Bee Foraging Maps Code Documentation

Last Updated 7.8.15

Files:

FullProgram.py

Functions necessary for running the entire process. Calls all three files described below. The 2 main functions are totalSVR and totalGauss. These functions both run through the entire process of calculating metrics, finding a fitting function, and creating a density map. The only difference is in the machine learning method they use.

If you want to implement new machine learning methods this is the place to do so. Both totalSVR and totalGauss follow the same outline, so a new function can be modeled on either of these. Simply replace the fit line with the new method.

DensityAlignment.py

This file contains functions necessary to analyze an entire transect and align images with the collected data from transecting. The needed data can be found in the BFS Master Survey Spreadsheet.

MachineLearning.py

This file contains the functions that assist in performing machine learning processes, as well as in producing the final density maps.

ImageProcess.py

This file contains all of the functions needed to process the images. This includes functions for calculating each metric, as well as functions that create tiles within a larger images and calculate metrics for the entire image.

Function Descriptions

ImageProcess.py

- **getSub(n, imageName, overlap)**
  - n is the size of one side of the tile. Tiles are square
  - imageName is a string containing the name of the image you are working on
  - overlap is the percentage by which to shift over. I.e. if overlap = 0.3 then the overlap is 70 %. Overlap must be a number that divides evenly into the tile (i.e. 0.75, 0.5, etc.)
  - Computes the metrics on each subtitle in the image. A subtile is a tile the size of the shift. These subtitles will be used to compose larger tiles later on. All metrics are stored in a dictionary keyed by the upper left corner of

the subtile.
    - Outputs the dictionary of metrics
- **allMetrics(dictionary, n, im, overlap)**
    - dictionary is the output of getSub
    - n is the tile size (same as getSub)
    - im is the PIL image (load the image of imageName)
    - overlap is the percentage to shift (same as getSub)
    - allMetrics computes the metrics on larger tiles of size n based on the smaller tiles computed in getSub and stored in the input dictionary. This two step process was implemented to avoid recalculating metrics on overlapping sections of tile and speed up the code.
    - allMetrics outputs an array of metrics. Each metric is in one row
        - For example, the first row of the array is the average red value for each tile
        - Tiles are ordered from left to right and top to bottom in the image.
- **calcMetrics(imageName, tileSize, overlap)**
    - A wrapper function to calculate the metrics for each tile of the image. Calls both getSub and allMetrics. returns a list of list of all the metrics for tiles in an image
    - imageName is the string name of the image
    - tileSize is the size of one side of a tile (same as n above)
    - overlap is the shift percentage between tiles (as above)
- **scaleMetrics(metricArray)**
    - Takes in an array of metrics. Scales these metrics so that all have a mean of 0 and a standard deviation of 1.
    - This step prevents bias in the machine learning algorithm due to different ranges for different metrics. For example, before scaling some metrics have values of 0.004 while others have values in the thousands. This can bias the machine learning algorithms to weight one metric more than others if uncorrected.
    - returns both the array of scaled metrics and the scaler used to scale them. This same scaler must be used on both the training data and the data from the entire image on which the algorithm is making predictions.
- **oneDensity((i,j), w, h, imageName)**
    - This is a debugging function that calculates the density of one tile based on a pre-made training set. It takes in the coordinates of the tiles within the image (i,j), the width and height of the desired tile, and the name of the image (imageName).
    - outputs the density of this tile.
    - This function was used to check if densities plotted were lining up with how they were calculated. It is not used in the actual functions.
- **oneDensOverlap((i,j), n, imageName, overlap, subTileDict, fit, scaler)**
    - Calculates the density of one tile, but uses the overlap process.
    - n is the tile size
    - imageName is the string name of the image (‘hi.jpg’)
    - subTileDict is the dictionary of subTile metrics from getSub
    - fit is the machine learning algorithm fit used to calculate densities

    - scaler is the scaler that was used on the training data for the machine learning and is used to scale the image metrics.
    - returns the density of the tile in a list.
- **allDensOverlap(n, imageName, overlap, densityList, metricList, fit, scaler)**
    - Calculates all densities for an entire image with overlap. This is the function used to compute densities in the final process.
    - n is the tileSize
    - imageName is the name of the image
    - overlap is the shift percentage between tiles
    - densityList is a list of densities in the same order as the metricList
    - metricList is a list of metrics in the image.
    - Returns a list of densities, again in order from left to right and top to bottom.
- **allDensities(w,h,imageName)**
    - Another debugging function. Calls oneDensity to calculate non overlapping densities on the entire image.
    - w is the width of the tiles
    - h is the height of the tiles
    - imageName is the name of the image in a string.
- **trainMetrics(imageName, density)**
    - imageName is the name of the image in a string.
    - density is the density of flowers in that image.
    - trainMetrics calculates each of the features for this image and returns a list with [metrics, density] together.
- **allTrainMetrics(imageList, densityList)**
    - takes in a list of image names and a list of densities. They must be in the same order.
    - imageList is a list of image names
    - Calculates the metrics on each image using trainMetrics.
    - Returns a list of metrics and a list of densities, again in the same order.
    - densityList is not changed in this process.
- **allTrainMetricsTransect(imageList, densityList)**
    - Similar to allTrainMetrics, it takes in a list of images and a list of densities.
    - imageList is a list of PIL images (NOT NAMES)
    - Still returns a list of metrics and a densityList
    - densityList is unchanged from the input.
    - This function is used to calculate training data when the input is a transect, which is divided into a list of subImages corresponding to the quadrants where data was taken.
- **trainMetricsTransect(image, density)**
    - Takes in a PIL image and a corresponding flower density.
    - Calculates each feature on this image.
    - returns [metrics, density]
    - metrics is a list containing the value of each metric on this image.
    - This function is called by allTrainMetricsTransect.
- **colorAvg(im)**
    - takes in im, a PIL image.

    - Calculates the average red, green, and blue values.
    - Return red, green, blue.
- **findYellow(im)**
    - takes in a PIL image
    - Calculates the number of pixels in the image that are within the given HSV ranges.
    - The ranges can be changed manually in the code (minHue, maxHue)
    - outputs the portion of yellow pixels (# yellow pixels/total pixels in image)
    - This version of the function is not used in the final process.
- **colorVariance(im)**
    - takes in a PIL image
    - calculates the percentage of variance (the standard deviation) of the hue histogram.
    - This function creates a histogram and then uses numpy to find the standard deviation.
    - returns a value representing the variance.
- **countEdgePixels(im)**
    - Takes in a PIL image
    - Using python ImageFilter's built in function FIND_EDGES, find all edges in the image.
    - Counts the number of edge pixels using thresholding
    - Divides this count by the number of pixels in the image.
    - Return a float of the portion of the image that is edge pixels
    - This is one representation of texture in the image.
- **textureAnalysis(im)**
    - Takes in a PIL image
    - Looks at intensity in a 7x7 kernel.
    - If the differences in intensity are greater than a given threshold, counts as a "texture" feature.
        - Threshold can be changed in the first line of this function.
        - the size of the kernel can also be changed (second line of the function, n)
        - Note that if you pass in images smaller than the kernel size at any point in the run the calculations will fail and you will get an error message.
            - If this error occurs it prints the width of the image and prints "Oops", then waits for user input.
- **findYellowFast(im)**
    - The same idea as findYellow
    - Takes in a PIL image
    - Uses Hue, Saturation, and Value thresholds to determine the color. Change these at the start of the function.
    - Calculates the portion of pixels that meet the threshold requirements in the image.
    - Uses map to speed up the process.
    - This function is used instead of findYellow
- **getHSV((r,g,b))**

- A helper function to find HSV values quickly.
- Takes in a tuple of red, green, and blue values for the pixel
- Outputs a tuple of HSV values for the pixel.

- **analysis(avg, yellow, var, edges, texture)**
  - Takes in lists of average color values, yellow values, variance values, edge values, and texture values for an image.
  - Calculates the average, min and max for each of these lists.
  - Not used for the final process, but useful for debugging and checking if each feature actually varies between different tiles in an image.

MachineLearning.py

- **svrAlg(X, densities)**
  - Takes in X, a list of training metrics, and densities, a corresponding list of density values.
  - Uses an SVR algorithm to calculate a fitting function. Change parameters of the machine learning algorithm here. (in clf = SVR(…)
  - returns the fitting function calculated by the machine learning.
- **gaussReg(metrics, densities)**
  - Takes in metrics, a list of training metrics, and densities, a corresponding list of density values.
  - Uses a Gaussian Regression algorithm to calculate a fitting function.
    - Change parameters here (gp = GaussianProcess(…))
  - Returns the fitting function calculated by the machine learning
- **densMap(fit, metricArray, n, overlap, imageSize, imageName)**
  - Fit is the machine learning fitting function
  - metricArray is an array of metrics on the image
  - n is the tile size
  - overlap is the shift percentage between tiles
  - imageSize is the width and height of the image in a list or tuple
  - imageName is a string with the file name of the image in it
  - Uses the fit to predict the densities for each tile in the metric Array.
  - Creates a contour plot of these densities
    - Uses linear interpolation to fill in gaps between tiles
    - Each density is considered to be at the center of the tile it was calculated from
  - Returns the grid of densities calculated through interpolation
  - Saves contour plot of densities overlaid with a map as ‘ContourPlot.jpg’
  - This function is not used in the final process.
- **densMapShort(densities, imageName, overlap, n)**
  - densities is a list of densities in the image
  - imageName is a string containing the file name
  - overlap is the shift percentage between tiles
  - n is the tile size
  - Interpolates between the densities, assuming that each is the density at the center of the tile it represents.
  - Plots these densities as a contour map

    - Saves the contour map as ‘TransectContour.jpg’ (plt.savefig(…))
        - Change the name here if you don’t want to overwrite old plots.
    - Does not return anything.
- **testDensMap(n, overlap, imageName)**
    - A debugging function that creates a basic density map. Not used in the final process.
    - Takes in n, the tile size
    - overlap is the shift percentage between tiles
    - imageName is a string containing the file name of the image.
    - Returns nothing
- **overlayMap(mapName, contourName)**
    - Another debugging function. Takes in names of 2 images, one for the map of the area you are interested in and one for the contour plot.
    - Overlays these images to create a combined map and saves as ‘OverlayMap.jpg’
    - Does not return anything
- **learnSVR(metricArray, n, overlap, imageSize, fit)**
    - A wrapper function for the machine learning algorithm and post-processing using SVR.
    - This function is not used in the final process
    - Returns nothing
- **learnGauss(metricArray)**
    - A wrapper function for the machine learning algorithm and processing using Gaussian Processes.
    - This function is not used in the final process
    - metricArray is an array of metrics on the whole image.
    - Used for debugging Gaussian plots.

DensityAlignment.py

- **divideTransect(Start, End, imageName)**
    - Start is a tuple containing pixel coordinates for where the transect begins
    - End is a tuple containing pixel coordinates for where the transect ends
    - imageName is a string containing the file name of the image.
    - Calls either transectVert of transectHoriz depending on the direction of the transect.
    - Splits the transect into 2x1 meter images corresponding to the quadrats that are sampled during transecting at the field station.
    - Returns a list of these PIL images.
    - Each image can be opened by using the command imList[X].show(), where X is the index of the image you want in the list.
- **transectHoriz(Start, End, image)**
    - Takes a horizontal transect and traverses it
    - Start is a tuple containing pixel coordinates for where the transect begins
    - End is a tuple containing pixel coordinates for where the transect ends
    - image is a PIL image containing the transect.
    - Returns a list of 2x1 meter PIL images

    - Called by divideTransect
- **transectVert(Start, End, image)**
    - Takes a vertical transect and traverses it
    - Start is a tuple containing pixel coordinates for where the transect begins
    - End is a tuple containing pixel coordinates for where the transect ends
    - image is a PIL image containing the transect.
    - Returns a list of 2x1 meter PIL images
    - Called by divideTransect
- **saveTransect(imageList, start)**
    - imageList is a list of PIL images representing a transect
        - For example, the output of divideTransect
    - Saves each image in the list with the name format 'X.jpg'
        - X is an integer
    - The first image is saved as 'start.jpg' and each subsequent image adds 1 to the integer.
    - Returns nothing
    - Use this function is you want to use multiple transects as training data. Save each one with non overlapping start values (i.e. 1 and 51, etc.).
- **main()**
    - A helper wrapper function to speed up testing
    - Checks if the length of the transect image list is correct (it should always be 50 because each transect is 50 meters long).
    - Returns the list of transect images.
    - Not used in any final processes.

Full Program.py

- **main()**
    - A helper function to run quickly
    - No longer used in any final process
    - Useful for debugging
- **totalSVR(densityList, imageName, tileSize, overlap, trainingType)**
    - densityList is a list of densities for training data
    - imageName is a string containing the file name of the image
    - tileSize is the size of the tiles to use
    - overlap is the shift percentage between tiles
    - trainingType is the type of training data to use
        - 0 for a transect
        - 1 for a list of saved pictures
            - Named as '1.jpg', '2.jpg', etc.
        - 2 for a previous data set (already calculated metrics on, loads in metrics from file).
        - 3 for any other kind of data. You must enter the data here manually in the format you want.
    - Executes SVR algorithm on the given training data
    - Calculates densities for the entire image
    - Interpolates between densities

- Creates a density map overlaid with the original image.
- Saves densities to a file
- Saves training data to a file
- Anything that is saved to a file you can use later to speed up the process when you make changes to a different file. Change if True to if False and make sure there is an else statement indicating that it should read from a file.
    - This prevents the program from needing to recalculate values unnecessarily, speeding it up a great deal!
- Does not return anything

- **totalGauss(densityList, imageName,tileSize, overlap, trainingType)**
    - densityList is a list of densities for training data
    - imageName is a string containing the file name of the image
    - tileSize is the size of the tiles to use
    - overlap is the shift percentage between tiles
    - trainingType is the type of training data to use
        - 0 for a transect
        - 1 for a list of saved pictures
            - Named as '1.jpg', '2.jpg', etc.
        - 2 for a previous data set (already calculated metrics on, loads in metrics from file).
        - 3 for any other kind of data. You must enter the data here manually in the format you want.
    - Executes Gaussian algorithm on the given training data
    - Calculates densities for the entire image
    - Interpolates between densities
    - Creates a density map overlaid with the original image.
    - Saves densities to a file
    - Saves training data to a file
    - Anything that is saved to a file you can use later to speed up the process when you make changes to a different file. Change if True to if False and make sure there is an else statement indicating that it should read from a file.
        - This prevents the program from needing to recalculate values unnecessarily, speeding it up a great deal!
    - Does not return anything
- **makePicList(numSites)**
    - Creates a list of image names
        - ['1.jpg', '2.jpg', '3.jpg', …]
    - numSites is an integer indicating the number of image names to include in the list.
    - For example if you save a group of transect images, you can call makePicList(50) to save the names of 50 images into a list for ease of access by other functions.
    - Returns a list of strings containing file names
    - If you change the naming convention for training files change the names created by this list.

How to Run

1. Create a training set
    - Single Transect:
        - Use divideTransect to split the transect into sub images that match the data collection
        - Create a list or an array with the transect data from the BFS master survey in it. Use the flower numbers as measures of density. For how to do this see totalSVR under “if trainingType == 0”. The first few lines create a density list of training data from one of the transects we used.
    - Multiple Transects
        - Use divideTransect on each image to split the transect into sub images
        - Use saveTransect to save each of these images.
            - Start the first transect at 1, the next at 51, etc.
        - Create a list or an array with the transect data from each of these transects (see for a single transect). This list must be in the same order as you saved the images. I.e. the transect you started at 1 should come first in the list, the one started at 51 should come second, etc.
    - Other list of pictures
        - For other pictures name each one as ‘1.jpg’, ‘2.jpg’, etc.
        - Create a density list in the same order manually, either from collected data or by estimation
    - Use an existing data set
        - Make sure your data set is saved in a .txt file with the appropriate name, matching the filename in the code.
2. Choose an overlap and tile size
    - Different overlaps and tile size give different resolutions of density maps. In general tile sizes of less than 100 have been working best so far. Remember that overlap must be a percentage that can easily fit into the image (i.e. 0.5 with a tile size 50). Basically all subtile must be the same size.
3. Choose an algorithm
    - Currently both SVR and Gauss are implemented
    - Choose one that will work best for this, or try both.
4. Check File Names
    - Several files are saved in the process of running. If you want to save a previous run make sure that you have changed the names files are saved to so that they do not overwrite an older file. Also make sure that any files you choose to read in are named correctly and in the correct directory.
5. Run the program
    - Call either totalSVR or totalGauss, depending on your choice in step (3)

- densityList is the list of densities you created in step (1)
- imageName should be the name of the image you are working on. Make sure this image is in your working directory.
- tile size and overlap are the values you chose in step (2)
- Training type is the kind of training set you created in step (1)
    - For a single transect trainingType is 0
    - For multiple transects or a list of pictures the trainingType is 1.
    - For preexisting data the training type is 2.
    - For any other kind of data set the training type is 3.
        - Note that if you create another kind of training set you must implement it yourself in the code under if trainingType == 3:
            - This option exists to allow for testing new kinds of data without modifying existing code.
            - For example if you have a set of only a few images named differently from convention, input the name list here and you can run the code without changing the names in the code or the names of your images.

6. Celebrate because you now have a density map!
    - Hopefully