University of the Philippines Manila

College of Arts and Sciences

Department of Physical Sciences and Mathematics

LeafSnap PH: A Mobile Application for Identifying Leaf Samples of Philippine Plants

A special problem in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Computer Science

Submitted by:

Marbert John C. Marasigan

May 2016

Permission is given for the following people to have access to this SP:

Available to the general public	Yes
Available only after consultation with author/SP adviser	No
Available only to those bound by confidentiality agreement	No

ACCEPTANCE SHEET

The Special Problem entitled "LeafSnap PH: A Mobile Application for Identifying Leaf Samples of Philippine Plants" prepared and submitted by Marbert John C. Marasigan in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science has been examined and is recommended for acceptance.

Marvin John C. Ignacio, M.Sc.(*candidate*)
Adviser

EXAMINERS:

	Approved	**Disapproved**
1. Gregorio B. Baes, Ph.D. (*candidate*)		
2. Avegail D. Carpio, M.Sc.		
3. Richard Bryann L. Chua, Ph.D.(*candidate*)		
4. Perlita E. Gasmen, M.Sc. (*candidate*)		
5. Ma. Sheila A. Magboo, M.Sc.		
6. Vincent Peter C. Magboo, M.D., M.Sc.		

Accepted and approved as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science.

Ma. Sheila A. Magboo, M.Sc.
Unit Head
Mathematical and Computing Sciences Unit
Department of Physical Sciences
and Mathematics

Marcelina B. Lirazan, Ph.D.
Chair
Department of Physical Sciences
and Mathematics

Leonardo R. Estacio Jr., Ph.D.
Dean
College of Arts and Sciences

Abstract

The task of identifying plants is time-consuming even for botanists. LeafSnap PH is a mobile application that will aid users in identifying plant species through their leaves. The system will allow the user to search the database, browse the list of collected leaf samples, and take pictures of leaves and analyze them. The leaf feature that will be extracted is its shape using contour points, and this will be used in identifying the leaf using support vector machine.

Keywords: leaf recognition, mobile application, support vector machine

Contents

List of Figures

I. Introduction

A. Background of the Study

One of the most time consuming tasks a botanist faces is the identification of a plant. They would usually use the leaf, flower, stem, fruit, etc. to extract discriminating features[1] . A key is "a device in which successive choices between contrasting statements are followed until the correct name is found by the process of elimination." A modern key is constructed in a dichotomous fashion, which means that a statement can only be either true or false[2]. This would require a long list given the approximate number of 400000 known plant species around the world[3].

Currently, there has been a lot of work on automating plant recognition [4, 5, 6, 7]. Wang *et al.* used the centroid-contour distance curve in finding the leaf shape[4]. Wu *et al.* extracted the leaf shape (using slimness, roundness, solidity, and moment invariants), leaf dent, and leaf vein and used a feed-forward back-propagation neural network to identify a leaf[5]. Du *et al.* uses the Douglas-Peucker approximation to get the polygonal approximation of the leaf and modified dynamic programming for shape matching[6]. Kumar *et al.* used histograms of curvature to obtain the shape of the leaf and used the Nearest Neighbors Algorithm to identify a list of candidates for the leaf[7]. The first three were implemented in a PC [4, 5, 6]. Only Kumar *et al.* was able to develop a mobile application for leaf identification called LeafSnap[7].

In iOS, Leafsnap is an application published in the App Store for users to be able to identify plant species using images of leaf samples[8, 9, 10]. A picture of the leaf is taken by a user via the smartphone's camera and the image is uploaded to a server, and the server responds with a list of possible plants based on the leaf's shape. In the server, a Support Vector Machine (SVM) is used to classify an image as leaf or non-leaf. Once the image has been identified as a leaf, the image will undergo initial segmentation using Expectation-Maximization to separate the foreground and the background. False-positive regions are removed and the leaf stem is also removed. Histograms of curvature are computed and species identification is then done by the nearest neighbors[7].

The main feature of LeafSnap is plant identification by taking a picture of a leaf using a smartphone's camera. The application can also handle textual searches of the common or scientific name of a leaf in its database. LeafSnap can also locate nearby plants in a map by a database, where users send information (geographical location, name of plant identified, etc.) to the server. The application also has a game where you match a certain leaf to its common or scientific name. Currently, the developers of Leafsnap are working on an Android version of the app, yet there is still no news on its release[7, 11]. Although, there are theses on an Android app similar to Leafsnap, but none of it has been released in the Google Play Store[12, 13]. There are two datasets of Leafsnap: one covers the trees of the Northeast United States[11], which is found in the original Leafsnap app[8, 9], the other covers the trees found in the United Kingdom[11], which is found in the Leafsnap UK app[10]. Currently, there is no local version of Leafsnap[11].

According to a forecast, by the year 2015, 65.8% of the world will have own a smartphone[14]. Of the 2.03 billion smartphone users[14], 65% of them have an Android-powered smartphone[15]. In the Philippines, Android-powered smartphone sales are at 91%[16]. This makes Android a good platform to develop apps, if app accessibility, whether in a local or global scale, is considered here.

The Philippine Plant Information System (PPIS) contains information on the plant collections and specimens from the different herbaria in the Philippines collected, aggregated and publicly shared. The PPIS allows users to search, store and share

Philippine plant information. The PPIS uses the Darwin Core standard to store the information.

The mobile application is part of a project along with the Philippine Plant Information System.

B. Statement of the Problem

Botanists, when identifying plant species, are always on the field and thus mobility is needed. Smartphones can provide the mobility botanists need, but not all botanists have an iPhone or iPad. With the absence of an Android-based, leaf recognition mobile application published in the market, there is a need to bring leaf recognition to Android users. The challenge is to provide that application with as much tools as needed to botanists given the limitations of a smartphone. There is also an absence of a mobile application for identifying Philippine plants or even a mobile database for Philippine plants.

C. Objectives of the Study

1. Create a mobile application that allow users to:
    - (a) Take a picture of a leaf. The image is stored in the device's file storage, then:
        - i. If the device has an internet connection and the user chooses to proceed with identification, the image will be sent to the server. The server will extract features, and use the extracted features to identify the leaf with a trained model, and then the server will return the top five (5) predictions. The user will then choose which label is the closest, and the leaf's common name and scientific name will be stored to the device's internal database.

        ii. If the device has no internet connection or the user chooses not to proceed with identification, the image is still stored in the file storage and can be identified at a later time.

    (b) Re-identify a leaf that has been previously identified.

    (c) Browse the gallery of images stored in the device.

    (d) Do a textual search of plants by their common or scientific name in the database.

2. Create a server that allows the AI expert to:

    (a) Extract features from images of JPEG format.

    (b) Train the support vector machine (SVM) using LIBSVM.

D. Significance of the Project

The system developed in Android would be able to provide a useful tool for botanists in the field to identify leaf samples. Even without a broad knowledge of plants, this would make the identification process less time-consuming.

E. Scope and Limitations

1. The project's database would only include local plants.

2. Leaf identification would only be limited to leaf samples in the database.

3. Leaf samples and those for identification are simple leaves.

4. Accuracy would depend on the trained model.

F. Assumptions

1. The user doesn't take a picture of any other part of the tree, only the leaf.

2. The leaf sample is placed on a white background.
3. The leaf sample is not tampered, torn, crumpled or distorted in any way.
4. The leaf sample is in its mature stage.
5. The leaf's orientation is upright with respect to the portrait orientation of the smartphone.
6. The leaf no longer contains the leaf stem.
7. The leaf image is bright (i.e. taken with a well-lit background), but should not be taken with the device's camera flash.

II. Review of Related Literature

Leafsnap is a mobile application for the identification of plant species through leaf recognition. It is currently deployed in iOS-run devices and an Android version of the application is still in the process of development. Currently, its dataset includes all 184 tree species of the Northeastern United States. The system is composed of a backend server, where front-end clients would send input images accepted by the server for recognition. The front-end application, a mobile application, provides clients with browsing, textual searching, and sorting functions. Images captured by clients through the mobile phones camera are sent to the server for recognition and a response with the most likely candidates are sent to the phone for the user to identify the captured leaf image[7].

In the LeafSnap application, the image is first classified as a leaf or non-leaf using a support vector machine. If an image is recognized as a leaf, the process will proceed. If, however, the image is recognized as a non-leaf, the application will inform the user that it has captured either something that is not a leaf or a leaf but doesn't meet the standards (a leaf placed on a white background with no clutter) and the user will be asked to take another picture. After the classification, the leaf image will undergo preprocessing, where the original leaf image will be converted to the hue-saturation-value color space. However, hue is not useful because the background would often have greenish tinge, therefore, only the saturation and value color space will be used in segmentation. Expectation-Maximization will be used for the initial segmentation. Next, false positive regions and the leaf stem is removed. This comprises the segmentation part. The feature that is to be extracted is the leaf shape. First, curvature values are computed using either the area measure or arclength measure. Using these measures, the contour is extracted and the curvature image is produced. Histograms are then taken at each row of the curvature image and this forms the Histograms of

Curvature over Scale (HoCS) feature.

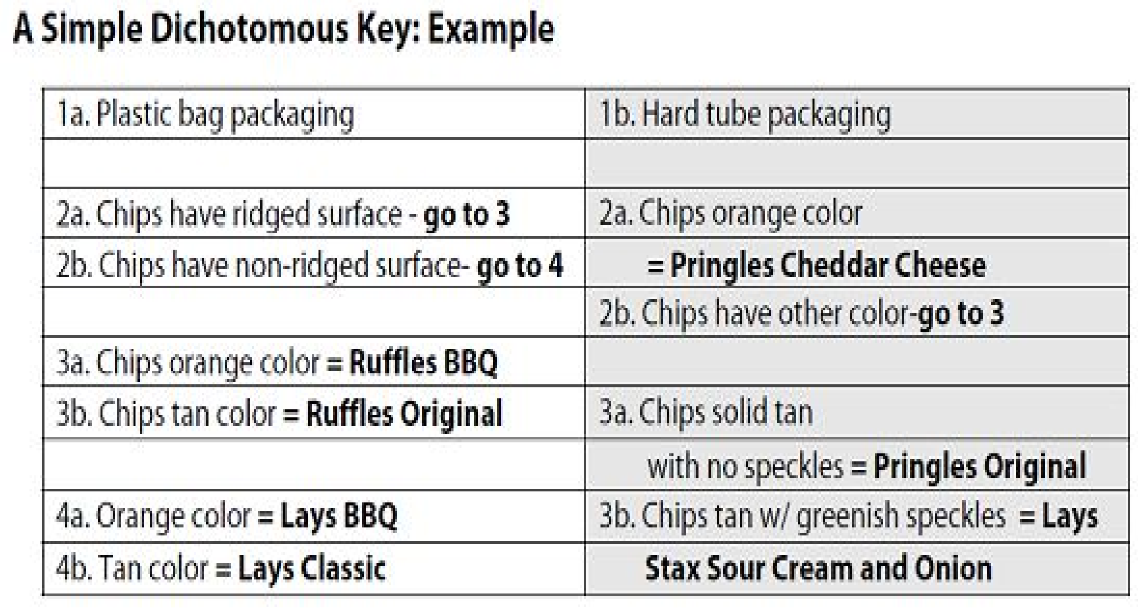
A lot of work has been done on leaf recognition and shape extraction. Munisami et al.[17] uses shape features derived from the aspect ratio, white area ratio, perimeter to area, perimeter to hull, hull area ratio, distance map x, distance map y, and centroid radial distrance measurements. These ratios are then used to compute for the Euclidean distances with those of the database. The 3 closest results are then returned using the k-nearest neighbors classifier. Satti et al.[18] extracted three types of shape features namely, geometric features, morphological features, and tooth features. The color feature and shape features are then used in identifying a leaf. The study found that Artificial Neural Networks are more accurate, with an accuracy of about 93%, than the k-Nearest Neighbors algorithm. Chaki and Parekh[19] compared the Moments-Invariant (MI) model and Centroid-Radii (CR) model of extracting leaf shape features. It is found that combining the MI and CR features would yield a higher accuracy. Larese[20] et al. used solely leaf venation to identify leaves of three legumes. In getting the veins, the researchers used unconstrained hit-or-miss transform to be able to extract the vein pattern. It is reported to have a higher accuracy compared to manual classification. Bai et al.[21] decomposed shapes into contour fragments and aggregated them ”to a compact vector of limited dimension to stand for a shape” called Vector of Aggregated Contour Fragments (VACF). It is found that using VACF yields higher accuracy compared to skeleton paths, class segment set, and contour segments, which aren also shape-classification algorithms. It is reported that VACF saves time and memory and useful for mobile development. Ehsanirad[22] used texture features to identify a leaf. The researcher compared the gray level co-occurence matrices (GLCM) algorithm and the principal component analysis (PCA) algorithm. The GLCM is a popular texture analysis algorithm. The PCA is based on the eigenspace approach. The GLCM yielded a 79% accuracy, while the PCA had about 98%.

The support vector machine (SVM) was first proposed by Vapnik[23]. It is " a computer algorithm that learns by example to assign labels to objects." [24] In the paper of Srivastava and Bhambhu, it is reported that SVM (with the Radial Basis Function or RBF kernel) gives more accurate results than other classifiers used[23]. The training set of the data are projected in a multiple or infinite dimensional space and the SVM finds a linear separating hyperplane with the maximal margin[25]. Classification is done by projecting the test data to the multiple or infinite dimensional space and checked in which side of the space it is in.

III. Theoretical Framework

A. Leaf Recognition

Plant recognition is done by many ways. The easiest way is to ask an expert who already knows. Another way is to consult picture books containing images of plants and its different parts like the leaf, twig, flower, etc. However, this is time-consuming, since one has to cross-examine the leaf to the images one-by-one. Another way of identifying plants is by using keys. Keys are devices where successive choices between contrasting statements are followed until the correct name of the plant is found. This is done by the process of elimination. Modern keys are dichotomous, meaning, each statement only has two choices. An example of a dichotomous key is found in Figure 1[26].



A Simple Dichotomous Key: Example

1a. Plastic bag packaging	1b. Hard tube packaging
2a. Chips have ridged surface - **go to 3**	2a. Chips orange color
2b. Chips have non-ridged surface- **go to 4**	**= Pringles Cheddar Cheese**
	2b. Chips have other color-**go to 3**
3a. Chips orange color **= Ruffles BBQ**	
3b. Chips tan color **= Ruffles Original**	3a. Chips solid tan
	with no speckles **= Pringles Original**
4a. Orange color **= Lays BBQ**	3b. Chips tan w/ greenish speckles **= Lays**
4b. Tan color **= Lays Classic**	**Stax Sour Cream and Onion**

Figure 1: A Simple Dichotomous Key

In identifying leaves, keys can also be used. Leaves can be classified in may ways[27]. By leaf arrangement on stems, leaves can be classified into four types as shown in

Figure 2[27]: Alternate, Opposite, Whorled, and Rosette.

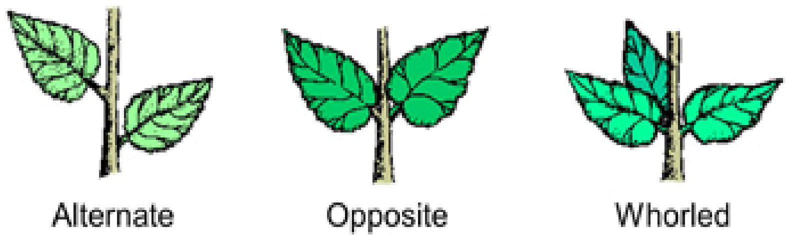


Figure 2: Types of Leaf Arrangement on Stem

A leaf can also be classified by leaflet arrangement on petiole as shown in Figure 3[27]. These are, simple or compound. Under compound, a leaf can be pinnately compound, palmately compound, or doubly compound.

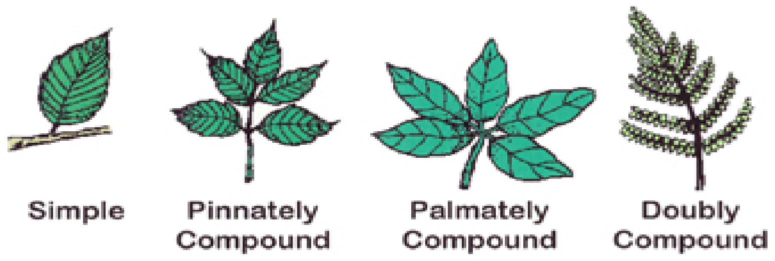


Figure 3: Types of Leaf Arrangement on Stem

A leaf can also be classified by its shape and it is the primary tool in plant identification. Common shapes are oval, lanceolate, elliptical, spatulate, cordate, oblanceolate, obcordate, oblong, linear, peltate, cuneate, reniform, and hastate as shown in Figure 4[27].

Leaf apex (or the tip) and base can also be used in identification. Leaf apex shapes can come in acuminate, acute, cuspidate, emarginate, mucronate, obcordate, obtuse, or truncate as shown in Figure 5[27]. Leaf base shapes can be actue, auriculate, cordate, hastate, oblique, rounded, sagittate, or tumcate as shown in Figure 6[27].

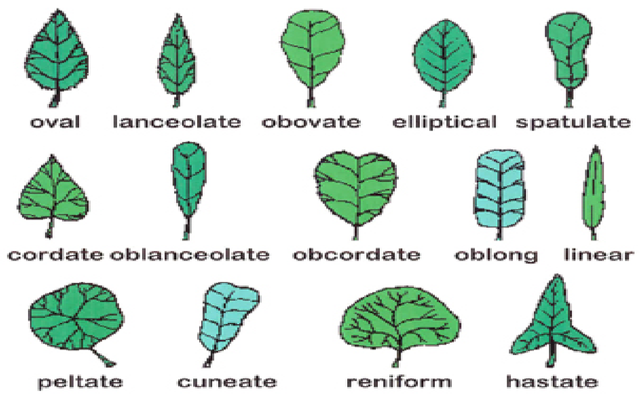


Figure 4: Types of Leaf Shapes

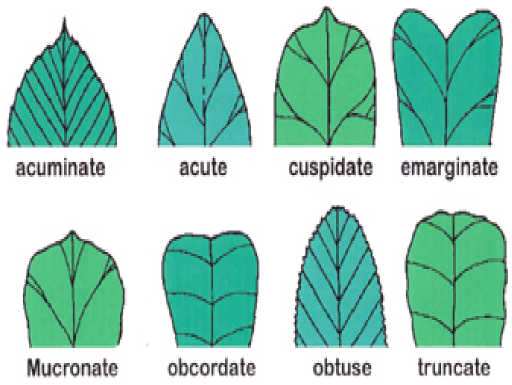


Figure 5: Leaf Apex Shapes

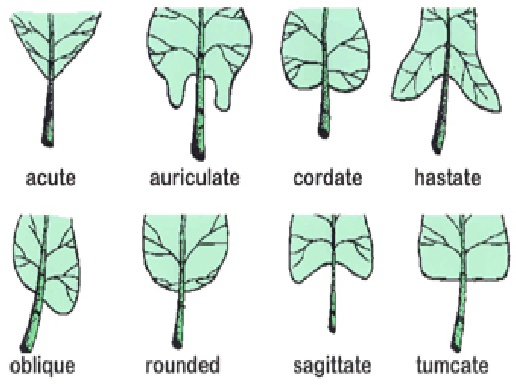


Figure 6: Leaf Base Shapes

Another tool in leaf identification is its margin. Shapes of leaf margins can be crenate, incised, sinuate, undulate, lobed, entire, serrate, serrulate, double serrate, or dentate as shown in Figure 7[27].

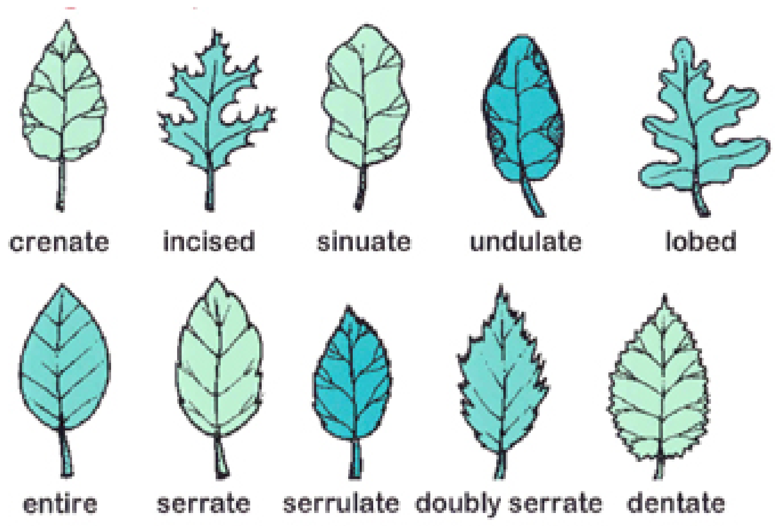


Figure 7: Leaf Margin Shapes

Leaves can also be classified into four types: conifer, ginkgo, monocot, and dicot. These four types can be classified further by their leaf venation.

Conifer types can be classified into four types based on their venation namely, scale-like, awl-shaped, linear-shaped, and needle-like as shown in Figure 8[27]. Needle-like leaves can be single, bundled, or clustered.

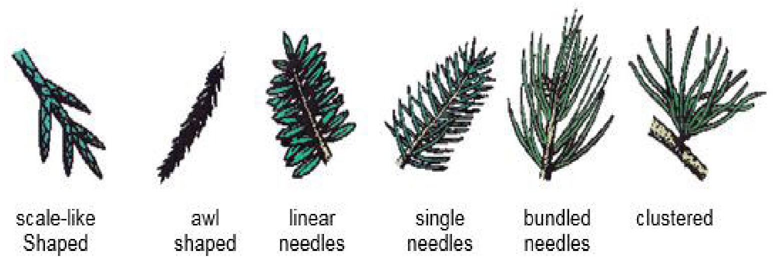


Figure 8: Conifer Type Leaf Venations

The ginkgo type has a dichotomous venation shown in Figure 9[27]. These are found in Ginkgo trees.

Monocot type plants like grasses, tulips, etc. have a parallel venation as shown in Figure 10[27], while Dicot type plants are net-veined. Net-veined plants can either

Figure 9: Ginkgo Leaf

have a pinnate venation or a palmate vennation as shown in Figure 11[27].

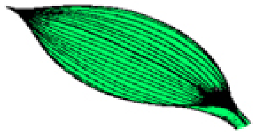


Figure 10: Parallel Venation in Monocot Leaf

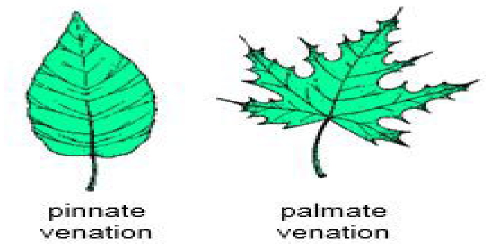


Figure 11: Venation of Dicot Leaves

B. Image Classification

According to [28], "Classification includes a broad range of decision-theoretic approaches to the identification of images (or parts thereof). All classification algorithms are based on the assumption that the image in question depicts one or more features and that each of these features belongs to one of several distinct and exclusive classes." In image classification, the numerical properties of image features are analyzed and these are classified into different categories. Classification algorigthms usually have two steps: training and testing. In the training step, the characteristic features of images are extracted, and based on these, a training class (which contains unique descriptions of each category) is created. In the test step, the training class is used to classify image features. Training classes should contain these three qualities:

1. Independence: A change in the description of a training class should not change the value of another
2. Discriminability: Different image features should result in different descriptions
3. Reliability: All image features within a training group must share the common definitive characteristics of that group

C. Computer Vision

"Computer vision is the science of endowing computers or other machines with vision, or the ability to see." Computer vision has two goals: to make things work (engineering), and to understand intelligence. In the engineering approach, computer vision is used to solve real-world problems. Examples are leaf recognition and facial recognition. Some are also in to understanding human intelligence by computer vision. Researchers are trying to build something that works like the human brain in order to understand how human brains store memory, learn, etc. Computer vision has applications in Optics, Photography and Photogrammetry, Computer Graphics and Art, Neuroscience and Physiology, psychology and Psychopysics, and Probability, Statistics and Machine Learning. [29]

1. Expectation-Maximization Algorithm

The Expectation-Maximization (EM) Algorithm is an iterative procedure to get the Maximum Likelihood Estimate (MLE) with hidden or missing data. The first step is the E-step or the expectation step. The missing data are estimated given the observed data and current estimate of the modeled parameters. The second step is the M-step or the maximization step. In this step, the likelihood function is maximized which assumes that the missing data are known. The likelihood is increased at each iteration and thus, convergence is assured[30]. LeafSnap uses the saturation and value fields in the HSV space for Expectation-Maximization [7]. Here, the EM algorithm is used to separate the leaf from the background. The probabitlity distribution of a pixel is modeled by the sum of two Gaussians. The Gaussians have a mean of μ_k and a shared covariance of Σ. Each gaussian are given weights of $\frac{1}{2}$. Basically, what the EM algorithm here does is to separate the leaf from the background. Figure 12[7] shows a leaf image in RGB (a), then it is converted to the saturation-value space (b), and then it is segmented using the EM algorithm in (c).

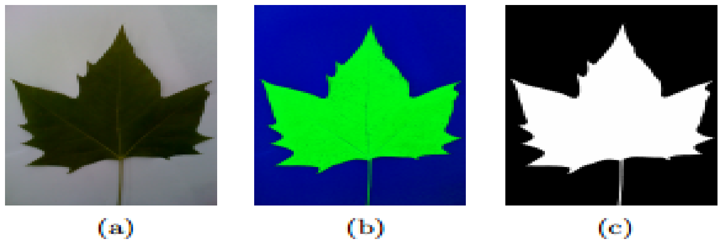


Figure 12: Expectation Maximization on a Leaf Sample Over The Saturation-Value Space

2. Histograms of Curvature over Scale

Functions of the curvature at a boundary point are computed using integral measures. "One measure in 2D is the area of intersection of a disk centered at a contour point and the inside of the contour". Another is the arclength, which is the fraction of the disk's perimeter inside the contour. The histograms of curvature values at each scale are computed and then concatenated to form the Histograms of Curvature over Scale (HoCS)[7]. In obtaining the leaf shape, we first get its curvature image. Each row in the curvature image represent the area/arclength (whichever approach) over a radius r. Radius increases as the row number goes up. Each column in the curvature image represent a contour point. To obtain the HoCS, we calculate a histogram at each row of the histogram. These histograms form the HoCS, where the scale is the radius of the circle.

D. Machine Learning

Machine Learning is a broad field and is quickly expanding, creating more subfields and types of Machine Learning[31]. According to [32], “Machine Learning has become one of the mainstays of information technology and with that, a rather central, albeit usually hidden, part of our life. With the ever increasing amounts of data becoming available there is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress.” One of the recent definitions of Machine Learning was given by Tom Mitchell in 1997: “A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.[31] Machine Learning algorithms consist these three things: a set of models to look through, a way to test the model, and a way to find a good model by undergoing only a few tests[33]. There are three types of Machine Learning algorithms[34]:

1. Supervised Learning: This algorithm consist of dependent variables (or outcomes) which are predicted from a given set of independent variables (or predictors). From these set of variables, we want to create a function that will map the inputs to the desired output. This training continues until it reaches a certain level of accuracy. Examples of supervised learning are Regression, K-Nearest Neighbors, etc.

2. Unsupervised Learning: This algorithm does not have a set of variables to predict, instead the population is clustered or grouped. Examples are K-Means and the Apriori Algorithm.

3. Reinforcement Learning: In this algorithm, the machine is exposed to an environment where it trains itself continually by trial-and-error. The machine learns through past experience and gather the best knowledge, then tries to make the

best decisions. An example would be the Markov Decision Process.

1. Support Vector Machine

A support vector machine (SVM) is a supervised learning technique applicable in classification and regression [35]. Linear SVM is a binary classifier.

According to [36], Linear SVM aims to find the optimal separation between two classes by using a hyperplane. However, not all datasets can be linearly divided like that in Figure 13[36].

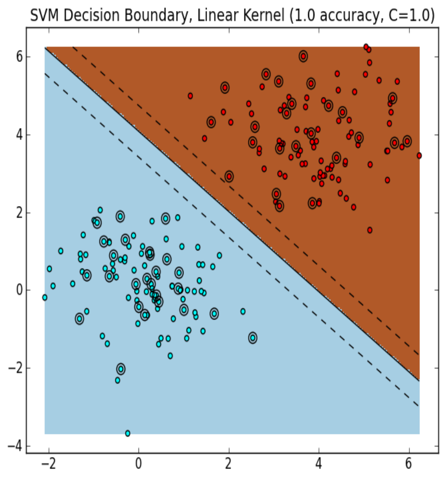


Figure 13: A Linearly Separable Dataset

Figure 14[36] shows an example of a dataset that is not linearly separable.

Ideally, an ellipse would be the best to separate the two classes. However, SVM only generates a linearly separating hyperplane. Therefore, in $\mathbb{R}^2$, it is impossible to "perfectly" fit a linear function that would separate the two classes. However, when the graph in Figure 14[36] is projected in $\mathbb{R}^3$, the data becomes linearly separable

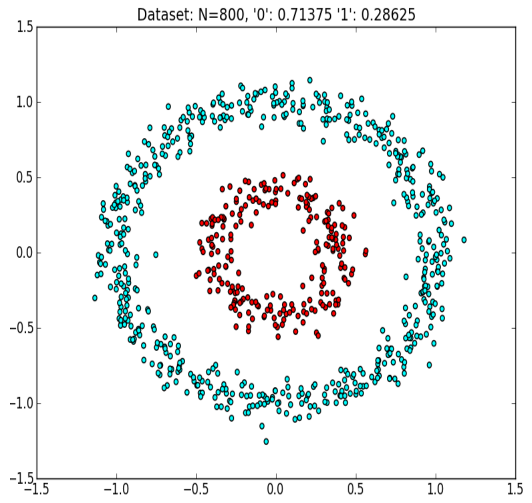


Figure 14: A Linearly Nonseparable Dataset

by a hyperplane. But, the challenge lies in finding a transformation $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. Assuming that the transformation ϕ was found, we transform the training set, say X with ϕ, giving us X'. We then train a linear SVM on the transformed training set X' to get the classifier f_{svm}. In testing, we transform x to $x' = \phi(x)$ and the output label is determined by $f_{svm}(x')$. If we then project back to $\mathbb{R}^2$, we will obtain the decision boundary that is nonlinear as shown in Figure 15[36].

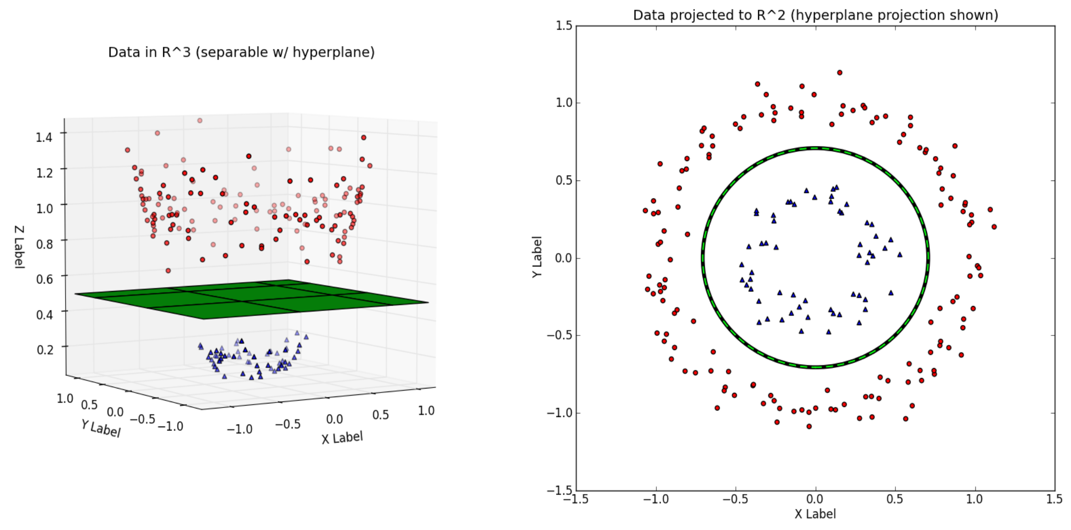


Figure 15: The Data in $\mathbb{R}^3$ With a Linear Separating Hyperplane (Left) and the Data Transformed Back to $\mathbb{R}^2$ Along With the Hyperplane.

Therefore, a dataset D that is linearly nonseparable in $\mathbb{R}^N$ may be linearly separable in $\mathbb{R}^M$ (where $M > N$) as long as a transformation ϕ is found. The problem is if $M \gg N$ (M grows very quickly with respect to N, e.g. $M \in O(2^N)$), then learning SVMs by data transformation would incur very large time complexities and space complexities. However, during training, SVM does not need to work in high-dimensional spaces. The optimization problem only uses the training examples to compute pair-wise dot products. There exist functions that, when given two vectors $\vec{v}$ and $\vec{w}$ in $\mathbb{R}^N$, implicitly computes the dot product of $\vec{v}$ and $\vec{w}$ in $\mathbb{R}^M$ without explicitly transforming the vectors to $\mathbb{R}^M$. These functions are called kernel functions $K(\vec{v}, \vec{w})$. This means, we do not need more memory to a higher dimension $\mathbb{R}^M$ and only a little addition to the computation time (which is the time used to compute the dot products). Since, SVM training uses dot products, we can replace these dot products with the kernel function $K(\vec{v}, \vec{w})$ to obtain nonlinear decision boundaries. This is called the "kernel trick." Most classifiers, would offer three popular kernels: the polynomial, radial basis function (RBF), and sigmoid kernels. The following are the mathematical expressions for each kernel:

1. Polynomial Kernel: $(\gamma \cdot \langle \vec{x_i}, \vec{x_j} \rangle + r)^d$

2. Radial Basis Function (RBF) Kernel: $e^{(-\gamma \cdot |\vec{x_i} - \vec{x_j}|^2)}$, where $\gamma > 0$

3. Sigmoid Kernel: $\tanh(\langle x_i, x_j \rangle + r)$

The C parameter, for all kernels, is the parameter that balances between misclassification and simplicity of the decision surface/hyperplane. A low C value makes the decision surface smoother, while a high C value allows more samples to become support vectors, or essentially trying to classify all training examples correctly[37]. The degree parameter, d of the polynomial kernel determines the flexibility of the classifier. A degree of 1 is the linear kernel. The degree parameter is similar to the γ

parameter of the RBF kernel. Both determine the flexibility of the resulting classifier in fitting the data. If γ, for the RBF kernel, or d, for the polynomial kernel grows too large, overfitting will occur[38].

E. Mobile Application

Mobile applications are "end-user software applications that are designed for a mobile device operating system and which extend that devices capabilities." [39] Mobile applications started off as small arcade games, ring tone editors, calculators, calendars, etc. back in the twentieth century[40]. Mobile applications nowadays include mobile payment, mobile navigation, mobile games, etc. [41] Among the mobile operating systems, the most popular are iOS, Android, and Windows Phone 8[42].

1. Android Software Development Kit

The Android Software Development Kit (SDK) includes required libraries, a debugger, an emulator, relevant documentation for the Android application programming interfaces (APIs), sample source codes, and tutorials for the Android OS to develop applications for the Android platform. Each time Google releases a new version of Android, the corresponding SDK will also be released. Android applications are written in Java, and therefore a Java Development Kit (JDK) would also be required[43].

sectionAppcelerator Titanium Appcelerator Titanium is an open, extensible development environment that allows developers to develop cross-platform mobile applications using HTML5, CSS and Javascript. The operating systems supported are iOS, Android and Windows Phone. The Appcelerator Titanium includes a Software Development Kit (SDK), Studio, an Eclipse-based IDE, Alloy, an MVC framework, and Cloud Services.[44]

IV. Design and Implementation

A. Use Case Diagram

The system has three users: the public user, the server, and the Philippine Plant Information System as shown in the Use Case Diagram in Figure 16.

The public user can download the database of leaves, uploaded by the Philippine Plant Information System, and make a textual search of the database.

The public user can also capture a leaf's image using the device's camera and upload them for the server to analyze. The server will extract the features of the given leaf, and feed these features to the support vector machine with the trained model, and then throw a response back to the user identifying the leaf.

The public user can also view the gallery of leaves he has captured with the device's camera, and check also their respective details.

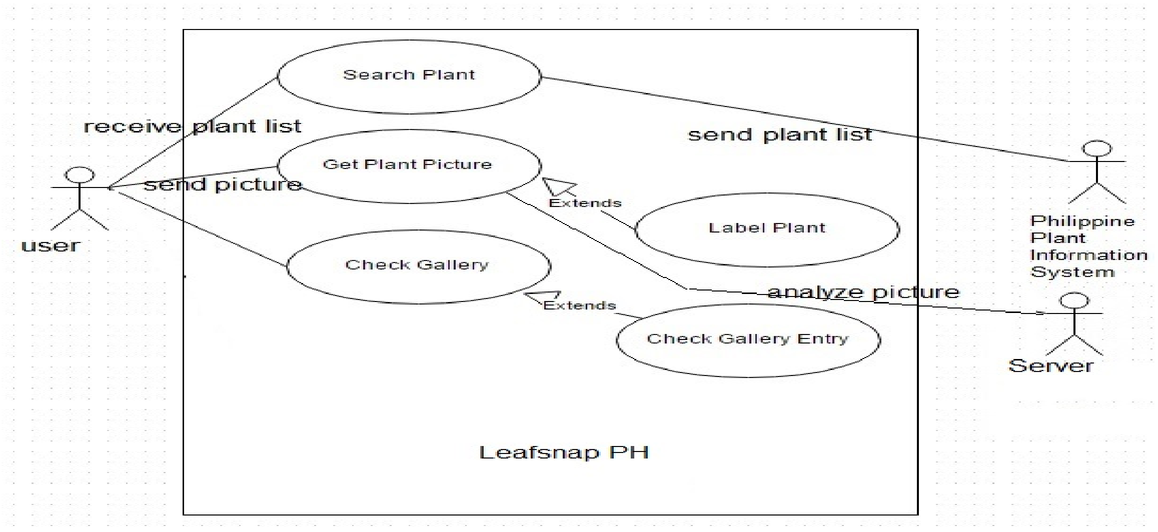


Figure 16: Use Case Diagram

B. Entity-Relationship Diagram

There is only one table for LeafSnap PH, as shown in Figure 17,and that is the leaves. This includes the id, sciName, commonName, longitude, latitude, day, month, year, and filename. id is the device-unique identifier given to a captured leaf sample. sciName is the scientific name of the leaf sample, while commonName is the common name of the leaf sample. latitude and longitude are the GPS coordinates of the leaf sample. day, month, and year stores the date in which the sample was taken. filename is the path where the leaf is stored inside the device.

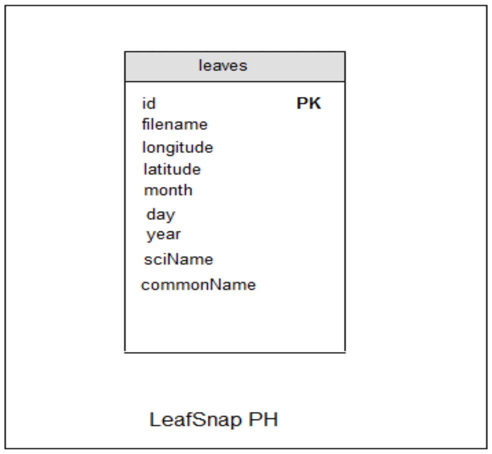


Figure 17: Entity-Relationship Diagram

C. Data Dictionary

D. Activity Flow Diagram

The process of capturing an image sample of a leaf to receiving a response from the server is shown in Figure 18. First, the user captures an image of the leaf sample. The mobile application will then save the image in the file system storage of the device. If the device doesn't have an internet connection, the process ends. However, if the device has an internet connection, the mobile app will then prompt the user if the

Field	Type	Description
id	integer	Unique identifier of the leaf image
filename	text	Filename where the leaf image is stored in the device
longitude	real	Longitudinal location where the image was captured
latitude	real	Latitudinal location where the image was captured
month	integer	Month when the image was taken
day	integer	Date when the image was taken
year	integer	Year when the image was taken
sciName	text	Scientific name of the leaf displayed in the image
commonName	text	Common name of the leaf displayed in the image

Table 1: leaves table

user wants to send the image now for identification. If the user chooses not to, the process ends. If the user chooses to send it now, the device would send the image along with its device ID (a unique identifier for Android devices) to the server. The server would then check if a directory exists for that device ID. If yes, the server would proceed to the identification process. If not, the server will create a directory unique to that device ID then proceed to the identification proces. The server starts the identification process by saving the image in the directory unique to the device ID. It will then extract the image's features, and then using those features, will identify then output the labels of the five closest matches. The server will proceed to get the scientific names and common names mapped to those labels. And then, the server will encode the scientific name-common name pairs in Javascript Object Notation (JSON) format and send that JSON string back to the mobile device. The device, upon receiving the response, will parse the JSON string. After parsing, the device will display the matches to the user. The user would then choose one of the match. And the process ends after the device has updated the scientific and common name of the leaf image captured.

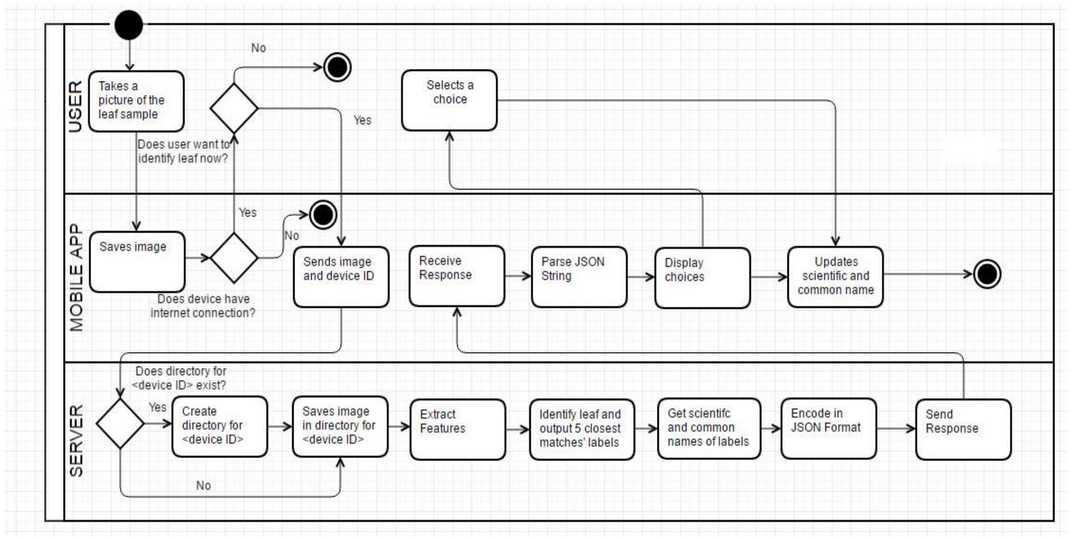

USER
MOBILE APP
SERVER
Takes a picture of the leaf sample
Does user want to identify leaf now?
No
Yes
Selects a choice
Saves image
Does device have internet connection?
Yes
No
Sends image and device ID
Receive Response
Parse JSON String
Display choices
Updates scientific and common name
Does directory for <device ID> exist?
Yes
Create directory for <device ID>
Saves image in directory for <device ID>
No
Extract Features
Identify leaf and output 5 closest matches' labels
Get scientifc and common names of labels
Encode in JSON Format
Send Response

Figure 18: Activity Flow Diagram

E. System Architecture

1. LIBSVM

LIBSVM is a library for support vector machines. LIBSVM was used in computer vision, natural language processing, neuroimaging, and bioinformatics. It supports two-class and multiclass support vector classification, support vector regression, and one-class support vector machine. The LIBSVM package includes the core C/C++ programs and sample data, the tools subdirectory, which is used for checking data format and for selecting SVM parameters, and other sub-directories containing pre-built binary files to other languages, like Python, Java, etc. LIBSVM supports five SVM formulations, namely, C-Support Vector Classification, ν-Support Vector Classification, Distribution Estimation (or the one-class SVM), ϵ-Support Vector Regression, and ν-Support Vector Regression. LIBSVM also supports probability estimates, even though SVM predicts only the class label.

2. OpenCV

OpenCV (Open Source Computer Vision Library) is an open source library used in computer vision and machine learning. It was written in C++ and has C, C++, Python, Java, and MATLAB interfaces, supported in Windows, Linux and Mac OS. It has more than 2500 optimized algorithms, which includes computer vision and machine learning algorithms. These algorithms can be used for face detection, object identification, image classification, etc. [45]

F. Technical Architecture

The system is downloadable and is run on an Android mobile device. The device's internal storage is where the database of leaves and gallery of leaves are stored. The database of leaves can be downloaded from the server with a reliable internet connection.

1. Minimum System Requirements

Mobile Device:

1. Processor: 1.4GHz Dual-core
2. RAM: 1 GB
3. Internal Memory: 8 GB
4. Camera: 5 MP, rear
5. 3G/Wi-Fi enabled
6. Operating System: Android v.2.3.x (Gingerbread)
7. A reliable internet connection

Server:

1. Processor: 1GHz
2. RAM: 2 GB
3. Internal Memory: 100 GB
4. Apache PHP version 5.6.8
5. Operating System: Windows Server 2012
6. A reliable internet connection

V. Results

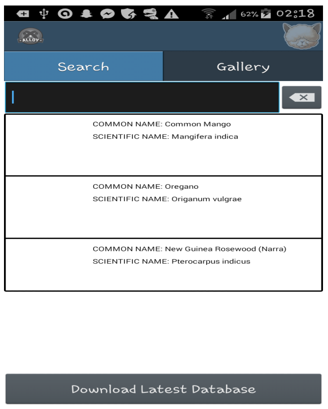
A. Splash Screen

Upon opening the application, the splash screen, as shown in Figure 19, displaying the Alloy standard splash screen, is displayed. After the splash screen, two tabs are displayed: ”Search” and ”Gallery”.

Figure 19: Alloy Splash Screen

B. Search

The Search window, as shown in Figure 20, is opened on the start of the application, after the splash screen closes.



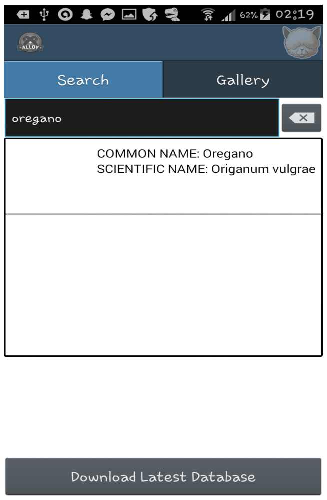
COMMON NAME: Common Mango
SCIENTIFIC NAME: Mangifera indica

COMMON NAME: Oregano
SCIENTIFIC NAME: Origanum vulgrae

COMMON NAME: New Guinea Rosewood (Narra)
SCIENTIFIC NAME: Pterocarpus indicus

Figure 20: Search Window

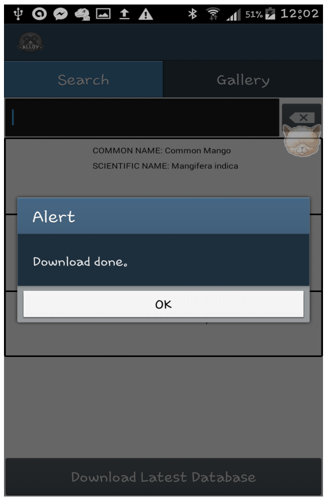
The user can search the database of leaves using the search box. After a successful query, the results are displayed on the list as shown in Figure 21.



COMMON NAME: Oregano
SCIENTIFIC NAME: Origanum vulgrae

Figure 21: Search Window After Displaying Search Results

The user can tap on a row to view the leaf's image. The user can also download the latest database from the server. Once the download is complete, an alert dialog will tell the user that the database has been updated as shown in Figure 22.



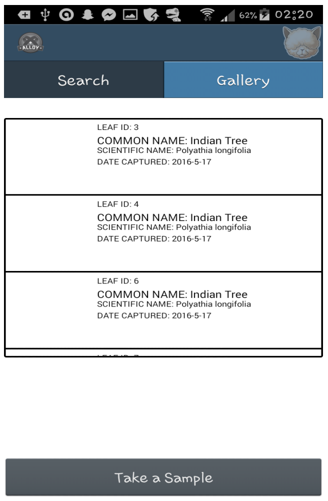
COMMON NAME: Common Mango
SCIENTIFIC NAME: Mangifera indica

Download done.

Figure 22: Alert After Database Is Successfully Downloaded

C. Gallery

The Gallery window can be opened through the Gallery tab as shown in Figure 23. Here, the list of all user-captured leaf images are stored and the user can browse through it.



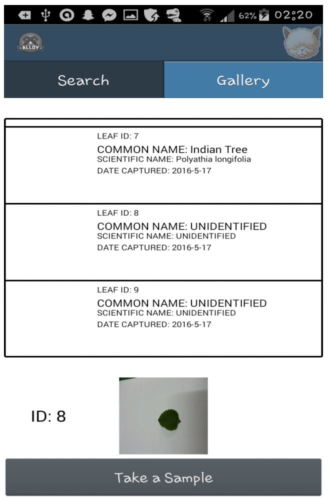
LEAF ID: 3
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

LEAF ID: 4
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

LEAF ID: 6
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

Figure 23: Gallery Window

The user can view the image of a leaf by tapping as shown in Figure 24. The user can then either identify a leaf or delete an entry by double-tapping as shown in Figure 25.

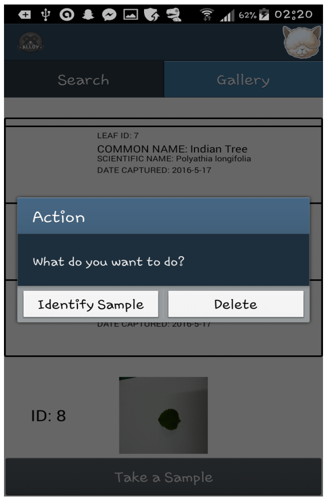


LEAF ID: 7
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

LEAF ID: 8
COMMON NAME: UNIDENTIFIED
SCIENTIFIC NAME: UNIDENTIFIED
DATE CAPTURED: 2016-5-17

LEAF ID: 9
COMMON NAME: UNIDENTIFIED
SCIENTIFIC NAME: UNIDENTIFIED
DATE CAPTURED: 2016-5-17

Figure 24: Gallery Window When User Taps an Item

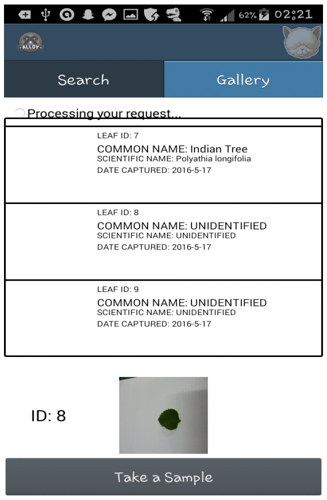


LEAF ID: 7
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

ID: 8

Figure 25: Prompt When A User Doubletaps an Item

When a user opts to identify/re-identify a leaf in the gallery, and if the device has an active, reliable internet connection, the device would send the image to the server to identify the leaf. An activity indicator is present to notify the user of the ongoing process, as shown in Figure 26.



Processing your request...

LEAF ID: 7
COMMON NAME: Indian Tree
SCIENTIFIC NAME: Polyathia longifolia
DATE CAPTURED: 2016-5-17

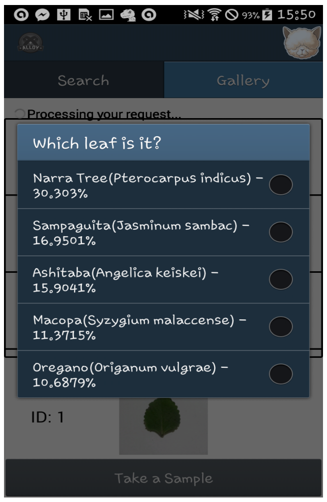
LEAF ID: 8
COMMON NAME: UNIDENTIFIED
SCIENTIFIC NAME: UNIDENTIFIED
DATE CAPTURED: 2016-5-17

LEAF ID: 9
COMMON NAME: UNIDENTIFIED
SCIENTIFIC NAME: UNIDENTIFIED
DATE CAPTURED: 2016-5-17

ID: 8

Figure 26: Activity Indicator for Processing Identify Request

An option dialog will then be displayed displaying the 5 closest choices. The scientific names, common names, and probabilities are displayed as shown in Figure 27. After the user has chosen one of the options, the database entry of that leaf is updated and the list is refreshed. However, if there is no internet connection, a prompt will display showing that the device must be connected to the internet to proceed with the identification/re-identification process as shown in Figure 28.



Which leaf is it?

Narra Tree(Pterocarpus indicus) - 30.303%

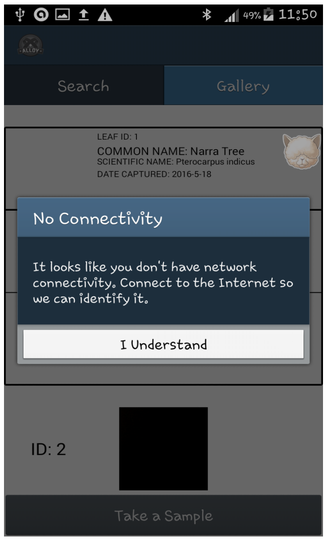
Sampaguita(Jasminum sambac) - 16.9501%

Ashitaba(Angelica keiskei) - 15.9041%

Macopa(Syzygium malaccense) - 11.3715%

Oregano(Origanum vulgrae) - 10.6879%

Figure 27: Option Dialog Displaying the 5 Closest Choices



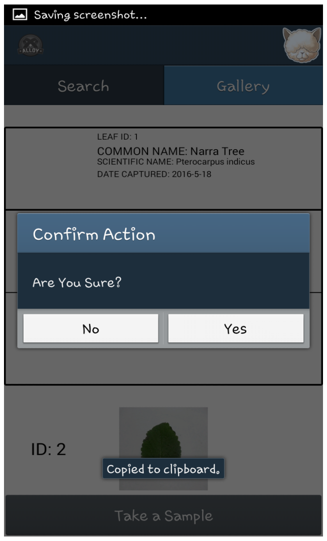
LEAF ID: 1
COMMON NAME: Narra Tree
SCIENTIFIC NAME: Pterocarpus indicus
DATE CAPTURED: 2016-5-18

No Connectivity

It looks like you don't have network connectivity. Connect to the Internet so we can identify it.

Figure 28: Prompt When User Attempts to Identify Image Without a Connection

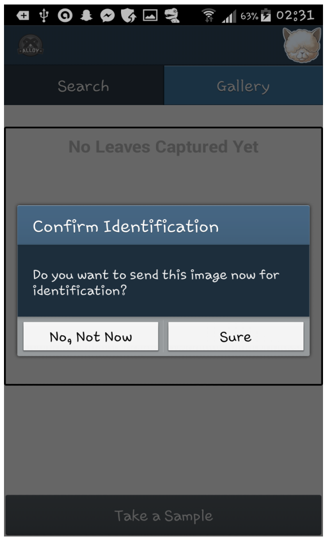
When a user chooses to delete a leaf image, a prompt will appear as shown in Figure 29. If the user chooses "Yes," the entry will be deleted.



LEAF ID: 1
COMMON NAME: Narra Tree
SCIENTIFIC NAME: Pterocarpus indicus
DATE CAPTURED: 2016-5-18

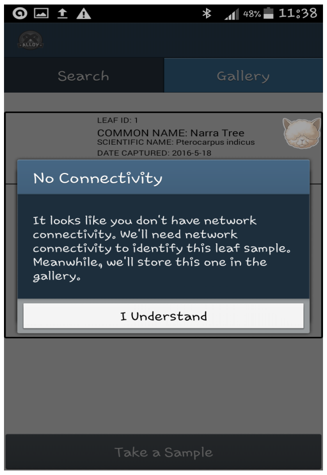
Figure 29: Prompt for Deleting an Item

A user can capture a new leaf image by tapping the "Take a Sample" button, and after tapping, the device's camera application will be opened. After the user successfully captures a picture of the leaf sample, it will be stored in the file system of the device. If the device has an active internet connection, the user can either opt to identify the image sample now or identify it at a later time as shown in Figure 30. The process for identification is the same as that when identifying/re-identifying a leaf from the gallery. If the device has no internet connection, a prompt will display stating that an internet connection is required to identify the image sample, as shown in Figure 31.



Do you want to send this image now for identification?

Figure 30: Prompt When User Attempts to Identify Image With a Connection



LEAF ID: 1
COMMON NAME: Narra Tree
SCIENTIFIC NAME: Pterocarpus indicus
DATE CAPTURED: 2016-5-18

It looks like you don't have network connectivity. We'll need network connectivity to identify this leaf sample. Meanwhile, we'll store this one in the gallery.

Figure 31: Prompt When An Image is Captured Without a Connection

VI. Discussions

LeafSnap PH is a mobile application that aims to classify Philippine plants by leaf images with a mobile front-end. Leafsnap[7] has the same goal, however, this uses Support Vector Machine (SVM) to identify the leaf sample, Leafsnap is an iOS-only application, and Leafsnap's database only contains American plants.

The feature extraction of the leaf is the same as [7]. The image is first converted from RGB to HSV. The Expectation-Maximization (EM) algorithm is used to separate the leaf from the background and then get the shape of the leaf using Canny Edge Detection. Using the image obtained from Canny Edge Detection, we determine the contour points and use these points as a reference for the image produced by the EM algorithm. We then proceed to get the white areas in the contour points from a radius of 5 units to 30 units with a 1 unit increment. This will be used to produce the curvature image, and is used as the feature for identifying the leaf. One major observation is that using the device's flash to capture an image drastically alters the HSV values of the image, thus when the EM algorithm is executed, the leaf cannot be fully separated from the background, thus using flash in capturing images is highly discouraged. Another issue that was addressed is that sometimes the background is too dark, and thus, using the EM algorithm treats the whole leaf as part of the background. The solution is to brighten the image before processing it for feature extraction. The brightness level, however, was achieved only through trial-and-error means, and therefore, a much more robust solution is needed.

The mobile application was written in HTML5, Javascript and CSS in compliance to the Appcelerator platform. SQLite is also needed to perform in-device database operations, namely, storing leaf information, deleting leaf information, and updating leaf information. For the database of leaves, it is assumed that Javacript Object

Notation (JSON) will be the standard format used as it is lightweight and is easily serialized into and out of the Appcelerator platform, which is Javascript-based[46]. For the feature extraction and labeling of the leaf in the server side, Visual C++, with the OpenCV extension for image processing, was used in writing the application. PHP was also used in the server side to handle the POST request by the mobile application.

A major issue in developing the mobile application is memory management. With such little RAM, comapred to desktops and laptops, operations are much more limited. Displaying all the leaf images in one go resulted in a lack of memory and thus, some of the images were not displayed. Therefore, a remedy is to only create a single image view and the image of the leaf can only be viewed upon tapping on a leaf sample.

The model was trained using 240 images. There are 8 classes and each class consist of 30 images. For the test set, 40 images that were not part of the 240-image training set were used. The 40 images were composed of 5 images from each of the 8 classes. Obtaining the optimal C, with a value of 2.0, and gamma, with a value of 3.0517578125e-05, parameters is done by running the grid.py script (which uses the RBF kernel) from LIBSVM. The model also underwent a 5-fold cross-validation. The model resulted with a 65% accuracy with the test set, when considering the closest answer, and a 90% accuracy, when considering the 5 closest answers.

VII. Conclusions

LeafSnap PH has been developed to give users a mobile front-end for identifying leaf samples of Philippine plants. This addresses the rapid growth of the mobile platform, the rapid increase in the number of mobile users, and the lack of a mobile application for such purpose. The convenience and quickness of the program allows those who have little to no botanical background to be able to identify leaves.

The accuracy of the model is 65% when considering the closest result, but taking into account the 5 closest results, the accuracy would be 90%. Leafsnap has an accuracy of 96.8% considering the 5 closest results[7]. Leafsnap US has a database of 184 species, compared to the model used, which has 8, therefore, comparing the accuracy of the two datasets would lead to inconclusive results.

VIII. Recommendations

LeafSnap PH can be further improved by increasing the accuracy of the training model to 90%. This can be achieved by providing more image inputs per class to the training set. The data can also be improved by taking into account the size of the leaf. Aside from increasing input images, other feature-extraction algorithms can be used. It is highly recommended for future studies to try the principal component analysis[22] or the vector of aggregated contour fragments[21]. To cover a wider range of plant species, it is also recommended to include complex leaves and conifers into the dataset.

An iOS version of it can be developed to expand the scope of users of the application. A map function, where location data of plants are sent to a server and can be viewed by all users, can also be implemented to utilize the longitude and latitude components of the database.

As stated in the conclusion, the accuracy of the model compared to Leafsnap is slightly lower, but the model has significantly fewer classes than that of Leafsnap. Therefore, it is highly recommended to increase the size of the database, by adding more classes, to make the results more conclusive.

IX. Bibliography

[1] T. Weier, C. Stocking, M. Barbour, and T. Rost, *Botany: An Introduction to Plant Biology*. John Wiley and Sons, 1982.

[2] D. Woodland, *Contemporary Plant Systematics*. Andrews University Press, 2009.

[3] "Plant species numbers." http://www.bgci.org/ourwork/1521/. Accessed: 2015-02-25.

[4] Z. Wang, Z. Chi, *et al.*, "Leaf image retrieval with shape features," *Lectures Notes in Computer Science*, vol. Advances in Information Systems: 4th International Conference, VISUAL 2000, Lyon, France, November 2000: Proceedings, 2000.

[5] Q. Wu, C. Zhou, and C. Wang, "Feature extraction and automatic recognition of plant leaf using artificial neural network," *Advances in Artificial Intelligence*, 2006.

[6] J.-X. Du, D.-S. Huang, *et al.*, "Computer-aided plant species identification (CAPSI) based on leaf shape matching technique," *Transactions of the Institute of Measurement and Control*, 2006.

[7] N. Kumar, P. Belhumeur, *et al.*, "Leafsnap: A computer vision system for automatic plant species identification," *Lecture Notes in Computer Science/Image Processing, Computer Vision, Pattern Recognition and Graphics*, vol. Computer Vision ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012. Proceedings, Part 6, 2012.

[8] C. University, U. of Maryland, and S. Institution, "Leafsnap." https://itunes.apple.com/us/app/leafsnap/id430649829?mt=8, 2014. Accessed: 2014-10-15.

[9] C. University, U. of Maryland, and S. Institution, "Leafsnap for ipad." https://itunes.apple.com/us/app/leafsnap-for-ipad/id433522683?mt=8, 2014. Accessed: 2014-10-15.

[10] C. University, U. of Maryland, and S. Institution, "Leafsnap uk." https://itunes.apple.com/us/app/leafsnap-uk/id877397884?mt=8, 2014. Accessed: 2014-10-15.

[11] C. University, U. of Maryland, and S. Institution, "Leafsnap: An electronic field guide." http://leafsnap.com/about/, 2011. Accessed: 2014-10-15.

[12] M. Sulč, "Image-based recognition of plants," 2012.

[13] T. Sixta, "Image and video-based recognition of natural objects," 2011.

[14] eMarketer, "Smartphone users worldwide will total 1.75 billion in 2014." http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536, 2014. Accessed: 2014-10-16.

[15] C. Ratcliff, "65https://econsultancy.com/blog/64376-65-of-global-smartphone-owners-use-android-os-stats#i.5nxxql18m7fbmx, 2014. Accessed: 2014-10-16.

[16] J. Pinaroc, "The philippines continues to embrace android." http://www.zdnet.com/the-philippines-continues-to-embrace-android-7000024072/, 2013. Accessed: 2014-10-16.

[17] T. Munisami, M. Ramsurn, *et al.*, "Plant leaf recognition using shape features and colour histogram with k-nearest neighbour classifiers," *Second International Symposium on Computer Vision and the Internet*, vol. 58, 2015.

[18] J. Chaki and R. Parekh, "Plant leaf recognition using shape based features and neural network classifiers," *International Journal of Advanced Computer Science and Applications*, vol. 2, 2011.

[19] V. Satti, A. Satya, and S. Sharma, "An automatic leaf recognition system for plant identification using machine vision technology," *International Journal of Engineering Science and Technology*, vol. 58, 2013.

[20] M. G. Larese, R. M. Craviotto, *et al.*, "Legume identification by leaf vein images classification," *Pattern Recognition*, vol. 47, 2014.

[21] S. Bai, X. Wang, and B. Xiang, "Aggregating contour fragments for shape classifications," *IEEE Conference on Image Processing*, 2014.

[22] A. Ehsanirad, "Plant classification based on leaf recognition," *International Journal of Computer Science and Information Security*, vol. 8, 2010.

[23] D. K. Srivastava and L. Bhambhu, "Data classification using support vector machine," *Journal of Theoretical and Applied Information Technology*, 2010.

[24] W. S. Noble, "What is a support vector machine," *Nature Biotechnology*, vol. 24, 2006.

[25] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification." http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf, 2003. Accessed: 2015-05-31.

[26] "Plant structures: Leaves." http://00.edu-cdn.com/files/592001_593000/592099/dichotomous-key-3.jpg. Accessed: 2016-05-20.

[27] "A simple dichotomous key example." http://www.ext.colostate.edu/mg/Gardennotes/134.html. Accessed: 2016-05-20.

[28] N. Mccrea, "Classification." http://homepages.inf.ed.ac.uk/rbf/HIPR2/classify.htm. Accessed: 2016-05-21.

[29] E. G. Learned-Miller, "Introduction to computer vision." https://people.cs.umass.edu/~elm/Teaching/Docs/IntroCV_1_19_11.pdf, 2011. Accessed: 2016-05-20.

[30] S. Borman, "The expectation maximization algorithm a short tutorial." http://www.cs.utah.edu/~piyush/teaching/EM_algorithm.pdf, 2004. Accessed: 2015-05-31.

[31] R. Fisher, S. Perkins, *et al.*, "An introduction to machine learning theory and its applications: A visual tutorial with examples." https://www.toptal.com/machine-learning/machine-learning-theory-an-introductory-primer. Accessed: 2016-05-21.

[32] A. Smola and S. Vishwanathan, *Introduction to Machine Learning*. Cambridge University Press, 2008.

[33] "Everything you wanted to know about machine learning, but were too afraid to ask (part one)." https://blog.bigml.com/2013/02/15/everything-you-wanted-to-know-about-machine-learning-but-were-too-afraid-to-ask-part-one/. Accessed: 2016-05-21.

[34] S. Ray, "Essentials of machine learning algorithms (with python and r codes)." http://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/. Accessed: 2016-05-21.

[35] "Introduction to support vector machines." http://www.svms.org/introduction.html. Accessed: 2015-05-31.

[36] "Everything you wanted to know about the kernel trick (but were too afraid to ask)." http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html. Accessed: 2016-05-20.

[37] "Rbf svm parameters." http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html. Accessed: 2016-05-20.

[38] A. Ben-Hur and J. Weston, "A users guide to support vector machines." http://www.cs.colostate.edu/~asa/pdfs/howto.pdf. Accessed: 2016-05-20.

[39] H. Yang, "Bon appétit for apps: Young american consumers' acceptance of mobile applications," *Journal of Computer Information Systems*, vol. 53, 2013.

[40] J. F. Clark, "History of mobile applications." http://www.uky.edu/~jclark/mas490apps/History%20of%20Mobile%20Apps.pdf. Accessed: 2014-12-09.

[41] H. Zheng, Y. Li, and D. Jiang, "Empirical study and model of user's acceptance for mobile commerce in china," *International Journal of Computer Science Issues*, vol. 9, 2012.

[42] A. Zamfiroiu and B. Vintila, "Management of mobile web application development with quality assurance," *Journal of Mobile, Embedded and Distributed Systems*, 2013.

[43] C. Janssen, "Android sdk." http://www.techopedia.com/definition/4220/android-sdk. Accessed: 2014-12-12.

[44] "Appcelerator platform." https://docs.appcelerator.com/platform/latest/#!/guide/Appcelerator_Platform. Accessed: 2016-05-19.

[45] O. D. Team, "About." http://opencv.org/about.html. Accessed: 2016-05-20.

[46] "Working with json data." http://docs.appcelerator.com/platform/latest/#!/guide/Working_with_JSON_Data. Accessed: 2016-05-18.

X. Appendix

A. Source Code

source–code/mobile/views/index.xml

```
<Alloy>
        <TabGroup backgroundColor="white" exitOnClose="true" >
            <Tab id="tab1" title="Search" icon="KS_nav_views.png">
                <Require src="search" id="search" />
            </Tab>
            <Tab id="tab2" title="Gallery">
                <Window id="gallery">
                        <View id = "captureButtonView" height="10%" top="90%">
                                <Button width="100%" height = "100%" title="Take a Sample" onClick
                                    ="showCamera" />
                        </View>
                </Window>
            </Tab>
        </TabGroup>
</Alloy>
```

source–code/mobile/controllers/index.js

```
var idNum = 0;
var firstLoad = true;
var url = '192.168.42.63:80/leafsnap/server.php'; //url to server
var db = Ti.Database.open('leafDB'); //open database
var fname;
var json;
var deviceID = Ti.Platform.getId();
var optionDialog;
var leafList = Ti.UI.createScrollView({
         layout: "vertical",
         top: "5%",
         height: "60%",
         width: "100%",
         borderColor: "black",
         borderWidth: 2,
         borderRadius: 5
});

var leafImage = Ti.UI.createImageView({
        height: 100,
        width: 100,
        top: "70%",
});

var idCaption = Ti.UI.createView({
        height: 100,
        width: 100,
        top: "70%",
        left: 0
});

var idCaptionLabel = Ti.UI.createLabel({
        font: {fontSize:20},
        color: "black",
        touchEnabled: false
});
idCaption.add(idCaptionLabel);

function showCamera(){
        var f;
        var db;
        var longt, lat;
        Ti.Media.showCamera({
                showControls: true,
                mediaTypes: Ti.Media.MEDIA_TYPE_PHOTO,
                allowEditing: false,
                success: function(event){
                        var d= new Date();
                        var capturedImage = event.media;
                        idNum++;
                        fname = deviceID + "_" + d.getFullYear() + "_" + (d.getMonth() + 1) + "_"
                            + d.getDate() + "_" + idNum + ".jpg";
                        f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, fname);
                        f.write(capturedImage);
                        Ti.Geolocation.Android.manualMode = false;
                        Ti.Geolocation.accuracy = Ti.Geolocation.ACCURACY_HIGH;
                        Ti.Geolocation.getCurrentPosition(function(e){
                                if(e.error){
```

```
                longt = 0;
                lat = 0;
        }
        else{
                longt = e.coords.longitude;
                lat = e.coords.latitude;
        }

});
db = Ti.Database.open('leafDB');
db.execute('INSERT INTO leaves (id, filename,longitude,latitude,month,day,
     year,sciName,commonName) VALUES (?,?,?,?,?,?,?,?,?)', idNum, fname,
     longt, lat, d.getMonth() + 1, d.getDate(), d.getFullYear() ,'','');
if (Ti.Network.networkType == Ti.Network.NETWORK_NONE){
        var alertDialog = Ti.UI.createAlertDialog({
                title: "No Connectivity",
                message: "It looks like you don't have network
                     connectivity. We'll need network connectivity to
                     identify this leaf sample. Meanwhile, we\'ll store
                     this one in the gallery.",
                buttonNames:['I Understand']
        });
        alertDialog.show();
        refreshGallery();
}
else{
        var alertDialog = Ti.UI.createAlertDialog({
                title: "Confirm Identification",
                message: "Do you want to send this image now for
                     identification?",
                buttonNames:['Sure', 'No, Not Now']
        });
        alertDialog.addEventListener('click',function(e){
                if(e.index==0){
                        //identify plant
                        var activityIndicator = Ti.UI.
                             createActivityIndicator({
                          color: 'black',
                          font: {fontFamily:'Helvetica Neue', fontSize
                               :15},
                          message: 'Processing your request...',
                          style: Ti.UI.ActivityIndicatorStyle.DARK,
                          top:0,
                          left:10,
                          height:Ti.UI.SIZE,
                          width:Ti.UI.SIZE
                        });
                        $.gallery.add(activityIndicator);
                        activityIndicator.show();
                        var xhr = Titanium.Network.createHTTPClient();
                        xhr.open('POST',url);
                        xhr.onload = function(response){
                                var response = this.responseText;
                                        json = JSON.parse(response);
                                        var choicesArray = new Array();
                                        for(var i = 0; i < 5; i++){
                                                choicesArray[i] = json.
                                                     choices[i].common +
                                                     "(" + json.choices[i].
                                                     scientific + ") - " +
                                                      json.choices[i].proby
                                                      + "%";
                                        }
                                        optionDialog = null;
                                        optionDialog = Ti.UI.
                                             createOptionDialog({
                                                options: choicesArray,
                                                destructive: 0,
                                                title : "Which leaf is it
                                                     ?"
                                        });
                                        optionDialog.addEventListener('
                                             click', function(e){
                                                db.execute('UPDATE leaves
                                                     SET sciName=? WHERE id
                                                     =?', json.choices[e.
                                                     source.selectedIndex].
                                                     scientific, idNum);
                                                db.execute('UPDATE leaves
                                                     SET commonName=? WHERE
                                                      id=?', json.choices[e
                                                     .source.selectedIndex
                                                     ].common, idNum);
                                                refreshGallery();
                                                activityIndicator.hide();
                                        });
                                        optionDialog.show();
                        };
                        xhr.send({
                            theImage: f,
                            imgFilename: fname,
                            deviceID: deviceID
```

```
                                                });

                                        }
                                        else if (e.index == 1){
                                                var alertDialog = Ti.UI.createAlertDialog({
                                                        title: "Store Image",
                                                        message: "Image is now stored in the
                                                             gallery, and you may opt to identify
                                                             the leaf at a later time.",
                                                        buttonNames:['OK']
                                                });
                                                refreshGallery();
                                        }
                                });
                                alertDialog.show();
                        }
                },
                cancel: function(){
                        return;
                },
                error: function(){
                        return;
                }
        });

}

function refreshGallery(){
        $.gallery.remove(leafList);
        leafList = null;
        leafList = Ti.UI.createScrollView({
                layout: "vertical",
                top: "5%",
                height: "60%",
                width: "100%",
                borderColor: "black",
                borderWidth: 2,
                borderRadius: 5
        });
        populateLeafView();
}


function populateLeafView(){

        var set = false; //boolean for checking non-emptiness of database
        var leavesSet = db.execute('SELECT id,filename,longitude,latitude,month,day,year,sciName,
             commonName FROM leaves');
        var f;
        var common, scientific;
        var leafId, leafFilename, leafLong, leafLat, leafMonth, leafDay, leafYear, leafSci,
             leafCommon, leafRow;
        var captionView, commonNameLabel, sciNameLabel, dateLabel, idLabel;
        /*leafList = Ti.UI.createScrollView({
                         layout: "vertical",
                         top: 0,
                         height: "85%",
                         width: "100%"
        });*/

        var leafData, alertDialog;
        while (leavesSet.isValidRow()){
                set = true;

                leafId = leavesSet.fieldByName('id');
                leafFilename = leavesSet.fieldByName('filename');
                leafLong = leavesSet.fieldByName('longitude');
                leafLat = leavesSet.fieldByName('latitude');
                leafMonth = leavesSet.fieldByName('month');
                leafDay = leavesSet.fieldByName('day');
                leafYear = leavesSet.fieldByName('year');
                leafSci = leavesSet.fieldByName('sciName');
                leafCommon = leavesSet.fieldByName('commonName');
                if (firstLoad){
                        idNum = leafId;
                }
                leafRow = Ti.UI.createView({
                        width: '100%',
                        height: 100,
                        top: 0,
                        left: 0,
                        borderColor: "black",
                        borderWidth: 1,
                        leaf_id: leafId,
                        focusable: true,
                        backgroundSelectedColor: "#BCED91"
                });
                if (leafSci == ''){
                        scientific = "UNIDENTIFIED";
                }
                else{
                        scientific = leafSci;
```

```
}
if (leafCommon == ''){
        common = "UNIDENTIFIED";
}
else{
        common = leafCommon;
}
captionView = Ti.UI.createView({
        left: 100,
        top: 0,
        touchEnabled: false
});
idLabel = Ti.UI.createLabel({
        font:{fontSize: 10},
        color: "black",
        text: "LEAF ID: " + leafId,
        top: 5,
        left: 5,
        touchEnabled: false
});
commonNameLabel = Ti.UI.createLabel({
        font:{fontSize:13},
        color: "black",
        text: "COMMON NAME: " + common,
        top: 20,
        left: 5,
        touchEnabled: false
});
sciNameLabel = Ti.UI.createLabel({
        font: {fontSize:10},
        color: "black",
        text: "SCIENTIFIC NAME: " + scientific,
        top: 35,
        left: 5,
        touchEnabled: false
});
dateLabel = Ti.UI.createLabel({
        font: {fontSize:10},
        color: "black",
        top: 50,
        left: 5,
        text: "DATE CAPTURED: " + leafYear + "-" + leafMonth + "-" + leafDay,
        touchEnabled: false
});
captionView.add(idLabel);
captionView.add(commonNameLabel);
captionView.add(sciNameLabel);
captionView.add(dateLabel);
//leafRow.add(leafImage);
leafRow.add(captionView);
leafRow.addEventListener('click', function(e){
        //db = Ti.Database.open('leafDB');
        leafData = db.execute('SELECT id,filename,longitude,latitude,month,day,
             year,sciName,commonName FROM leaves WHERE id=?', e.source.leaf_id);
        f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, leafData
             .fieldByName('filename'));
        leafImage.setImage(f);
        idCaptionLabel.setText("ID: " + e.source.leaf_id);
        //db.close();
});
leafRow.addEventListener('dblclick',function(e){
        //db = Ti.Database.open('leafDB');
        var leafID = e.source.leaf_id;
        leafData = db.execute('SELECT id,filename,longitude,latitude,month,day,
             year,sciName,commonName FROM leaves WHERE id=?', leafID);
        var filename = leafData.fieldByName('filename');
        f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, leafData
             .fieldByName('filename'));
        alertDialog = null;
        alertDialog = Ti.UI.createAlertDialog({
                title: "Action",
                message: "What do you want to do?",
                buttonNames:['Delete', 'Identify Sample']
        });
        alertDialog.show();
        alertDialog.addEventListener('click',function(e){
                if(e.index==1){
                        if (Ti.Network.networkType == Ti.Network.NETWORK_NONE){
                                alertDialog = null;
                                alertDialog = Ti.UI.createAlertDialog({
                                        title: "No Connectivity",
                                        message: "It looks like you don't have
                                             network connectivity. Connect to the
                                             Internet so we can identify it.",
                                        buttonNames:['I Understand']
                                });
                                alertDialog.show();
                        }
                        else{
                                //identify plant
                                var activityIndicator = Ti.UI.
                                     createActivityIndicator({
                                          color: 'black',
                                          font: {fontFamily:'Helvetica Neue', fontSize
                                              :15},
                                          message: 'Processing your request...',
                                          style: Ti.UI.ActivityIndicatorStyle.DARK,
                                          top:10,
                                          left:10,
                                          height:Ti.UI.SIZE,
                                          width:Ti.UI.SIZE
                                        });
                                        $.gallery.add(activityIndicator);
                                        activityIndicator.show();
                                        var xhr = Titanium.Network.createHTTPClient({
                                                onload: function(e){
                                                        var response = this.responseText;
                                                        json = JSON.parse(response);
                                                        var choicesArray = new Array();
                                                        for(var i = 0; i < 5; i++){
                                                                choicesArray[i] = json.
                                                                    choices[i].common +
                                                                    "(" + json.choices[i].
                                                                    scientific + ") - " +
                                                                     json.choices[i].proby
                                                                     + "%";
                                                        }
                                                        optionDialog = null;
                                                        optionDialog = Ti.UI.
                                                            createOptionDialog({
                                                                options: choicesArray,
                                                                destructive: 0,
                                                                title : "Which leaf is it
                                                                    ?"
                                                        });
                                                        optionDialog.addEventListener('
                                                            click', function(e){
                                                                db.execute('UPDATE leaves
                                                                    SET sciName=? WHERE id
                                                                    =?', json.choices[e.
                                                                    source.selectedIndex].
                                                                    scientific, leafID);
                                                                db.execute('UPDATE leaves
                                                                    SET commonName=? WHERE
                                                                     id=?', json.choices[e
                                                                    .source.selectedIndex
                                                                    ].common, leafID);
                                                                refreshGallery();
                                                                activityIndicator.hide();
                                                        });
                                                        optionDialog.show();
                                                }
                                        });
                                        xhr.open('POST', url);
                                        xhr.send({
                                            theImage: f,
                                            imgFilename: filename,
                                            deviceID: deviceID
                                         });
                                }
                        }
                        else if(e.index == 0){
                                alertDialog = null;
                                alertDialog = Ti.UI.createAlertDialog({
                                        title: "Confirm Action",
                                        message: "Are You Sure?",
                                        buttonNames:['Yes','No']
                                });
                                alertDialog.addEventListener('click', function(e){
                                        if(e.index==0){
                                                f.deleteFile();
                                                db.execute('DELETE FROM leaves WHERE id
                                                    =?',leafID);
                                                refreshGallery();
                                                leafImage.setImage(null);
                                                idCaptionLabel.setText('');
                                        }
                                });
                                alertDialog.show();
                        }
                });
                //db.close();
        });
        leafList.add(leafRow);
        leavesSet.next();
}
//db.close();
if (set == false){
        mainView = Ti.UI.createView({
                width:'100%',
                height:'100%'
        });
        var noLeafLabel = Ti.UI.createLabel({
                top:10,
                        textAlign: 'center',
                        font: {fontSize:20, fontWeight:'bold'},
                        text: "No Leaves Captured Yet"
                });
                leafList.add(noLeafLabel);
        }
        $.gallery.add(leafList);
        $.gallery.add(leafImage);
        $.gallery.add(idCaption);
        firstLoad = false;
}

/*function showMap(){
        var Map = require('ti.map');
        var win = Ti.UI.createWindow();
        var id;
        //var plantViews[];
        //edit this section
        /*for(id=0;id<5;id++){
                plantViews[id] = Map.createAnnotation({
                        latitude: 5,
                        longitude: 10,
                        title: "SOME SCIENTIFIC NAME",
                        subtitle: "COMMON NAME",
                        pincolor: Map.ANNOTATION_GREEN,
                        myid: id
                });
        }
        var mapView = Map.createView({
                mapType: Map.NORMAL_TYPE,
                region:{
                        latitude: 122.0,
                        longitude: 13.0,
                        latitudeDelta: 1,
                        longitudeDelta: 1,
                },
                animate:true,
                regionFit:true,
                userLocation:true,
                //annotations:[plantViews],
        });
        win.add(mapView);

}*/

// initialize SQLite database
db.execute('CREATE TABLE IF NOT EXISTS leaves(id INTEGER PRIMARY KEY, filename TEXT, longitude
     REAL, latitude REAL, month INTEGER, day INTEGER, year INTEGER, sciName TEXT, commonName TEXT)
     ;');
//db.close();

//check for rear camera support
var cameras = Ti.Media.availableCameras;
var rearCameraExist = false;
for (var c=0; c< cameras.length; c++){
        if(cameras[c] == Ti.Media.CAMERA_REAR){
                rearCameraExist = true;
        }
}
if (!rearCameraExist){
        var alertDialog = Ti.UI.createAlertDialog({
                title: "No Connectivity",
                message: "It looks like this phone doesn't have a rear camera. This app needs one
                     to run.",
                buttonNames:['I\'ll Get A New Phone']
        });
        alertDialog.show();
        alertDialog.addEventListener('click',function(e){
                if(e.index==0){
                        $.win1.close();
                        var activity = Ti.Android.currentActivity;
                        activity.finish();
                }
        });
}

populateLeafView();

Ti.App.addEventListener('close', function(e){
        db.close();
        var activity = Ti.Android.currentActivity;
        activity.finish();
});

$.index.open();
```

source–code/mobile/views/search.xml

```
<Alloy>
        <Window id="search">
```

```
                <View id = "downloadXML" height="10%" top="90%">
                        <Button width="100%" height = "100%" title="Download Latest Database"
                             onClick="downloadDatabase" />
                </View>
        </Window>
</Alloy>
```

source–code/mobile/controllers/search.js

```
// Arguments passed into this controller can be accessed via the `$.args` object directly or:
var args = $.args;
var f;
var jsonLink = "192.168.42.63:80/leafsnap/leafdb.json";
var jsonfname = "leafdb.json";
var leafImage = Ti.UI.createImageView({
        height: 100,
        width: 100,
        top: "70%",
});

var JSONtext, JSONdata, json;

var i;
var textBox = Ti.UI.createSearchBar({
        showCancel: true,
        top: 0,
        left: 0,
        width:"100%",
        height:"10%",
        cancelButtonTitle: "Clear"
});

textBox.addEventListener('return', function(e){
        clearTable();
        textBox.hide();
        textBox.show();
        var searchAvailable = false;
        var searchString = textBox.value;
        var leafImgURL, leafSci, leafCommon, leafRow;
        var captionView, commonNameLabel, sciNameLabel, idLabel;
        var alertDialog;
        for (i in json){ //edit for XML
                leafImgURL = json[i].image;
                leafSci = json[i].scientific;
                leafCommon = json[i].common;
                if(((searchString.toUpperCase()).indexOf(leafSci.toUpperCase()) >= 0) || ((
                     searchString.toUpperCase()).indexOf(leafCommon.toUpperCase()) >= 0) ){
                        leafRow = Ti.UI.createView({
                                width: '100%',
                                height: 100,
                                top: 0,
                                left: 0,
                                borderColor: "black",
                                borderWidth: 1,
                                focusable: true,
                                backgroundSelectedColor: "#BCED91",
                                leaf_img: leafImgURL
                        });

                        captionView = Ti.UI.createView({
                                left: 100,
                                top: 0,
                                touchEnabled: false
                        });
                        commonNameLabel = Ti.UI.createLabel({
                                font:{fontSize:15},
                                color: "black",
                                text: "COMMON NAME: " + leafCommon,
                                top: 10,
                                left: 5,
                                touchEnabled: false
                        });
                        sciNameLabel = Ti.UI.createLabel({
                                font: {fontSize:15},
                                color: "black",
                                text: "SCIENTIFIC NAME: " + leafSci,
                                top: 30,
                                left: 5,
                                touchEnabled: false
                        });
                        captionView.add(commonNameLabel);
                        captionView.add(sciNameLabel);
                        leafRow.add(captionView);
                        leafRow.addEventListener('click', function(e){
                                leafImage.setImage(leaf_img);
                        });
                        leafView.add(leafRow);
                        searchAvailable = true;
                        }
        }
        if (searchAvailable == false){
                var noXMLLabel = Ti.UI.createLabel({
                        top:10,
                        textAlign: 'center',
                        font: {fontSize:20, fontWeight:'bold'},
                        text: "No Results Found"
                });
                leafView.add(noXMLLabel);
                empty = false;
        }
});

textBox.addEventListener('cancel', function(e){
        refreshTable();
});

$.search.add(textBox);
var leafView = Ti.UI.createScrollView({
                layout: "vertical",
                top: "10%",
                height: "55%",
                width: "100%",
                borderColor: "black",
                borderWidth: 2,
                borderRadius: 5
});

function downloadDatabase(){
        //download XML file
        if (Ti.Network.networkType == Ti.Network.NETWORK_NONE){
                var alertDialog = Ti.UI.createAlertDialog({
                        title: "No Connectivity",
                        message: "It looks like you don't have network connectivity. We'll need
                            network connectivity to download the latest XML file.",
                        buttonNames:['I Understand']
                });
                alertDialog.show();

        }
        else{
                var xhr = Ti.Network.createHTTPClient({
                        onload: function(){
                                f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory,
                                    jsonfname);
                                f.write(this.responseData);
                                alert("Download done.");
                                refreshTable();
                        }
                });
                var activityIndicator = Ti.UI.createActivityIndicator({
                  color: 'black',
                  font: {fontFamily:'Helvetica Neue', fontSize:15},
                  message: 'Processing your request...',
                  style: Ti.UI.ActivityIndicatorStyle.DARK,
                  top:0,
                  left:10,
                  height:Ti.UI.SIZE,
                  width:Ti.UI.SIZE
                });
                activityIndicator.show();
                xhr.open('GET', jsonLink);
                xhr.send();
        }
}

function clearTable(){
        $.search.remove(leafView);
        leafView = null;
        leafView = Ti.UI.createScrollView({
                layout: "vertical",
                top: "10%",
                height: "55%",
                width: "100%",
                borderColor: "black",
                borderWidth: 2,
                borderRadius: 5
        });
        $.search.add(leafView);
}
function refreshTable(){
        clearTable();
        loadTable();
}

function loadTable(){
        var empty = true;
        f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, jsonfname);
        if (f.exists() == true){
                empty = false;
                var leafImgURL, leafSci, leafCommon, leafRow;
                var captionView, commonNameLabel, sciNameLabel, idLabel;
                var alertDialog;
```

```
            JSONtext = f.read().text;
            JSONdata = JSON.parse(JSONtext);
            json = JSONdata.leaf;
            for (i in json){ //edit for XML
                empty = false;
                leafImgURL = json[i].image;
                leafSci = json[i].scientific;
                leafCommon = json[i].common;
                leafRow = Ti.UI.createView({
                    width: '100%',
                    height: 100,
                    top: 0,
                    left: 0,
                    borderColor: "black",
                    borderWidth: 1,
                    focusable: true,
                    backgroundSelectedColor: "#BCED91",
                    leaf_img: leafImgURL
                });

                captionView = Ti.UI.createView({
                    left: 100,
                    top: 0,
                    touchEnabled: false
                });
                commonNameLabel = Ti.UI.createLabel({
                    font:{fontSize:10},
                    color: "black",
                    text: "COMMON NAME: " + leafCommon,
                    top: 10,
                    left: 0,
                    touchEnabled: false
                });
                sciNameLabel = Ti.UI.createLabel({
                    font: {fontSize:10},
                    color: "black",
                    text: "SCIENTIFIC NAME: " + leafSci,
                    top: 30,
                    left: 0,
                    touchEnabled: false
                });
                captionView.add(commonNameLabel);
                captionView.add(sciNameLabel);
                leafRow.add(captionView);
                leafRow.addEventListener('click', function(e){
                    leafImage.setImage(leaf_img);
                });
                leafView.add(leafRow);
            }
        }
        else{
            var noXMLLabel = Ti.UI.createLabel({
                top:10,
                textAlign: 'center',
                font: {fontSize:20, fontWeight:'bold'},
                text: "No JSON File Found"
            });
            leafView.add(noXMLLabel);
            empty = false;
        }
        if (empty){
            mainView = Ti.UI.createView({
                width:'100%',
                height:'100%'
            });
            var noLeafLabel = Ti.UI.createLabel({
                top:10,
                textAlign: 'center',
                font: {fontSize:20, fontWeight:'bold'},
                text: "JSON File Empty"
            });
            leafView.add(noLeafLabel);
        }
}

f = Ti.Filesystem.getFile(Ti.Filesystem.applicationDataDirectory, jsonfname);
var g = Ti.Filesystem.getFile(Ti.Filesystem.resourcesDirectory, jsonfname);

//var data = Ti.XML.parseString(f.read().text);

if(f.exists() == false){
        f.write(g.read());
}

$.search.add(leafView);
refreshTable();
$.search.add(leafImage);
```

source–code/server/extractsingle.cpp

```
#include "stdafx.h"
```

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
#include <opencv/cv.h>
#include <numeric>
#include <iostream>
#include<fstream>
#include <boost/filesystem.hpp>
using namespace cv;
using namespace std;
using namespace boost::filesystem;
RNG rng(12345);

#define PI 3.1416
#define IMGROWS 600
#define IMGCOLS 300
#define CURVATURE_ROWS 25

const double Epsilon = 1.0e-08;
void observe_probs(const cv::Mat& probs)
{
        std::vector<double> t(probs.cols, 0.0);
        for (int n = 0; n < probs.rows; ++n) {
                const double* gamma_n = probs.ptr<double>(n);
                double s = 0.0;
                for (int k = 0; k < probs.cols; ++k) {
                        s += gamma_n[k]; // \gamma_{n,k}
                        t[k] += gamma_n[k];
                }
                assert(std::abs(s - 1.0) < Epsilon);
        }
        double total = std::accumulate(t.begin(), t.end(), 0.0);
        assert(std::abs(total - probs.rows) < Epsilon);
}



void observe_weights(const cv::Mat& weights)
{
        cv::MatConstIterator_<double> first = weights.begin<double>();
        cv::MatConstIterator_<double> last = weights.end<double>();
        double s = 0.0;
        while (first != last) { // loop over k
                s += *first; // *first means \pi_{k}
                ++first;
        }
        assert(std::abs(s - 1.0) < Epsilon);
}

void observe_labels_and_means(const cv::Mat& labels, const cv::Mat& means, int height, int width,
    String filename, String source)
{
        const int dimension = 2;
        cv::Mat rgb_image(height, width, CV_8UC3);
        cv::MatIterator_<cv::Vec3b> rgb_first = rgb_image.begin<cv::Vec3b>();
        cv::MatIterator_<cv::Vec3b> rgb_last = rgb_image.end<cv::Vec3b>();
        cv::MatConstIterator_<int> label_first = labels.begin<int>();

        cv::Mat means_u8;
        means.convertTo(means_u8, CV_8UC1, 255.0);
        cv::Mat means_u8c3 = means_u8.reshape(dimension);

        while (rgb_first != rgb_last) {
                const cv::Vec3b& rgb = means_u8c3.ptr<cv::Vec3b>(*label_first)[0];
                *rgb_first = rgb;
                ++rgb_first;
                ++label_first;
        }
        //Grayscale matrix
        cv::Mat grayscaleMat(rgb_image.size(), CV_8U);

        //Convert BGR to Gray
        cv::cvtColor(rgb_image, grayscaleMat, CV_BGR2GRAY);

        //Binary image
        cv::Mat binaryMat(grayscaleMat.size(), grayscaleMat.type());

        //Apply thresholding
        cv::threshold(grayscaleMat, binaryMat, 100, 255, cv::THRESH_BINARY);
        bitwise_not(binaryMat, grayscaleMat);
        //Show the results
        Mat zeroRow(30, grayscaleMat.cols, CV_8UC1, Scalar(0));
        Mat zeroCol(grayscaleMat.rows + 60, 30, CV_8UC1, Scalar(0));
        //extend image
        vconcat(grayscaleMat, zeroRow, grayscaleMat);
        vconcat(zeroRow, grayscaleMat, grayscaleMat);
        hconcat(grayscaleMat, zeroCol, grayscaleMat);
        hconcat(zeroCol, grayscaleMat, grayscaleMat);

        cv::imwrite(source + "\\EM\\" + filename + "_em_result.jpg", grayscaleMat);
}
```

```
int main(int argc, const char * argv[])
{
        Size imsize(IMGCOLS, IMGROWS);
        path p(argv[1]);
        int i, j, k, l;
        String filename = boost::filesystem::basename(p);
        String source = p.parent_path().string();
        ofstream myfile;
        boost::filesystem::path dir(source + "\\EM");
        if (boost::filesystem::create_directory(dir)){
                cout << "Created EM directory" << "\n\n";
        }
        myfile.open(filename + "_features", ios::trunc);
        if (extension(p.filename()) != ".jpg"){
                return -1;
        }
        else{
                cv::Mat loadedimage = cv::imread(source + "\\" + filename + ".jpg");
                cv::Mat image;
                cv::Mat brightImage;
                resize(loadedimage, image, imsize);
                cv::imwrite(source + "\\EM\\" + filename + "_resize.jpg", image);
                image.convertTo(brightImage, -1, 0.5, 80);
                cv::imwrite(source + "\\EM\\" + filename + "_bright.jpg", brightImage);
                assert(brightImage.type() == CV_8UC3);
                const int image_rows = brightImage.rows;
                const int image_cols = brightImage.cols;
                vector <Mat> newMatrix;
                cv::Mat hsvImage, newImage;
                vector <Mat> channels, newChannels;
                cvtColor(brightImage, hsvImage, CV_BGR2HSV);
                split(hsvImage, channels);
                newChannels.push_back(channels[1]);
                newChannels.push_back(channels[2]);
                merge(newChannels, newImage);


                newImage.convertTo(newImage, CV_8UC1);

                int dimension = 2;

                cv::Mat reshaped_image = newImage.reshape(1, image_rows * image_cols);
                assert(reshaped_image.type() == CV_8UC1);
                assert(reshaped_image.rows == image_rows * image_cols);
                assert(reshaped_image.cols == dimension);
                // create an input for the EM Algorithm
                cv::Mat samples;
                reshaped_image.convertTo(samples, CV_64FC1, 1.0 / 255.0);
                assert(samples.type() == CV_64FC1);
                assert(samples.rows == image_rows * image_cols);
                assert(samples.cols == dimension);

                const int cluster_num{ 2 };
                cv::EM model{ cluster_num };

                // prepare outputs
                cv::Mat labels;
                cv::Mat probs;
                cv::Mat log_likelihoods;

                // execute EM Algorithm
                cout << "Executing EM Algorithm for " << filename << "..." << endl;
                model.train(samples, log_likelihoods, labels, probs);

                assert(log_likelihoods.type() == CV_64FC1);
                assert(log_likelihoods.rows == image_rows * image_cols);
                assert(log_likelihoods.cols == 1);

                assert(labels.type() == CV_32SC1);
                assert(labels.rows == image_rows * image_cols);
                assert(labels.cols == 1);
                cout << "Observing Probabilities" << endl;
                assert(probs.type() == CV_64FC1);
                assert(probs.rows == image_rows * image_cols);
                assert(probs.cols == cluster_num);
                observe_probs(probs);
                cout << "Observing Labels and Means" << endl;
                const cv::Mat& means = model.get<cv::Mat>("means");
                assert(means.type() == CV_64FC1);
                assert(means.rows == cluster_num);
                assert(means.cols == dimension);
                observe_labels_and_means(labels, means, image_rows, image_cols, filename, source);
                cout << "Observing Weights" << endl;
                const cv::Mat& weights = model.get<cv::Mat>("weights");
                assert(weights.type() == CV_64FC1);
                assert(weights.rows == 1);
                assert(weights.cols == cluster_num);
                observe_weights(weights);

                //Getting contours
                vector<vector<Point>> contours;
                vector<Vec4i> hierarchy;
```

```
cv::Mat grayscaleImage = cv::imread(source + "\\EM\\" + filename + "_em_result.jpg
     ");
cv::Mat newgrayscaleImage = cv::imread(source + "\\EM\\" + filename + "_em_result.
     jpg");
assert(grayscaleImage.type() == CV_8UC1);
cv::cvtColor(grayscaleImage, newgrayscaleImage, CV_BGR2GRAY);
Canny(grayscaleImage, newgrayscaleImage, 100, 1500, 3);
imwrite(source + "\\Canny\\" + filename + "_canny.jpg", newgrayscaleImage);

Scalar colour;
cv::Mat cannyimage = newgrayscaleImage;
cv::Mat EMimage = cv::imread(source + "\\EM\\" + filename + "_em_result.jpg");
vector<Point> edge;
for (j = 0; j < cannyimage.rows; j++){
        for (i = 0; i < cannyimage.cols; i++){
                if (cannyimage.at<uchar>(j, i) == 255){
                        edge.push_back(Point(i, j));
                }
        }
}
float radius;
Mat cropped;
float whites = 0.0;
cv::Mat curvatureImage;
Rect r;
Mat roi;
Mat mask;
int area, zerocounter;
bool set = false;


cout << "Calculating areas..." << endl;
Mat colors(25, 1, CV_8UC1, Scalar(0));
for (i = 0; i < edge.size(); i++){
        j = 0;
        zerocounter = 0;
        for (radius = 5.0; radius < 30.0;){

                //initialize region of interest
                r.x = edge[i].x - radius;
                r.y = edge[i].y - radius;
                r.width = radius * 2;
                r.height = radius * 2;
                roi = Mat(EMimage, r);
                mask = Mat(roi.size(), roi.type(), Scalar::all(0));
                circle(mask, Point(radius, radius), radius, Scalar::all(255), -1);
                cropped = roi & mask;

                //calculate white points
                whites = 0;
                for (k = 0; k < cropped.rows; k++){
                        for (l = 0; l < cropped.cols; l++){
                                if (cropped.at<uchar>(l, k) == 255){
                                        whites++;
                                }
                        }
                }
                //get area
                unsigned char area = uchar((whites * 255) / (PI * radius * radius)
                     );
                colors.at<uchar>(0, j) = area;
                radius = radius + 1;
                j++;
                if ((int)area == 0)
                        zerocounter = zerocounter + 1;
        }
        if (!set && zerocounter < CURVATURE_ROWS){
                curvatureImage = colors;
                set = true;
        }
        else if (set && zerocounter < CURVATURE_ROWS){
                hconcat(curvatureImage, colors, curvatureImage);
        }
}
imwrite(source + "\\EM\\" + filename + "_curvature_image.jpg", curvatureImage);


int bins = 16;
float range[] = { 0, 256 };
const float* histRange = { range };
vector<Mat> histograms;
vector<float> features;
Mat histogram;
int ch[] = { 0 };
for (i = 0; i < curvatureImage.rows; i++){
        stringstream ss;
        ss << i;
        calcHist(&curvatureImage.row(i), 1, 0, Mat(), histogram, 1, &bins, &
             histRange, true, false);
        for (j = 0; j < histogram.rows; j++){
                features.push_back(histogram.at<float>(j, 0));
        }
```

```
                }
                int leafnumber = 0;
                myfile << leafnumber << " ";
                for (i = 0; i < features.size(); i++){
                        myfile << i + 1 << ":" << features[i] << " ";
                }
                myfile << "\n";
        }
        return 0;
}
```

source–code/server/keytosci.cpp

```
#include "stdafx.h"
#include <numeric>
#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <sstream>
#include "strtk.hpp"
using namespace std;

int main(int argc, const char * argv[]){
        string leafSciname, leafindexstr, line;
        fstream filename;
        map<int, string> leafNumMap;
        map<int, string>::iterator it;
        int leafindex;
        filename.open("leafNumSciList.txt", ios::in);
        while (getline(filename, line)){
                vector<string> vec;
                strtk::parse(line, ":", vec);
                leafindexstr = vec[0];
                leafindex = std::stoi(leafindexstr);
                leafSciname = vec[1];
                leafNumMap[leafindex] = leafSciname;
        }
        filename.close();
        leafindexstr = argv[1];
        leafindex = std::stoi(leafindexstr);
        it = leafNumMap.begin();
        it = leafNumMap.find(leafindex);
        leafSciname = it->second;
        cout << leafSciname;
}
```

source–code/server/scitocomm.cpp

```
#include "stdafx.h"
#include <numeric>
#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <sstream>
#include "strtk.hpp"
using namespace std;

int main(int argc, const char * argv[]){
        string leafSciname, line, leafCommonname;
        fstream filename;
        map<string, string> leafSciMap;
        map<string, string>::iterator it;
        filename.open("leafSciCommonList.txt", ios::in);
        while (getline(filename, line)){
                vector<string> vec;
                strtk::parse(line, ":", vec);
                leafSciname = vec[0];
                leafCommonname = vec[1];
                leafSciMap[leafSciname] = leafCommonname;
        }
        filename.close();
        leafSciname = argv[1];
        it = leafSciMap.begin();
        it = leafSciMap.find(leafSciname);
        leafCommonname = it->second;
        cout << leafCommonname;
}
```

source–code/training/extract.cpp

```
#include "stdafx.h"
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/ml/ml.hpp>
#include <opencv/cv.h>
```

```
#include <numeric>
#include <iostream>
#include<fstream>
#include <boost/filesystem.hpp>
using namespace cv;
using namespace std;
using namespace boost::filesystem;
RNG rng(12345);

#define PI 3.1416
#define IMGROWS 600
#define IMGCOLS 300
#define CURVATURE_ROWS 25

void observe_probs(const cv::Mat& probs)
{
        std::vector<double> t(probs.cols, 0.0);
        for (int n = 0; n < probs.rows; ++n) {
                const double* gamma_n = probs.ptr<double>(n);
                double s = 0.0;
                for (int k = 0; k < probs.cols; ++k) {
                        s += gamma_n[k]; // \gamma_{n,k}
                        t[k] += gamma_n[k];
                }
                assert(std::abs(s - 1.0) < Epsilon);
        }
        double total = std::accumulate(t.begin(), t.end(), 0.0);
        assert(std::abs(total - probs.rows) < Epsilon);
}

const double Epsilon = 1.0e-08;



void observe_weights(const cv::Mat& weights)
{
        cv::MatConstIterator_<double> first = weights.begin<double>();
        cv::MatConstIterator_<double> last = weights.end<double>();
        double s = 0.0;
        while (first != last) { // loop over k
                s += *first; // *first means \pi_{k}
                ++first;
        }
        assert(std::abs(s - 1.0) < Epsilon);
}

void observe_labels_and_means(const cv::Mat& labels, const cv::Mat& means, int height, int width,
     String filename, String source)
{
        const int dimension = 2;
        cv::Mat rgb_image(height, width, CV_8UC3);
        cv::MatIterator_<cv::Vec3b> rgb_first = rgb_image.begin<cv::Vec3b>();
        cv::MatIterator_<cv::Vec3b> rgb_last = rgb_image.end<cv::Vec3b>();
        cv::MatConstIterator_<int> label_first = labels.begin<int>();

        cv::Mat means_u8;
        means.convertTo(means_u8, CV_8UC1, 255.0);
        cv::Mat means_u8c3 = means_u8.reshape(dimension);

        while (rgb_first != rgb_last) {
                const cv::Vec3b& rgb = means_u8c3.ptr<cv::Vec3b>(*label_first)[0];
                *rgb_first = rgb;
                ++rgb_first;
                ++label_first;
        }
        //Grayscale matrix
        cv::Mat grayscaleMat(rgb_image.size(), CV_8U);

        //Convert BGR to Gray
        cv::cvtColor(rgb_image, grayscaleMat, CV_BGR2GRAY);

        //Binary image
        cv::Mat binaryMat(grayscaleMat.size(), grayscaleMat.type());

        //Apply thresholding
        cv::threshold(grayscaleMat, binaryMat, 100, 255, cv::THRESH_BINARY);
        bitwise_not(binaryMat, grayscaleMat);
        //Show the results
        Mat zeroRow(30, grayscaleMat.cols, CV_8UC1, Scalar(0));
        Mat zeroCol(grayscaleMat.rows + 60, 30, CV_8UC1, Scalar(0));
        //extend image
        vconcat(grayscaleMat, zeroRow, grayscaleMat);
        vconcat(zeroRow, grayscaleMat, grayscaleMat);
        hconcat(grayscaleMat, zeroCol, grayscaleMat);
        hconcat(zeroCol, grayscaleMat, grayscaleMat);

        cv::imwrite(source + "\\EM\\" + filename + "_em_result.jpg", grayscaleMat);
}
int main(int argc, const char * argv[])
{
        String source = argv[1];
        Size imsize(IMGCOLS, IMGROWS);
```

```
path p (argv[1]);
int i, j, k, l;
directory_iterator it{ p };
char choice = ' ';
ofstream myfile;
boost::filesystem::path dir(source + "\\EM");
while (toupper(choice) != 'O' || toupper(choice) != 'A'){
        if (boost::filesystem::exists("features")){
                cout << "features.txt already exists. Do you want to overwite (o) or
                     append (a) \"features\"? [o/a]: ";
                cin >> choice;
                if (toupper(choice) == 'O'){
                        myfile.open("features", ios::trunc);
                        break;
                }
                else if (toupper(choice) == 'A'){
                        myfile.open("features", ios::app);
                        break;

                }
        }
        else{
                myfile.open("features");
                break;
        }
}
if (boost::filesystem::create_directory(dir)){
        cout << "Created EM directory" << "\n\n";
}
while (it != directory_iterator{}){
        String filename = boost::filesystem::basename(*it);
        cout << filename << extension(*it) << endl;
        if (extension(*it) != ".jpg"){
                *it++;
                continue;
        }
        else{
                cv::Mat loadedimage = cv::imread(source + "\\" + filename + ".jpg");
                cv::Mat image;
                cv::Mat brightImage;
                resize(loadedimage, image, imsize);
                cv::imwrite(source + "\\EM\\" + filename + "_resize.jpg", image);
                image.convertTo(brightImage, -1, 0.5, 80);
                cv::imwrite(source + "\\EM\\" + filename + "_bright.jpg", brightImage);
                assert(brightImage.type() == CV_8UC3);
                const int image_rows = brightImage.rows;
                const int image_cols = brightImage.cols;
                vector <Mat> newMatrix;
                cv::Mat hsvImage, newImage;
                vector <Mat> channels, newChannels;
                cvtColor(brightImage, hsvImage, CV_BGR2HSV);
                split(hsvImage, channels);
                newChannels.push_back(channels[1]);
                newChannels.push_back(channels[2]);
                merge(newChannels, newImage);


                newImage.convertTo(newImage, CV_8UC1);

                int dimension = 2;

                cv::Mat reshaped_image = newImage.reshape(1, image_rows * image_cols);
                assert(reshaped_image.type() == CV_8UC1);
                assert(reshaped_image.rows == image_rows * image_cols);
                assert(reshaped_image.cols == dimension);
                // create an input for the EM Algorithm
                cv::Mat samples;
                reshaped_image.convertTo(samples, CV_64FC1, 1.0 / 255.0);
                assert(samples.type() == CV_64FC1);
                assert(samples.rows == image_rows * image_cols);
                assert(samples.cols == dimension);

                const int cluster_num{ 2 };
                cv::EM model{ cluster_num };

                // prepare outputs
                cv::Mat labels;
                cv::Mat probs;
                cv::Mat log_likelihoods;

                // execute EM Algorithm
                cout << "Executing EM Algorithm for " << filename << "..." << endl;
                model.train(samples, log_likelihoods, labels, probs);

                assert(log_likelihoods.type() == CV_64FC1);
                assert(log_likelihoods.rows == image_rows * image_cols);
                assert(log_likelihoods.cols == 1);

                assert(labels.type() == CV_32SC1);
                assert(labels.rows == image_rows * image_cols);
                assert(labels.cols == 1);
                cout << "Observing Probabilities" << endl;
```

```
assert(probs.type() == CV_64FC1);
assert(probs.rows == image_rows * image_cols);
assert(probs.cols == cluster_num);
observe_probs(probs);
cout << "Observing Labels and Means" << endl;
const cv::Mat& means = model.get<cv::Mat>("means");
assert(means.type() == CV_64FC1);
assert(means.rows == cluster_num);
assert(means.cols == dimension);
observe_labels_and_means(labels, means, image_rows, image_cols, filename,
     source);
cout << "Observing Weights" << endl;
const cv::Mat& weights = model.get<cv::Mat>("weights");
assert(weights.type() == CV_64FC1);
assert(weights.rows == 1);
assert(weights.cols == cluster_num);
observe_weights(weights);

//Getting contours
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
cv::Mat grayscaleImage = cv::imread(source + "\\EM\\" + filename + "
     _em_result.jpg");
cv::Mat newgrayscaleImage = cv::imread(source + "\\EM\\" + filename + "
     _em_result.jpg");
assert(grayscaleImage.type() == CV_8UC1);
cv::cvtColor(grayscaleImage, newgrayscaleImage, CV_BGR2GRAY);
Canny(grayscaleImage, newgrayscaleImage, 100, 1500, 3);
imwrite(source + "\\Canny\\" + filename + "_canny.jpg", newgrayscaleImage)
     ;

Scalar colour;
cv::Mat cannyimage = newgrayscaleImage;
cv::Mat EMimage = cv::imread(source + "\\EM\\" + filename + "_em_result.
     jpg");
vector<Point> edge;
for (j = 0; j < cannyimage.rows; j++){
        for (i = 0; i < cannyimage.cols; i++){
                if (cannyimage.at<uchar>(j, i) == 255){
                        edge.push_back(Point(i, j));
                }
        }
}
float radius;
Mat cropped;
float whites=0.0;
cv::Mat curvatureImage;
Rect r;
Mat roi;
Mat mask;
int area, zerocounter;
bool set = false;


cout << "Calculating areas..." << endl;
Mat colors(25,1,CV_8UC1,Scalar(0));
for (i = 0; i < edge.size(); i++){
        j = 0;
        zerocounter = 0;
        for (radius = 5.0; radius < 30.0;){

                //initialize region of interest
                r.x = edge[i].x - radius;
                r.y = edge[i].y - radius;
                r.width = radius * 2;
                r.height = radius * 2;
                roi  = Mat(EMimage, r);
                mask = Mat(roi.size(), roi.type(), Scalar::all(0));
                circle(mask, Point(radius, radius), radius, Scalar::all
                     (255), -1);
                cropped = roi & mask;

                //calculate white points
                whites = 0;
                for (k = 0; k < cropped.rows; k++){
                        for (l = 0; l < cropped.cols; l++){
                                if (cropped.at<uchar>(l, k) == 255){
                                        whites++;
                                }
                        }
                }
                //get area
                unsigned char area = uchar((whites * 255) / (PI * radius *
                      radius));
                colors.at<uchar>(0, j) = area;
                radius = radius + 1;
                j++;
                if ((int)area == 0)
                        zerocounter = zerocounter + 1;
        }
        if (!set && zerocounter < CURVATURE_ROWS){
                curvatureImage = colors;
```

```
                            set = true;
                        }
                        else if (set && zerocounter < CURVATURE_ROWS){
                            hconcat(curvatureImage, colors, curvatureImage);
                        }
                    }
                    imwrite(source + "\\EM\\" + filename + "_curvature_image.jpg",
                        curvatureImage);


                    int bins = 16;
                    float range[] = { 0, 256 };
                    const float* histRange = { range };
                    vector<Mat> histograms;
                    vector<float> features;
                    Mat histogram;
                    int ch[] = { 0 };
                    for (i = 0; i < curvatureImage.rows; i++){
                        stringstream ss;
                        ss << i;
                        calcHist(&curvatureImage.row(i), 1, 0, Mat(), histogram, 1, &bins,
                            &histRange, true, false);
                        for (j = 0; j < histogram.rows; j++){
                            features.push_back(histogram.at<float>(j, 0));
                        }
                    }
                    int leafnumber;
                    cout << "Please enter the number corresponding to " << filename << ".jpg (
                        refer to the list in leaflist.exe):";
                    cin >> leafnumber;
                    myfile << leafnumber << " ";
                    for (i = 0; i < features.size(); i++){
                        myfile << i + 1 << ":" << features[i] << " ";
                    }
                    myfile << "\n";
                    *it++;
            }
        }
        return 0;
}
```

source–code/training/keytosci.cpp

```
#include "stdafx.h"
#include <numeric>
#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <sstream>
#include "strtk.hpp"
using namespace std;

int main(int argc, const char * argv[]){
        string leafSciname, leafindexstr, line, temp;
        stringstream ss;
        fstream filename, numListFile;
        map<int, string> leafmap;
        map<int, string>::iterator it;
        int leafindex;
        filename.open("leafNumSciList.txt", ios::in);
        while (getline(filename, line)){
                vector<string> vec;
                strtk::parse(line, ":", vec);
                leafindexstr = vec[0];
                leafindex = std::stoi(leafindexstr);
                leafSciname = vec[1];
                leafmap[leafindex] = leafSciname;
        }
        filename.close();
        int choice = 0;
        do{
                cout << "Leaf Keys and Scientfic Names Listing" << endl;
                cout << "[1] Add Entry" << endl;
                cout << "[2] Search Leaf" << endl;
                cout << "[3] List All" << endl;
                cout << "[4] Delete Entry" << endl;
                cout << "[5] Exit" << endl;
                cout << "Choice: ";
                getline(cin, temp);
                stringstream ss(temp);
                ss >> choice;
                switch (choice){
                case 1:
                        cout << "Enter the leaf's scientific name: ";
                        getline(cin, leafSciname);
                        cout << "Enter the leaf's number: ";
                        getline(cin, leafindexstr);
                        leafindex = std::stoi(leafindexstr);
                        cout << leafindex;
                        leafmap[leafindex] = leafSciname;
                        cout << "[INFO] Leaf added successfully." << endl << endl;
                        break;
                case 2:
                        cout << "[SEARCH] Enter the leaf's number: ";
                        getline(cin, leafindexstr);
                        leafindex = std::stoi(leafindexstr);
                        it = leafmap.begin();
                        it = leafmap.find(leafindex);
                        if (it != leafmap.end()){
                                cout << leafindex << ": " << it->second << endl << endl;
                        }
                        else
                                cout << "[ERROR] Leaf does not exist" << endl << endl;
                        break;
                case 3:
                        cout << "List of Leaves:" << endl;
                        for (it = leafmap.begin(); it != leafmap.end(); it++){
                                cout << it->first << ": " << it->second << endl;
                        }
                        cout << endl;
                        break;
                case 4:
                        cout << "[DELETE] Enter the leaf's name: ";
                        getline(cin, leafindexstr);
                        it = leafmap.begin();
                        if (it != leafmap.end()){
                                ss << leafindexstr;
                                ss >> leafindex;
                                leafmap.erase(leafindex);
                                cout << "[INFO] Record deleted successfully." << endl << endl;
                        }
                        else
                                cout << "[ERROR] Leaf does not exist" << endl << endl;
                        break;
                case 5:
                        break;
                default:
                        cout << "[ERROR] Invalid choice!" << endl << endl;
                }

        } while (choice != 5);
        filename.open("leafNumSciList.txt", ios::out | ios::trunc);
        numListFile.open("leafNumList.txt", ios::out | ios::trunc);
        for (it = leafmap.begin(); it != leafmap.end(); it++){
                filename << it->first << ":" << it->second << "\n";
                numListFile << it->first << "\n";
        }
        filename.close();
        numListFile.close();
        return 0;
}
```

source–code/training/scitocomm.cpp

```
#include "stdafx.h"
#include <numeric>
#include <iostream>
#include <fstream>
#include <string>
#include <map>
#include <sstream>
#include "strtk.hpp"
using namespace std;

int main(int argc, const char * argv[]){
        string leafSciname, leafCommonname, line, temp;
        fstream filename;
        map<string, string> leafmap;
        map<string, string>::iterator it;
        int leafindex, lastindex;
        char delim;
        filename.open("leafSciCommonList.txt", ios::in);
        while (getline(filename,line)){
                vector<string> vec;
                strtk::parse(line, ":", vec);
                leafSciname = vec[0];
                leafCommonname = vec[1];
                leafmap[leafSciname] = leafCommonname;
        }
        filename.close();
        int choice = 0;
        do{
                cout << "Leaf Scientific And Common Names Listing" << endl;
                cout << "[1] Add Entry" << endl;
                cout << "[2] Search Leaf" << endl;
                cout << "[3] List All" << endl;
                cout << "[4] Delete Entry" << endl;
                cout << "[5] Exit" << endl;
                cout << "Choice: ";
                getline(cin, temp);
                stringstream ss(temp);
                ss >> choice;
                switch (choice){
                case 1:
                        cout << "Enter the leaf's scientific name: ";
                        getline(cin, leafSciname);
                        cout << "Enter the leaf's common name: ";
                        getline(cin, leafCommonname);
                        leafmap[leafSciname] = leafCommonname;
                        cout << "[INFO]Leaf added successfully." << endl << endl;
                        break;
                case 2:
                        cout << "[SEARCH] Enter the leaf's name: ";
                        getline(cin, leafSciname);
                        it = leafmap.begin();
                        it = leafmap.find(leafSciname);
                        if (it != leafmap.end())
                                cout << leafSciname << ": " << it->second << endl << endl;
                        else
                                cout << "[ERROR]Leaf does not exist" << endl << endl;
                        break;
                case 3:
                        cout << "List of Leaves:" << endl;
                        for (it = leafmap.begin(); it != leafmap.end(); it++){
                                cout << it->first << ": " << it->second << endl;
                        }
                        cout << endl;
                        break;
                case 4:
                        cout << "[DELETE] Enter the leaf's name: ";
                        getline(cin, leafSciname);
                        it = leafmap.begin();
                        if (it != leafmap.end()){
                                leafmap.erase(leafSciname);
                                cout << "[INFO]Record deleted successfully." << endl << endl;
                        }
                        else
                                cout << "[ERROR]Leaf does not exist" << endl << endl;
                        break;
                case 5:
                        break;
                default:
                        cout << "[ERROR]Invalid choice!" << endl << endl;
                }

        } while (choice != 5);
        filename.open("leafSciCommonList.txt", ios::out | ios::trunc);
        for (it = leafmap.begin(); it != leafmap.end(); it++){
                filename << it->first << ":" << it->second << "\n";
        }
        filename.close();
        return 0;
}
```

XI. Acknowledgment

"Now unto Him that is able to keep you from falling, and to present you faultless before the presence of His glory with exceeding joy, to the only wise God our Saviour, be glory and majesty, dominion and power, both now and ever. Amen." - Jude 24-25 (KJV)

Finally, after 4+1 years of labor, every course has been conquered and I am now going to graduate. This manuscript is the mark of victory, signifying I have completed every requirement for B.S. Computer Science. Of course, I wouldn't have reached this point without falling not a few times, but in those times, I stand back up a nd push on and I couldn't have done it without God and the people who continuously support me. So let me take this time to thank each of them one-by-one.

First to our Almighty God and our Saviour Jesus Christ, if it wasn't for His grace and mercy. I couldn't have survived these years. I wouldn't have even entered UP Manila if it wasn't for Him! His grace carried me through the endless waves of requirements, and if it wasn't for Him, I would still be stuck doing my thesis (or possibly, one of those machine problems). His grace is more than enough. His power magnified in my weakness. His joy in me despite my pain. He's done so much that no mouth can express, nor pen could ever tell (or keyboard could type). And I could only bow down in reverent awe, worshiping and praising Him, for His love.

To my parents who was the instrument of God for providing my needs, encouraging me (and nagging me to finish my thesis). If it wasn't for them, I couldn't have gotten a degree or even entered college. Although they were busy in their work, they took time to check on the progress of my SP (even though I knew they couldn't understand the technical side of things).

To my sister for being very understanding and patiently enduring the noisy keyboard and mouse I have. Thank you for lending me your phone when I needed to test my application.

To the Student Group coworkers of Christian Gospel Center (both past and present), if it wasn't for your spiritual nuturing, I could've lost faith and fell along the way. Your consistent encouragement and kind corrections were instrumental to be who I am today. I thank God for all of you, being noble vessels in the house of God, that you continue to minister, not only to me, but to all students, that we may all be molded into the likeness of Christ, to be God-glorifying individuals and to be overcomers in these dark days. May the good Lord continue to bless you and use you in your ministry.

To the brothers and sisters at Christian Gospel Center, thank you for your prayers and your endless support. Your lives are an encouragement to me to continue on this pathway of faith. Your companionship in this walk has proven to be invaluable and this proves that we are all co-pilgrims. I pray that we may all continue to walk this path until we see Him and that I, too, may be an example to you and those following behind as you are examples to me. May we continue to encourage one another in love and unity. The Lord remembers your prayers and may He richly reward you.

To my SP advisers sir Co, Sir Bernie, and finally sir Marvin and to the project head sir Solano. Thank you sir Solano for the topic, although the plan didn't work out as planned, but I thank you for your patience (yes, it was delayed for a year, at least on my part). Thank you sir Co for choosing me to be part of this project. Thank you sir Bernie for helping me in my proposal and providing me the journals I needed. Thank you sir Marvin for guiding me in my final manuscript and in the development of my project. Thanks also for giving us an introductory crash course to Machine Learning and SVM, which some of us needed.

To the professors of U.P. Manila, thank you for the knowledge that you have imparted and the time. I know that a lot of our subjects were petitioned, and we thank you for agreeing to be our instructor. The knowledge imparted to me are invaluable as I plan on to pursue further studies. A special shoutout to Ate Eden, you may not be our instructor, but the moral support you have shown to our batch has been invaluable.

To block 12, thank you guys for the acquaintanceship. I know I haven't been to much of the activities, but, yeah, thanks.

Special shoutout to the following peeps: Jason Chua, Benedict Tiu, Jerahmeel Chua, John Bengemin Uy, and Aaron Uy. Thanks for being there for me. Thanks for the prayers, chats, jokes, encouragements, heart-to-hearts(?!), meals together, time together, and whatever else.

Shoutout also to the freshie batch of 2014, to B, A, S, K, D and your equivalent hexadecimal thingys. Thanks for the stress-reliever.

This marks the end of a chapter of my life, but also marks the beginning of a new one. May God continue to find me faithful in His service as I start my career.