Adding New Algorithms into AvatolCV

AvatolCV was architected to allow new algorithms to be hooked in as needed, without changing any of the UI java codebase. This is accomplished through the use of a configuration file which tells AvatolCV what it needs to know about the algorithm. This file is named algPropertiesMac.txt or algPropertiesWindows.txt depending on which platform the algorithm is going to run on.

During an AvatolCV session, this file is used to both find the script/executable to run, and to generate a properties file (called a runConfig file) that will be passed to the script/executable with any runtime information it requires. AvatolCV will invoke the algorithm in a separate process with an invocation of this form:

<path_of_algorithm> <path_of_runConfigFile>

This means that you will most likely need to write a wrapper script around your algorithm. For example, if your algorithm takes six arguments, the wrapper script you write will accept the runConfig properties file, and use information from that to populate a command invocation with the proper arguments. The issue then becomes: what to put into the algProperties file so that the proper information shows up in the runConfig file.

We'll use a series of questions to help explain the details of how to construct an algProperties file and where to put it. Following that, as a reference, we'll show all the runConfig files shipped with AvatolCV, and example runConfig files are generated from those.

Most entries in the algProperties file are key=value entries. Some are more complicated statements. The questions and answers below will illuminate the details of the algProperties file.

Question: Where do I put my algorithm and algProperties<platform>.txt file, so AvatolCV will find it?

AvatolCV is installed under a root folder called avatol_cv. Below that is a modules folder which is where the algorithms live. The landscape is :

```
avatol_cv/modules/3rdParty
                /orientation
                /scoring
                /segmentation
```

Orientation, scoring and segmentation algorithms go under the relevant folders. Depended up libraries can be put into the 3rdParty folder if they are used by multiple algorithms.

AvatolCV will scan subfolders in the orientation, scoring, and segmentation folders to find algorithms. Let's say I have a new orientation algorithm called "orientX" that I want to hook in. I would create a folder called

```
avatol_cv/modules/orientation/orientX
```

...and create my algProperties file called

```
avatol_cv/modules/orientation/orientX/algPropertiesMac.txt
```

…for the Mac platform. If the algorithm were also supported on windows, then a file called algPropertiesWindows.txt would also be created, as there may be a need for some of the answers to be different.

Question: How do I control whether the algorithm appears in the UI as a segmentation, orientation, or scoring algorithm?

Though we have placed our algProperties file beneath the orientation folder, AvatolCV looks for an entry in algProperties to decide which flavor the algorithm really is. For this it uses the key “algType”, which can have the following values: segmentation, orientation or scoring. For our orientX example, we would have:

```
algType=orientation
```

Question: How do I specify how to invoke the algorithm?

We use a key called “launchWith”:

```
launchWith=orientationXRunner.sh
```

This script needs to be placed in the same directory as this algProperties file. The wrapper script, orientionXRunner.sh will be given one argument, the path of the runConfig file that is generated during the AvatolCV session.

Question: How do I control what name appears for the algorithm in the UI?

We use the key called “algName”:

```
algName=orientX
```

Question: How do I specify what description appears for the algorithm in the UI?

For this, we use the key “description”:

```
description=Algorithm orientX will orient images of yetis to a
consistent position.
```

Question: How do I specify runtime path information to pass to the wrapper script, within the runConfig file?

For this we use the “dependency” key, which has some special aspects. A dependency entry is of the form:

```
dependency:someKeyName=pathOfDependency
```

Notice that the word dependency is just the first portion of the key. The name of the key that will be passed into the runConfig file is the second half, to the right of the colon. So if we have

```
dependency:pathLibraryY
```

...as the key in algProperties, then the key in the generated runConfig file will be “pathLibraryY”.

Dependencies can be placed under the modules folder under your new algorithm folder or under the 3rdParty folder. In either case, use the special string “<modules>” as the root, and AvatolCV will replace that with the runtime path of the modules folder. For example,

```
dependency:pathLibraryY=<modules>/3rdParty/liby/liby-318
```

… this will result in an entry something like this in the runConfig file:

```
pathLibraryY=/Users/jedirvine/av/avatol_cv/modules/3rdParty/liby/liby
-318
```

If you specify a full path instead of using the <modules> string, that full path will be used as the value of the key/value pair in the runConfig file.

Question: How is AvatolCV's ad hoc typing system used to help manage synchronizing data between outputs and inputs of stages in the pipeline?

AvatolCV contains a typing system for data files, the purpose of which is to expose expectations about inputs and outputs for different stages of the pipeline. The original intention was to have the system validate output and input expectations between stages, but there was not time to complete this feature. Currently then, the only consumer of this type information is human eyes ,though AvatolCV's algProperties file parser will throw an error if the information is missing. This means the utility of this type information is in exposing, for easy reference, that data relationship in the algProperties file as a reminder of how things need to be aligned. As we'll see in the next sections, we include type information in both the inputRequired statements and the outputGenerated statements.

Here's an example to illustrate. Let's say a segmentation algorithm has two types of output files, and that the orientation algorithm that will follow it, assumes that it's input will be in those same formats. One output file type is a cropped version of the original image. For this, we concoct a typename such as "isolatedSpecimenImage". The other output is a mask version of that cropped image where the specimen of interest is rendered uniformly in green, the background is blue, and clutter to ignore is red. For this we make up the cumbersome but expressive typename "mask_SpecimenGreen_BackgroundBlue_ClutterRed". These two types would be used in the outputGenerated statement of segmentation and the inputRequired statement of the next stage, orientation. Someone who wishes to add a new orientation algorithm to the system that can consume the output of the segmentation stage, can look at the the algProperties file of segmentation to get a clue as to the nature of the output. Someone who wishes to add a new segmentation algorithm that the can generate outputs to be fed into that orientation algorithm, can look at the types represented in the algProperties file for orientation to get a sense of what's expected.

Since the type information for an output is not yet validated against the input type of the next stage, it means that in reality, any string could be used for the type, and even if the strings match, if the data is really in violation of expectations at an input, it will just result in a runtime error within the algorithm - AvatolCV won't complain.

Question: How will AvatolCV know how to manage input and output data for the segmentation stage?

First, let's consider the concept of "images in play" for a particular AvatolCV run. During a session, it's possible that particular images become excluded from consideration. This can happen for several reasons. For example, an image might have poor quality and be excluded by the user for that reason. We consider images that have not been excluded for any reason to be "in play" for a given run. At any stage in the pipeline then, AvatolCV needs to specify to the algorithm which images are in play.

Second, a word about how AvatolCV manages its pipeline processing. AvatolCV uses a class called AlgorithmSequence to track which algorithms are chosen during the run. Since segmentation and orientation are optional, it's feasible to execute a pipeline in any of the following configurations (depending on what the specific algorithms can do):

1. segmentation->orientation->scoring
2. segmentation->scoring (raw images already oriented as desired)
3. orientation->scoring (raw images need no segmentation)
4. Scoring (just scoring on raw input data)

As shipped, AvatolCV has two functional pipelines: one for shape aspects of simple leaves that uses segmentation->orientation->scoring, and one for presence/absence of small characters that has only a scoring algorithm in the pipeline. In general, as the session proceeds, AvatolCV leverages the AlgorithmSequence to know where it is in the pipeline, and thus where it needs to find inputs and where it should put outputs.

Segmentation can only be chosen as the first stage of the pipeline. Thus, AvatolCV knows its input needs to be the raw images from the data source, and knows which of those images are in play.

AvatolCV will create a file that we refer to as the inputManifestFile that contains a list of all the pathnames of the “in play” raw images, and put awareness of the inputManifestFile in the runConfig, so the algortihm’s wrapper script can access the manifest. In the algProperties file, we control what name is given to the inputManifestFile using the “inputRequired” statement.

```
inputRequired:<inputManifestFilename> refsFilesWithSuffix * ofType
rawImage
```

(See the question about the ad hoc typing system for an explanation of “ofType”) Note that the “inputRequired” statement is not in the form of a key=value style entry because there is too much information that needs to be conveyed.

Translating the above into english, it tells AvatolCV:

1. Input of some type is required, so go to the current folder where inputs are to be found (as per AlgorithmSequence)
2. Find all the files there that have any suffix (“*” means match everything)
3. Put the pathnames of all these files into a file called <inputManifestFilename>
4. Put <inputManfestFilename> into the current session data folder.
5. In this case the “ofType rawImage” is just there to make the parser of these lines happy. This portion will be relevant in the orientation and scoring stages.
6. Add a key/value into the runConfig file where the key is <inputManifestFilename> and the value is the full path of that file that was just created.

An example:

```
inputRequired:testImagesFile refsFilesWithSuffix * ofType rawImage
```

...would cause the following entry to be created in the runConfig file:

```
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/2016070
8_01/testImagesFile_segmentation.txt
```

Note that the "_segmentation" string is appended to the filename to ensure its uniquely used for that stage of the pipeline.

For output data, we use the "outputGenerated" statement. It is of the form:

```
outputGenerated:ofType <someType> withSuffix <someSuffix>
```

(See the question about the ad hoc typing system for an explanation of <someType>). The withSuffix keyword doesn't cause AvatolCV to do anything - it is just documenting what the algorithm is going to do. However, **IMPORTANT NOTE - AvatolCV needed a convention for knowing how to select the appropriate files for input to a given stage. The convention was chosen that files will be disambiguated using a suffix (vs a prefix, for example). This means an algorithm may need to be tweaked so that it generates output filenames as per AvatolCV's expectations, OR, the wrapper script of the algorithm will need to adjust the filenames to match AvatolCV's expectations.** So, regarding "withSuffix <someSuffix>", the expectation is that the suffix will occur before the file descriptor. For example, if the suffix specified was "_croppedMask", then the output filename is expected to look like "image12345_croppedMask.jpg"

For our segmentation example, we might have two types of output files generated, so we would have two outputGenerated statements:

```
outputGenerated:ofType isolatedSpecimenImage withSuffix _croppedOrig
outputGenerated:ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
withSuffix _croppedMask
```

Question: How will avatolCV know how to manage input and output data for the orientation stage?

If segmentation is skipped, and orientation is chosen as the first stage, AlgorithmSequence knows that the input needs to be raw images. If segmentation has been run, AlgorithmSequence knows that the input images will be the output of that prior stage. Let's look at this latter case to examine nuances and how they are controlled.

Let's say we have an orientation algorithm called orientX. It expects two types of input files:
-files that are cropped versions of the original files
-files that are mask file versions of those that are generated by some segmentation algorithm

This means we will need two inputRequired statements - one for each type of input file. Each inputRequired statement will again be of the form:

```
inputRequired:<inputManifestFilename> refsFilesWithSuffix
<someSuffix> ofType <someType>
```

In this case, though, the "refsFilesWithSuffix <someSuffix>" becomes relevant, because AvatolCV expects output files from a particular stage to be placed into a single folder. This means that the files need to be named with different conventions so they can be told apart by the next stage. The conventions are:

- If there is only one type of file output from a stage, and that one type is needed by the next stage, then use "`refsFilesWithSuffix *`", which means "only pay attention to the imageID (which starts the filename of each image) to when gathering pathnames of "in play" images.
- If there is more than one type of file output from a stage, and they are all needed by the next stage, then use "`refsFilesWithSuffix <someSuffix>`" in each inputRequired statement.

For example, for our cropped and mask-of-cropped case, we could have:

```
inputRequired:testImagesFile refsFilesWithSuffix _croppedOrig ofType
isolatedSpecimenImage
```

```
inputRequired:testImagesMaskFile refsFilesWithSuffix _croppedMask
ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
```

...which would cause the following two lines to appear in the runConfig file:

```
testImagesMaskFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/201
60708_01/testImagesMaskFile_orientation.txt
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/2016070
8_01/testImagesFile_orientation.txt
```

For output data, as in the segmentation stage, we use the "outputGenerated" statement. It is of the form:

```
outputGenerated:ofType <someType> withSuffix <someSuffix>
```

(See the question about the ad hoc typing system for an explanation of <someType>). The withSuffix keyword references the <someSuffix> value that is appended to the filename of each output file generated. **IMPORTANT NOTE - AvatolCV needed a convention for knowing how to select the appropriate files for input to a given stage. The convention was chosen that files will be disambiguated using a suffix (vs a prefix, for example). This means an algorithm may need to be tweaked so that it generates output filenames as per AvatolCV's expectations, OR, the wrapper script of the algorithm will need to adjust the filenames to match AvatolCV's expectations.** The suffix occurs before the file descriptor. For example, if the suffix specified was "_orientedMask", then the output filename might look like "image12345_orientedMask.jpg"

For our orientation example, we might have two types of output files generated, so we would have two outputGenerated statements:

```
outputGenerated:ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
withSuffix _orientedMask
outputGenerated:ofType isolatedSpecimenImage withSuffix _orientedOrig
```

Question: How will AvatolCV know how to manage input and output data for the scoring stage?

The process for the scoring stage is the same as for orientation, with the exception that no outputGenerated statements are needed because scoring is the final stage.

Question: If the scoring algorithm I want to add expects to score multiple characters at a time, how do I instruct AvatolCV to allow multiple selections at the appropriate screen?

Include this line in the scoring algorithm's algProperties file:

```
scoringScope=MULTIPLE_ITEM
```

Question: If the scoring algorithm I want to add expects to score one character at a time, how do instruct AvatolCV to allow only a single selection at the appropriate screen?

Include this line in the scoring algorithm's algProperties file:

```
scoringScope=SINGLE_ITEM
```

Question: If the scoring algorithm I want to add will score presence/absence of parts, how do instruct AvatolCV to

place the algorithm in that category on the screen where scoring algorithms are selected?

Include this line in the scoring algorithm's algProperties file:

```
scoringFocus=SPECIMEN_PART_PRESENCE_ABSENCE
```

Question: If the scoring algorithm I want to add will score shape or texture aspects, how do instruct AvatolCV to place the algorithm in that category on the screen where scoring algorithms are selected?

Include this line in the scoring algorithm's algProperties file:

```
scoringFocus=SPECIMEN_SHAPE_OR_TEXTURE_ASPECT
```

Question: If the scoring algorithm I want to add expects to divide images into training and scoring sets by what taxon they are, how do I instruct AvatolCV to enforce that?

Include this line in the scoring algorithm's algProperties file:

```
trainTestConcernRequired=true
```

Question: If the scoring algorithm I want to add relies on point annotation data (location of characters specified using Morphobank's point, rectangle, or polygon image annotation

feature), how do I instruct AvatolCV to manage that data, pass it to the algorithm, and render those annotations on images in the UI?

Include this line in the scoring algorithm's algProperties file:

```
includePointAnnotationsInScoringFile=true
```

Question: If the scoring algorithm I want to add expects training image files to have a particular suffix or not, how do tell AvatolCV this?

Include this line in the scoring algorithm's algProperties file:

```
trainingLabelImageSuffix=_orientedOrig
```

...or if there is no special suffix, use

```
trainingLabelImageSuffix=*
```

algProperties and runConfig files shipped with AvatolCV

For reference, here are the algProperties files for all the algorithms shipped with AvatolCV, and example consequent runConfig files.

(Note - the inputOptional directive is present in some algProperties files below, but is not fully implemented/honored)

Example algPropertiesMac.txt file for Segmentation (basicSegmenter):

10-249-195-126:yaoSeg jedirvine$ pwd

/Users/jedirvine/av/avatol_cv/modules/segmentation/yaoSeg

10-249-195-126:yaoSeg jedirvine$ cat algPropertiesMac.txt
launchWith=segmentationRunner.sh
algName=basicSegmenter
algType=segmentation
description=Algorithm simpleLeafSegmenter will isolate simple leaves from background clutter. It produces a mask file where the green area represents the isolated leaf. This algorithm is shipped with training data.
inputRequired:testImagesFile refsFilesWithSuffix * ofType rawImage
inputOptional:userProvidedGroundTruthImagesFile refsFilesWithSuffix _GT ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
inputOptional:userProvidedTrainImagesFile refsFilesWithSuffix * ofType rawImage
outputGenerated:ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed withSuffix _croppedMask
outputGenerated:ofType isolatedSpecimenImage withSuffix _croppedOrig

Consequent runConfig_segmentation.txt

10-249-195-126:20160708_01 jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01

10-249-195-126:20160708_01 jedirvine$ cat runConfig_segmentation.txt
segmentationOutputDir=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/segmentedData
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/testImagesFile_segmentation.txt

Example algPropertiesMac.txt file for Orientation (basicOrientation):

10-249-195-126:yaoOrient jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/modules/orientation/yaoOrient

10-249-195-126:yaoOrient jedirvine$ cat algPropertiesMac.txt
launchWith=orientationRunner.sh
algName=basicOrientation
algType=orientation
description=Algorithm simpleLeafOrienter will align leaves so their stems lie along the horizontal plane. Then it ensures that the apex and base are consistent in their position from image to image. This algorithm is shipped with training data.
dependency:pathLibSsvmMatlab=<modules>/3rdParty/libsvm/libsvm-318/matlab
dependency:pathVlfeat=<modules>/3rdParty/vlfeat/vlfeat-0.9.20/toolbox/vl_setup
inputRequired:testImagesMaskFile refsFilesWithSuffix _croppedMask ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
inputRequired:testImagesFile refsFilesWithSuffix _croppedOrig ofType isolatedSpecimenImage
outputGenerated:ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed withSuffix _orientedMask
outputGenerated:ofType isolatedSpecimenImage withSuffix _orientedOrig

Consequent runConfig_orientation.txt:

10-249-195-126:20160708_01 jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01

10-249-195-126:20160708_01 jedirvine$ cat runConfig_orientation.txt
orientationOutputDir=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/orientedData
pathLibSsvmMatlab=/Users/jedirvine/av/avatol_cv/modules/3rdParty/libsvm/libsvm-318/matlab
pathVlfeat=/Users/jedirvine/av/avatol_cv/modules/3rdParty/vlfeat/vlfeat-0.9.20/toolbox/vl_setup
testImagesMaskFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/testImagesMaskFile_orientation.txt
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/testImagesFile_orientation.txt

Example algPropertiesMac.txt file for Scoring (shapeTextureScoring):

10-249-195-126:leafScore jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/modules/scoring/leafScore

10-249-195-126:leafScore jedirvine$ cat algPropertiesMac.txt
launchWith=leafScore.sh
algName=ShapeTextureScoring
algType=scoring
description=Scores shape related characters of simple shapes such Monocot leaves.
scoringFocus=SPECIMEN_SHAPE_OR_TEXTURE_ASPECT
scoringScope=SINGLE_ITEM
trainingLabelImageSuffix=_orientedOrig
inputRequired:testImagesFile refsFilesWithSuffix _orientedOrig ofType isolatedSpecimenImage
inputRequired:testImagesMaskFile refsFilesWithSuffix _orientedMask ofType mask_SpecimenGreen_BackgroundBlue_ClutterRed
dependency:pathLibSsvmMatlab=<modules>/3rdParty/libsvm/libsvm-318/matlab
dependency:pathVlfeat=<modules>/3rdParty/vlfeat/vlfeat-0.9.20/toolbox/vl_setup

Consquent runConfig_scoring.txt:

10-249-195-126:20160708_01 jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01

10-249-195-126:20160708_01 jedirvine$ cat runConfig_scoring.txt
scoringOutputDir=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/scoredData
trainingDataDir=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/trainingDataForScoring

pathLibSsvmMatlab=/Users/jedirvine/av/avatol_cv/modules/3rdParty/libsvm/libsvm-318/matlab
pathVlfeat=/Users/jedirvine/av/avatol_cv/modules/3rdParty/vlfeat/vlfeat-0.9.20/toolbox/vl_setup
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/testImagesFile_scoring.txt
testImagesMaskFile=/Users/jedirvine/av/avatol_cv/sessions/leafDev/20160708_01/testImagesMaskFile_scoring.txt

Example algPropertiesMac.txt file for Scoring (partsScoring):

10-249-195-126:batskullDPM jedirvine$ cat algPropertiesMac.txt
launchWith=batSkullScore.py
algName=partsScoring
algType=scoring
scoringFocus=SPECIMEN_PART_PRESENCE_ABSENCE
scoringScope=MULTIPLE_ITEM
trainTestConcernRequired=true
includePointAnnotationsInScoringFile=true
requiresPresentAndAbsentTrainingExamplesForCharacter=true
description=Scores presence/absence of simple parts, for example, types of teeth.
inputRequired:testImagesFile refsFilesWithSuffix * ofType rawImage
outputGenerated:ofType isolatedSpecimenImage withSuffix _scored
trainingLabelImageSuffix=*

Consquent runConfig_scoring.txt:

10-249-195-126:20161006_01 jedirvine$ pwd
/Users/jedirvine/av/avatol_cv/sessions/AVAToL Computer Vision Matrix/20161006_01
10-249-195-126:20161006_01 jedirvine$ cat runConfig_scoring.txt
scoringOutputDir=/Users/jedirvine/av/avatol_cv/sessions/AVAToL Computer Vision Matrix/20161006_01/scoredData
trainingDataDir=/Users/jedirvine/av/avatol_cv/sessions/AVAToL Computer Vision Matrix/20161006_01/trainingDataForScoring
testImagesFile=/Users/jedirvine/av/avatol_cv/sessions/AVAToL Computer Vision Matrix/20161006_01/testImagesFile_scoring.txt