Leaf Shape Clusterization and Tree Association

Yuqing Tan (Beatrice)

Supervised by Prof. Ian Stavness

University of Saskatchewan, CMPT400

Abstract

This is a university undergraduate level project on unsupervised machine learning of leaf shapes. It borrows the idea and the dataset from the existing Leafsnap system, which is presented as a mobile app that does automatic plant species identification. Since I am only provided with the segmented (black-and-white) images, extracting features for clustering also becomes a task in this project. I made several models with Matlab to capture features from leaf contours and learned about advantages and drawbacks of them based on clustering result of Weka software. A simpler model sometimes does a better job than a complicated model as long as it catches the essences of images. Two major directions for future study can be refining features extraction models and adding new flavors of machine learning on leaf shapes.

1. Introduction

There's a project across North America and the UK called "Leafsnap: An Electronic Field Guide", which aims to build an automatic plant species identification system based on visual clues as a mobile app. This system connects to a database of around 180 species in northeastern America, which stores high definition photos of plant parts (e.g. leaf, flower and fruit).

When a user turns on the app and snaps a photo in, the system does the followings:

(a) classifying the image into one of leaf, flower and fruit
(b) segmenting the image (removing the background and returning a binary value image)
(c) associating with species based on curvature features
(d) providing top 10 matches and details of those species to the user

This project can accelerate species identification for scientific use, share knowledge among the general public and improve the current geographic information system (GIS).

My CMPT400 project simulates the species association of the Leafsnap project by doing unsupervised machine learning. The dataset was obtained from Leafsnap's official website[1] as a package of original and segmented images. I followed the three steps in this project:

(a) sampling: selecting images from the dataset
(b) extracting: collecting features from images as plain text data

[1] leafsnap.com

(c) clustering: a common method of unsupervised machine learning, which puts the images into a specified number of clusters, and is measured by grouping (if images of same species also belong to the same cluster) and distinguishing (if images from different species are put into different clusters)

Matlab numerical computing and Weka machine learning software are used for extracting and clustering respectively. I spent relatively more time on extracting than the other two steps, because it has the most changeable factors (i.e. different models of extracting features can lead to totally different clustering result). For clustering, several algorithms are already built into Weka that I can use directly.

2. Sampling

In the dataset, there are 50 clear segmented images on average for each species. Some species have less (e.g. 15) and some have more (e.g. more than 100).

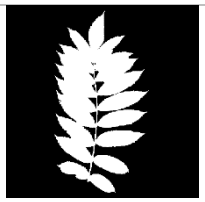
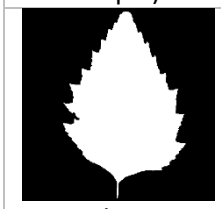
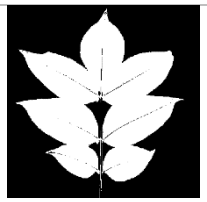
When designing the first two models of extracting, I used a sample of three species with a total of 76 images.

		
Acer saccharum (Sugar maple)	Acer saccharinum (Silver maple)	Acer pseudoplatanus (sycamore maple)

Figure 2.1

After improving automation on my script, I first increased the sample to 100 images (20 species * 5 images each) and increased again to 300 images (20 species * 15 images each) to create a simple and fair coverage.

Thumbnails, Latin and English names (in alphabetical order of Latin names)

				
Acer palmatum (Japanese maple)	Acer saccharinum (Silver maple)	Acer saccharum (Sugar maple)	Aesculus flava (Buckeye)	Ailanthus altissima (Ailanthus)
				
Betula nigra (River birch)	Catalpa speciose (Catalpa)	Evodia daniellii (Evodia)	Liquidambar styraciflua (Sweetgum)	Metasequoia glyptostroboides (Dawn redwood)

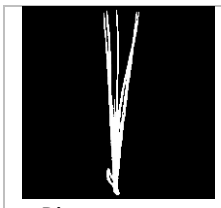
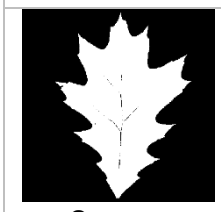
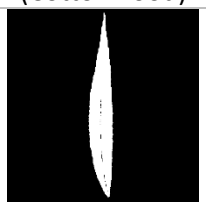
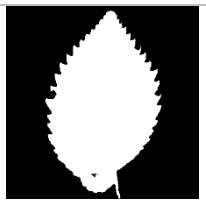
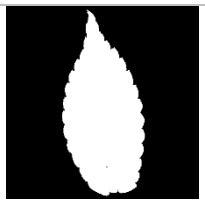
 Pinus peucea (Balkan pine)	 Populous deltoides (Cottonwood)	 Prunus sargentii (Sargent's cherry)	 Ptelea trifoliate (Wafer ash)	 Quercus montana (Chestnut oak)
 Quercus velutina (Black oak)	 Salix babylonica (Babylon willow)	 Staphylea trifolia (American bladdernut)	 Ulmus Americana (American elm)	 Zelkova serrata (Japanese elm)

Figure 2.2

3. Extracting

Each image contains condensed visual clues for human to get a perception. Extracting features is the process of asking myself how I perceive the images and turning these perceptions into algorithms. I based my implementation on Matlab's built-in functions regionprops() and bwboundaries(). The former one can return many basic geometries of a binary image, such as area, perimeter, orientation and convex hull; the later one can return an array of boundary points specifying X and Y coordinates. I can wrap up my attempts over the 4-month period (from December 2015 to March 2016) into three major models: 3-meta-properties model, round-scanning model, concave triangle model.

(1) 1st model: 3-meta-properties model (December, 2015)

This model was motivated by quantitative measurement of epidermal cell shape. **Three features** will be calculated from axis lengths, area and perimeter. I have intentionally made the range of these features to be (0, 1) to be suitable for calculating distance during clustering.

Property	Calculation	Range	Indication
Aspect ratio	(short axis) / (long axis)	(0,1)	value: small → big shape: oblong → round
Compactness (or fluffiness in the opposite sense)	area / ((short axis) * (long axis))	(0,1)	value: small → big shape: fluffy → compact
Circularity (or jaggedness in the opposite sense)	(area * pi * 4) / (perimeter^2)	(0,1)	value: small → big shape: jagged → smooth

Figure 3.1

Long axis is also called major axis length in Matlab, or the width when image is in landscape orientation;
Short axis is also called minor axis length in Matlab, or the height when image is in landscape orientation;

Advantage: independent of leaf orientation; easy to script.

Drawback: too simple, ignoring a lot of details.

(2) 2nd model: round-scanning model (January, 2016)

This model was based on a traditional thought about capturing curvatures – differential technique from given data. My design is even simpler which only collects data points without finding a best-fit curve. It can be viewed as **one feature with n values or n trivial features**.

It picks n points evenly distanced on the leaf boundary and collects a tuple (D/R, A) from each point, where:

D is the distance between the point and the centroid (in pixels);
R is the average radius of the shape (in pixels);
A is the angle between two vectors – one from centroid to the point and the other from centroid to root position;

There will be n*2 values collected from each picture; range for D/R is (0, infinity), for A is [0, 360)

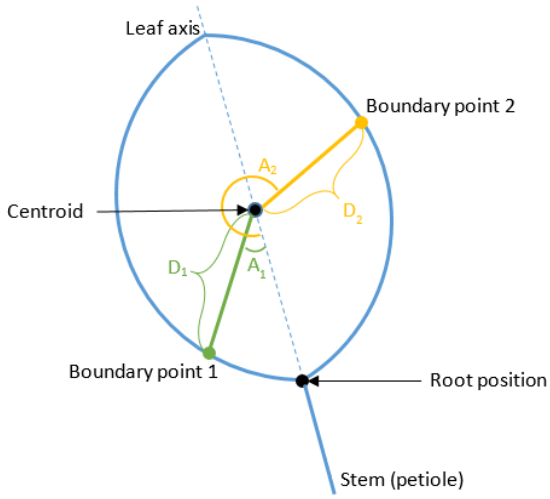


Figure 3.2.1

An example of successfully collected data (D/R, A) being plotted.

The leaf

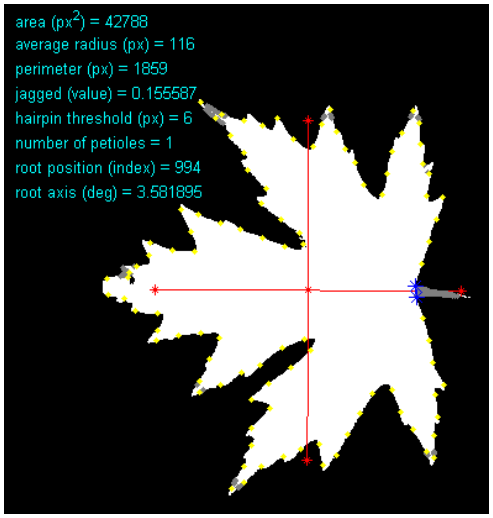


Figure 3.2.2

The plot

X axis is A (angle in degrees); Y axis is D/R (the ratio)
The plot looks like the leaf's shape being stretched from its center. The region below the blue curve is inside the leaf and above is outside the leaf.

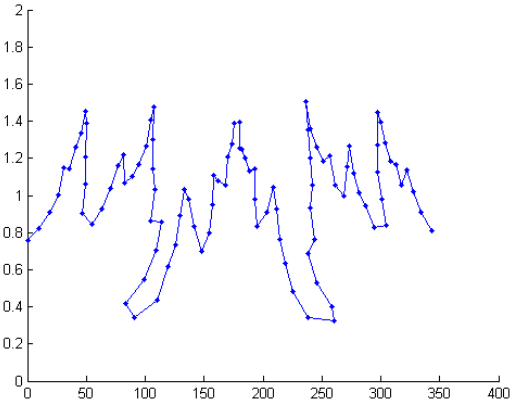


Figure 3.2.3

Advantage: first attempt on collecting more features than meta properties only.

Drawback: requiring the knowledge of shape orientation (leaf stem), and highly sensitive to it, since wrong orientation will mess up the sequence of data; amplifying the noise; not solving the leaf shapes which are naturally not symmetric.

Two algorithms were implemented to build this model: one is **detecting the stem** if it appears in the picture, the other is **finding the most symmetric axis** of the picture. They both require "walking" along the boundary (i.e. iterating through the array of boundary points).

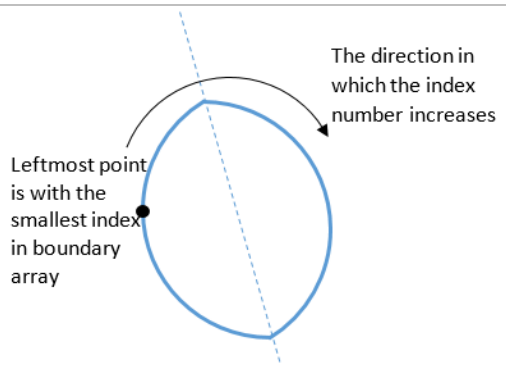


Figure 3.2.4

(a) Detecting the stem

This is not globally applicable, since in Leafsnap project, they actually intended to remove stems from all images, but I still got a lot of them with the stems to test this algorithm.

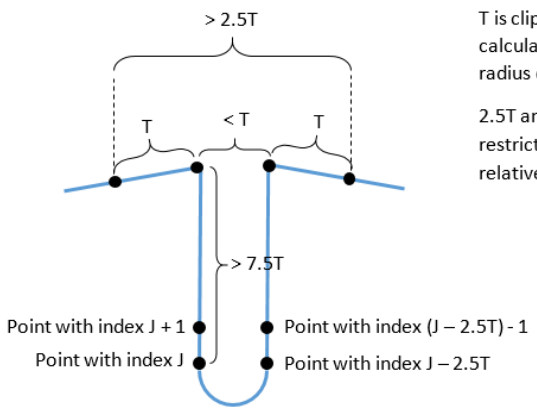
Steps:

1. Defining a threshold T (e.g. T=5) such that a clip shape with a width smaller than this threshold will likely be a stem. The threshold is obtained from average radius divided by a constant to adjust to the image's scale.
2. Iterating through the array of boundary points and calculating the Euclidean distance between points J and J - 2.5T (assuming J is the index in the array).
3. If the above Euclidean distance is smaller than threshold T, then it might represent part of a clip shape. Entering into a loop to decide the length of the clip.

```
If dist(J, J - 2.5T) < T
  P ← J
  Q ← J - 2.5T
  While dist(P, Q) < T
    If dist(P + 1, Q - 1) < T
      P ← P + 1
      Q ← Q – 1
    Else
      Break
    End if
  End while
End if
```

4. The clip shape has to pass some more validations to be recognized as a stem (see the below illustrations).
5. If multiple stems are identified in an image, pick the one with smallest Euclidean distance to the centroid.

Clip shape 1: possibly a leaf stem



T is clip shape threshold, calculated from average radius divided by a constant

2.5T and 7.5T are picked as restrictions to achieve relatively accurate result

Figure 3.2.5

Clip shape 2: not a leaf stem

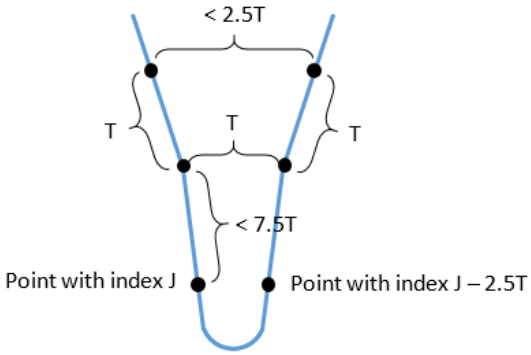


Figure 3.2.6

Examples of a success and a failure of this algorithm (clip shapes are marked out in **grey**, and stem is marked by **blue star**).

Success on a silver maple leaf

The algorithm can distinguish sharps from the real stem.

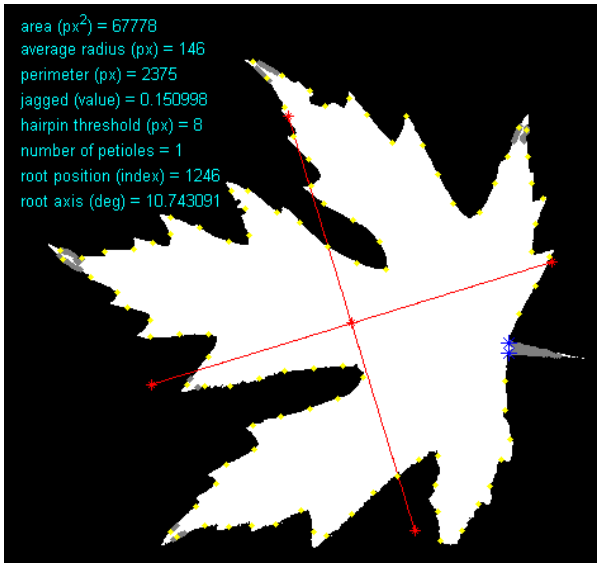


Figure 3.2.7

Failure on dawn redwood

The algorithm gets confused with dendritic shapes, and it doesn't distinguish intruding parts from protruding parts.

Figure 3.2.8

(b) Finding the most symmetric axis

This was based on angle calculation between interpolated points.

sumAngles = $(a_1 – b_1)^2 + (a_2 – b_2)^2 + … + (a_n – b_n)^2$ If sumAngles is zero, the sub-shape is perfectly symmetric with its axis passing point J.	This illustration picks 3 interpolated points on each side of J; while in my actual implementation I picked 6 points on each side to improve the accuracy.

In my algorithm the point with smallest sumAngles among all boundary points is picked as root position; the line connecting root position and the centroid is taken as the stem.
An example of the success of this algorithm on sycamore maple (stem position marked out in **blue**).

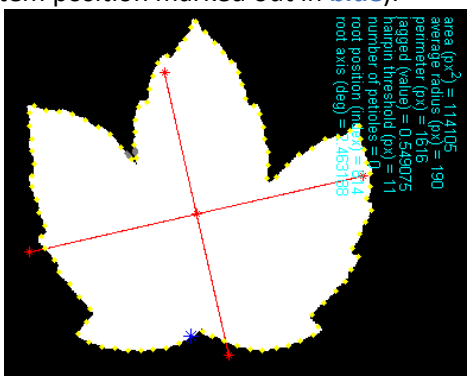


Figure 3.2.9

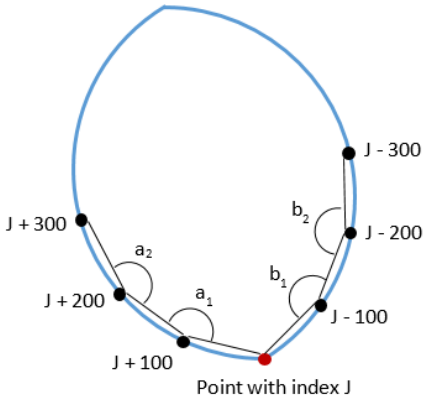


Figure 3.2.10

An overview of stem/symmetry detection on a sample of 100 images (20 species * 5 images each).

X axis is the species; Y axis is the correct rate (compared with human judgement).

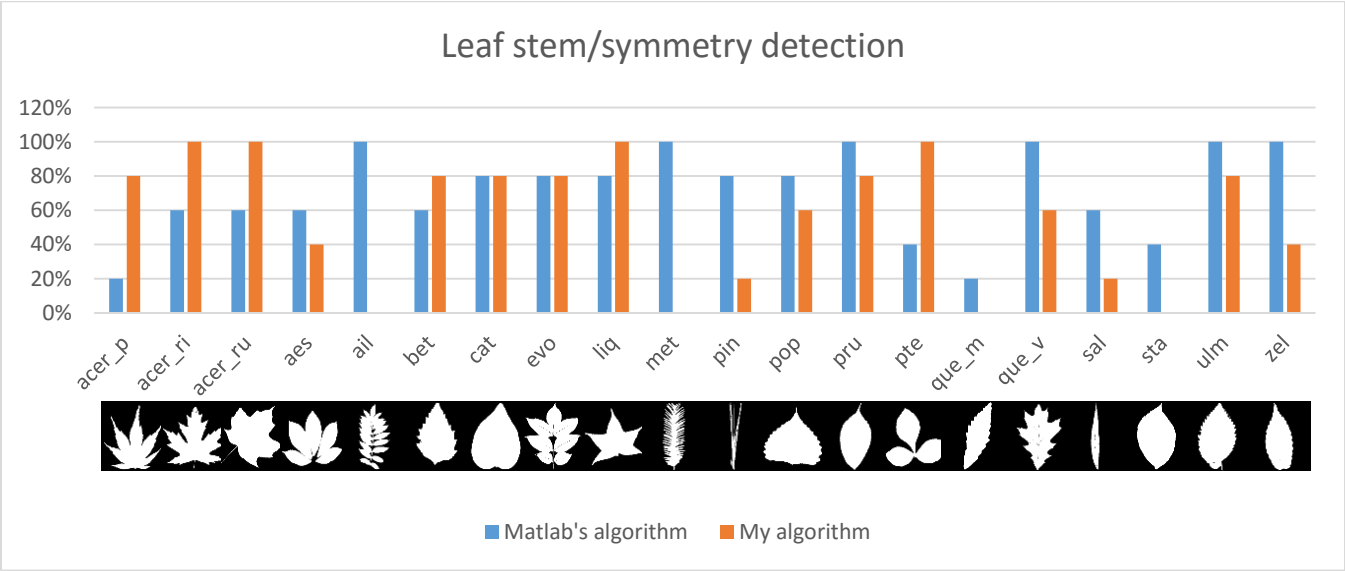


Figure 3.2.11

(3) 3rd model: concave-triangle model (March, 2016)

I designed this model when I got a better understanding about machine learning and data types in Matlab. It goes in the opposite direction to the round-scanning model – rather than measuring distance from the centroid outward, this model measures distance from edges of the convex polygon into the shape. It is based on

calculations on concave triangles (the red triangle in the below chart) and collects **ten features** from each picture.

Steps:

1. Get the convex set of the shape from Matlab's built-in function regionprops(). In the right-hand chart, they are marked and circled in blue.
2. Between any two points in the convex set, walk along shape boundary and measure the distance from boundary point to bottom (the edge connecting the two points), and apply hill climbing algorithm to find all maxima and minima.
3. In the below chart, the boundary points are stretched as a 2D plot. X-axis is the indices in boundary point array; Y-axis is the distance from boundary point to an edge of the convex hull. The region below the blue curve is outside the leaf and above is inside the leaf.

For each concave triangle (major jag), the following properties are calculated:
number of maxima: how many trivial jags it has;
(**triangle height**) / (**average radius of the shape**): how deep the jag is;
(**triangle height**) / (**triangle bottom**): shape of the jag;
(**curve length**) / (**triangle bottom**): how much the boundary is folded;
top angle: formed by the maxima and left/right bottom vertices;
average of two side angles: there are two of these angles in one triangle, where the tops are midways on boundary between the maxima and left/right bottom vertices;

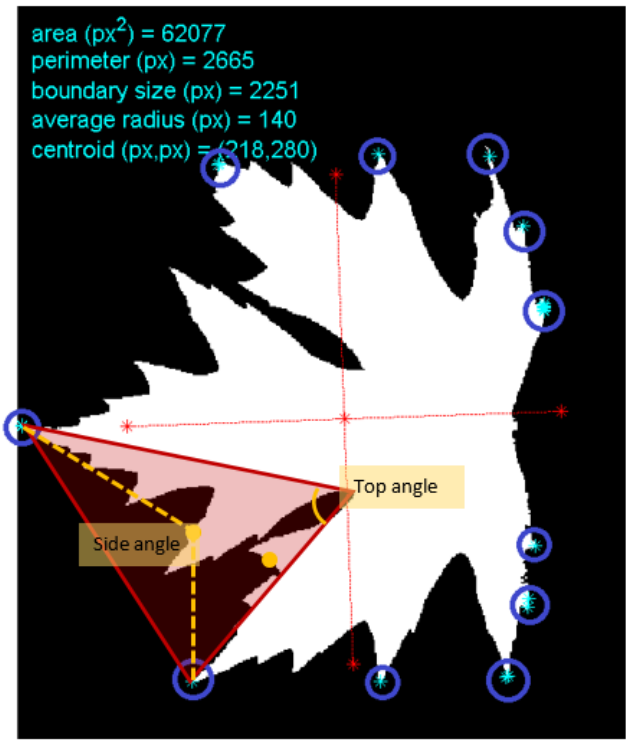


Figure 3.3.1

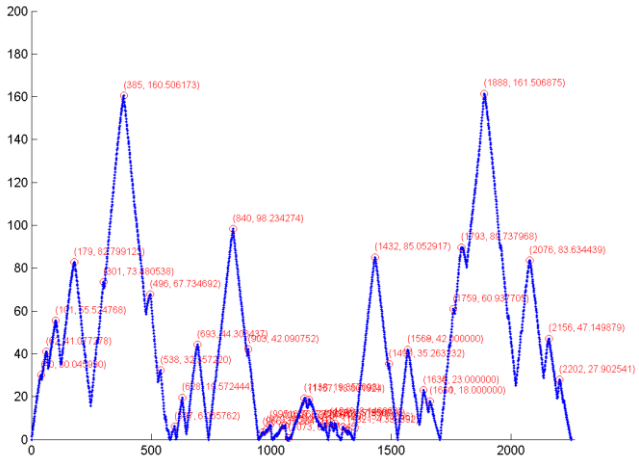


Figure 3.3.2

The following matrix is an example of data collected from the leaf above:
Each row represents a major jag (a concave triangle); columns are properties of a jag.

Index of left bottom vertex	Index of top vertex	Index of right bottom vertex	(triangle height) / (average radius of the shape)	(triangle height) / (triangle bottom)	(curve length) / (triangle bottom)	top angle	average of two side angles
0001	0385	0577	1.1	0.7	3.2	063.6	111.9
0578	0628	0645	0.1	0.4	1.6	076.4	124.2

0645	0693	0738	0.3	0.8	2.0	064.1	088.9
0738	0840	0947	0.7	1.3	3.4	039.9	068.8
0947	0995	1005	0.0	0.1	1.3	099.2	122.3
1005	1057	1067	0.0	0.1	1.3	097.7	158.1
1067	1138	1237	0.1	0.1	1.2	152.8	163.7
1237	1248	1260	0.1	0.5	1.6	111.2	114.7
1260	1265	1285	0.1	0.4	1.6	111.8	149.8
1291	1298	1318	0.0	0.3	1.4	095.7	115.0
1318	1321	1344	0.0	0.2	1.1	140.0	156.5
1345	1432	1526	0.6	1.0	2.5	051.6	081.9
1526	1568	1617	0.3	0.7	1.9	066.7	094.6
1617	1634	1703	0.2	0.3	1.5	081.1	110.8
1703	1888	2250	1.2	0.8	3.2	060.5	094.1

Figure 3.3.3

These features will be collected:

Total number of major jags or number of concave triangles divided by 100	floating points	1 feature	Typical range (0.05, 0.15)
Total number of trivial jags from all concave triangles divided by 100	floating points	1 feature	Typical range (0.15, 0.50)
Average and standard deviation of column 4 in Figure 3.3.3	floating points	2 features	Typical range (0, 1)
Average and standard deviation of column 5 in Figure 3.3.3	floating points	2 features	Typical range (0, 1)
Average and standard deviation of column 6 in Figure 3.3.3	floating points	2 features	Typical range (1, 5)
Average of column 7 in Figure 3.3.3 divided by 100	floating points	1 feature	Typical range (0.6, 0.9)
Average of column 8 in Figure 3.3.3 divided by 100	floating points	1 feature	Typical range (0.9, 1.2)

** Some values are divided by 100 to maintain similar ranges of all 10 features.

Figure 3.3.4

Advantage: combining advantages of the first two models - breaking curvature features into details and having low dependence on image orientation.

Drawback: having poor performance when the leaf has no jags (perfectly smooth); needing more conditions to deal with various shapes; features are of different units and scales (numbers, ratios and angles) which leaves a problem in distance calculation during clustering.

(4) Other attempts

A common defect of the three models is not handling discrete shapes well – some parts of a leaf drops apart and become multiple regions after image segmentation. Matlab's functions would identify these regions with separate centroids and perimeters. I partially solved this problem by calculating centroid according to weight (number of area pixels inside the region) and summing up the perimeters.

Centroid = $(C_1 * W_1 + C_2 * W_2 + \ldots + C_n * W_n) / (W_1 + W_2 + \ldots + W_n)$

However, I still didn't figure out how to connect the boundaries of separate regions. The arrays of boundary cannot be simply concatenated because we'll also need the convex hull and other analysis to be based on a

single region. A method described by a Leafsnap's paper is to fill in the bottlenecks with pixels to achieve a continuous curvature[2].

The Matlab script on features extraction[3] is automated (or at least semi-automated) in order to work with a large sample size. It takes one input folder and three output folders which are defined at the top of the script and which must be created ahead of time.

```
dirpath_in = '160401in/'; % input images, 160401in/<species_folders>
dirpath_outimg = '160401outimg/'; % output images with a lot of markings
dirpath_outtxt = '160401outtxt/'; % output text data of concave-triangle model
dirpath_outplot = '160401outplot/'; % plot of text data
```

After execution, it fills in the output folders with images and text files, and gives three extra output files outside of all folders which is in Weka format. You need to change the extension from .txt to .arff before reading into Weka.

```
output_weka1 = fopen('weka1.txt', 'a'); % round-scanning model
output_weka2 = fopen('weka2.txt', 'a'); % 3-meta-properties model
output_weka3 = fopen('weka3.txt', 'a'); % concave-triangle model
```

4. Clustering

Weka has built in a number of clustering algorithms, such as expectation-maximum (EM), K-means, farthest first. I don't see much improvement when switching to another algorithm, so I stick with K-means algorithm to make it a controlled condition.

In the below chart (figure 4), each row stands for a model; each column stands for a species; and each color stands for a cluster. I re-shaded the clusters so we can get a vertical comparison among the models. The round-scanning model has the worst performance with each species spreading to many clusters and each cluster covering many species. Wrong axis detection and amplifying the noise are the main problems of this model. On the contrary, the 3-meta-properties model does better with only 3 features.

This chart was obtained from collecting features from 300 images across 20 species (15 images for each), dividing into 3 groups (7 + 7 + 6 species), and running K-means clustering algorithm 9 times (3 groups * 3 models)[4]. **The run time of collecting features from 300 images is about 2 hours.** If commenting out the round-scanning model in the script it takes only a half time (1 hour). The reason for dividing them into 3 groups for clustering is that **putting 300 data into >= 10 clusters** takes too long (**longer than 2 hours** when tried with several algorithms in Weka on both my laptops), whereas **putting them into 7 clusters takes less than 5 minutes** for each iteration of clustering. I assume that the algorithms take exponential run time upon the number of clusters.

[2] Leafsnap: A Computer Vision System for Automatic Plant Species Identification
[3] 160401ProcessImage.m
[4] 160401output.xlsx has the numerical data of this chart

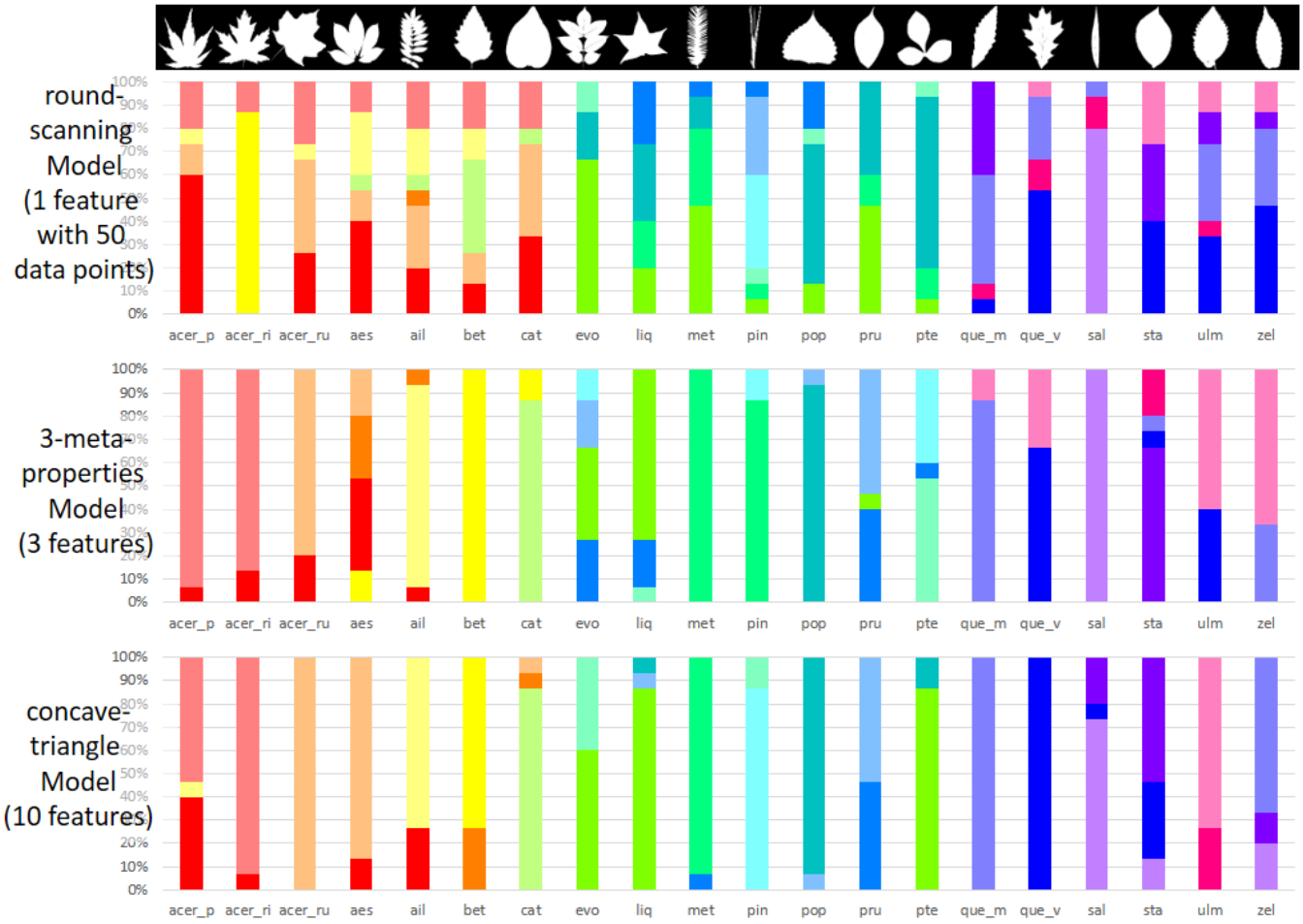


Figure 4

5. Discussion for future study

This CMPT400 project provides a good starting point for me to get into image processing and machine learning. Both topics are important joints between computer science and other disciplines. Topics for future study based on leaf shape recognition is plenty.

For sampling, I can choose a bigger dataset with some very similar species, and see how it drags down the performance.

For extracting, finding the symmetry is not an appropriate approach, since some species are naturally asymmetrical (e.g. elm leaves); for symmetrical leaves, the symmetry can be skewed by camera's perspective and dropping parts due to segmentation. However, determining leaf's orientation remains as an important issue to solve because it can add considerable accuracy to any kind of features extraction model. It can be separated from finding the symmetry (e.g. determining where the stem comes in) by analyzing the pattern of convex set.

There are more approaches to try apart from determining orientation:

(a) Connecting discrete shapes by filling in bottleneck regions (as discussed in the previous section);
(b) Constructing a new features extraction model as illustrated in Leafsnap's paper. It draws a circle of radius R centering at a boundary point, measures the arc length and number of pixels inside/outside the shape. For the next iteration, it grows the radius of the circle and collects the same type of data. Finally, it collects data from different radius of circles into a histogram for species association.

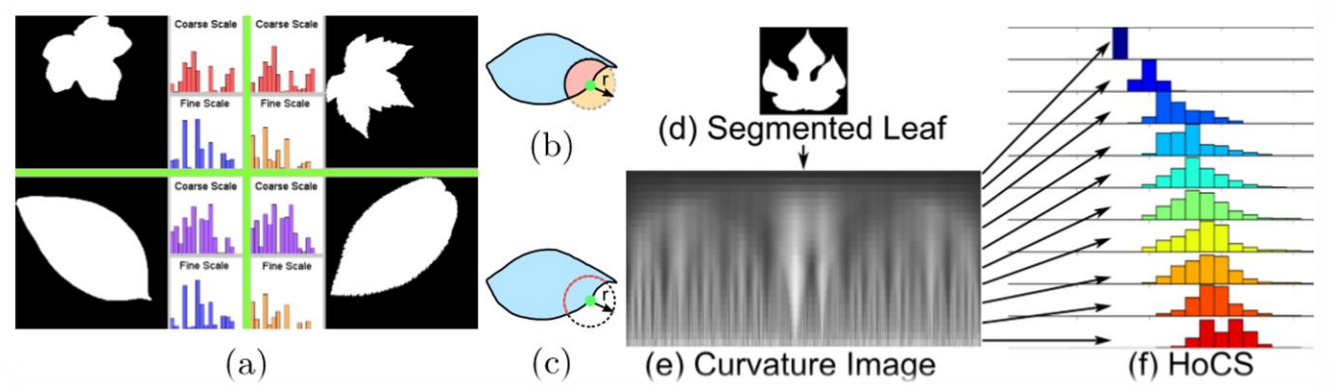


Figure 5

From the machine learning perspective, I can add in a different flavor by doing supervised machine learning (classifying instead of clustering), which requires to train a model containing a number of species, and test with more images to see how many of them are classified correctly.

References

Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida Lopez, and Joao V. B. Soares. Leafsnap: A Computer Vision System for Automatic Plant Species Identification. 2012.

Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach (Third Edition). 2010.

Zhaozhong Wanga,n, Lianrui Fua, Y.F. Lib. Unified detection of skewed rotation, reflection and translation symmetries from affine invariant contour features. Pattern Recognition 47 (2014) 1764-1776.

Tat-Jen Cham and Roberto Cipolla. Symmetry detection through local skewed symmetries. Image and Vision Computing, 1995, Vol.13(5), pp.439-450.