer to avatol_cv/modules/Darwin_Library_Installation_Instructions_MACOS.pdf

The dependencies should be installed into the 3rdParty folder, such that after installation, the landscape looks like this:

```
$ pwd
/Users/jedirvine/av/avatol_cv/modules/3rdParty
$ ls
cmake-3.0.0           darwin          libsvm          vlfeat
cmake-3.0.0.tar.gz    darwin.tgz      libsvm.tgz      vlfeat.tgz
```

The **partScoring** algorithm is housed in the github repository github.com/AVATOL/bat. If you want to run this in your development environment, you will need to install it by doing this on Mac (windows process the same, except for windows-ish details), starting at the avatol_cv directory:

```
$ pwd
/Users/jedirvine/av/avatol_cv
$ cd modules/scoring/batskullDPM/
$ git clone https://github.com/AVATOL/bat.git
(...)
$ cd bat   // Mac or Windows
$ git checkout -- new_integ  // gets the correct branch in play
```

Then, within MATLAB, navigate to the bat directory and type

```
%mex -setup           // sets the compiler for MATLAB
%compile              // runs compile.m to compile some files.
```

The **highClutterSegmenter** algorithm (which is not part of either functioning pipeline that includes a scoring stage), can be installed into a development environment by these steps, starting at the avatol_cv directory:

```
$ pwd
/Users/jedirvine/av/avatol_cv
$ cd modules/segmentation/hcsearchSeg/
$ git clone https://github.com/AVATOL/nematocyst.git
(...)
$ cd nematocyst
$ python setup.py    // python 2.7, Mac or Windows
```

MATLAB Version

AvatolCV works with MATLAB 2015b, but this association is only because the path to the MATLAB executable is hardcoded in algorithm wrapper scripts. If you have another version of MATLAB in play, and you want to run the shipped algorithms in your development environment, refer to the “How To Run AvatolCV Using Different Versions of Matlab” section of the AvatolCV User’s Guide.

What the Installer Does In Detail

AvatolCV is broken up into the following bundles:

- docs
- modules_3rdParty
- modules_osu
- java

usage: python updateAvatolCV.py <installRoot>

Here is the process updateAvatolCV.py uses:

1. ensures specified installation root dir exists

2. creates an avatol_cv dir under that installation root dir
3. determines a platform code - what system we are running on : win | mac | unsupported
4. generates the name of the downloadManifest file that it needs to download (from http://web.engr.oregonstate.edu/~irvineje/AvatolCV/) using:

    downloadManifest_<platform_code>.txt

5. generates a temporary name that it will save that file under

    downloadManifest_<platform_code>_new.txt

6. downloads that file and saves as that name

    new_manifest_pathname = downloadFile(avatolcv_root, new_manifest_filename, manifest_truename)

7. if this is not the first installation maneuver, the prior maneuver would have saved the file as...

    downloadManifest_<platform_code>_old.txt

    ...so we look to see if that file exists

8. If that old file exists we dump it to the console, then dump the new one to the console for sanity comparison by user
9. We determine which bundle names need downloading by comparing old and new.
 - If no old file exists, everything in new will be pulled.
 - If any bundle name represented in new is not represented in old, it is pulled
 - otherwise, we compare the datestamp (i.e. the version) associated with new to see if it is different than old. If so, we download.

 Note that we don't check for a more recent version, just difference with prior file. This will simplify regressing back to prior version if need be.

 If no bundles are needed, we declare we are up to date and exit, otherwise we continue
10. foreach bundle we need to download, we first uninstall the prior-installed version of that bundle using the

    allFiles_<bundle_name>.txt

    file that came with each bundle as a guide for which files to delete.

11. we delete the old manifest file and rename the new one to
downloadManifest_<platform_code>_old.txt

...so it can be checked at the later download

12. now we install each bundle by looking up the bundle filename from the downloadManifest and pulling from the distro site.

This yields a .tgz bundle that we then unwrap underneath the avatolcv dir