Classification of plants using time series analysis algorithms

Αναγνώριση και κατηγοριοποίηση φυτών από εικόνες με χρήση καινοτόμων αλγορίθμων ανάλυσης χρονοσειρών

Submitted to University of the Aegean

in partial fulfillment of the requirements for the degree of

Information and Communication Systems Engineer

A Diploma thesis of

Maria Galani

October 2017

APPROVED BY:

Maragkoudakis Emmanouil, Supervisor

Associate Professor

Department of Information and

Communication Systems Engineering

Stamatatos Efstathios, Member

Associate Professor

Department of Information and

Communication Systems Engineering

Drosos Dimitrios, Member

Assistant Professor

Department of Information and

Communication Systems Engineering

UNIVERSITY OF THE AEGEAN

October 2017

ABSTRACT

The purpose of this diploma thesis is to determine the species of a plant from an image of its leaf, using a time series classification system. The outline of each leaf is described with a time series, which is then analyzed with time series analysis algorithms to classify the plant's class.

Plant identification and classification is a common interest in the field of computer vision and machine learning. Identifying a plant from an image can be challenging: the leaf might have its stem, or be on a branch, or the lighting might be insufficient.

Keywords: data mining, digital image processing, classification algorithms, time series analysis, plant classification

Maria Galani

Department of Information and Communication Systems Engineering

UNIVERSITY OF THE AEGEAN

TABLE OF CONTENTS

ABSTRACT 3
1 INTRODUCTION 7
1.1 Data mining and Machine learning 7
1.2 Computer vision and Digital Image Processing 8
1.3 Time series 8
1.4 Information Retrieval 9
2 RELATED WORK 10
2.1 Introduction 10
2.2 Flavia 10
2.2.1 Image preprocessing 10
2.2.2 Plant classification 11
2.3 Leafsnap 11
2.3.1 Image preprocessing 11
2.3.2 Species identification 12
2.4 Swedish leaf dataset and PRICoLBP 13
2.4.1 Leaf recognition 13
2.5 Symbolic representation of time series 13
2.5.1 SAX 14
2.5.2 Data mining using SAX 14
2.6 SAX-VSM 15
3 DIGITAL IMAGE PROCESSING 16
3.1 Introduction 16
3.2 Filtering 16
3.3 Image processing techniques 18
3.4 Morphological operations 19
3.5 Segmentation 20
3.6 Edge detection 20
4 TIME SERIES DATA MINING 21

4.1 Preprocessing time series data 21
4.1.1 One-against-All 21
4.1.2 Time series normalization 21
4.1.3 Dimensionality reduction techniques 22
4.2 Distance and similarity measures 24
4.2.1 Euclidean distance 24
4.2.2 Dynamic time warping 24
4.2.3 MINDIST 25
4.3 Data mining tasks 26
4.4 Data mining algorithms 27
4.4.1 k-nearest neighbor (kNN) 27
4.4.2 Support vector machines (SVM) 28
4.4.3 Decision trees 28
4.4.4 Random forests 29
4.4.5 Gradient boosting machines or gradient boosting/boosted trees 29
4.4.6 Naive Bayes 30
5 INFORMATION RETRIEVAL TECHNIQUES 32
5.1 Introduction 32
5.2 TF-IDF weighting 32
5.3 Other information retrieval operations 33
6 IMPLEMENTATION 35
6.1 The dataset 35
6.2 Image preprocessing 36
6.3 Time series processing 38
6.4 The classification models 44
6.4.1 Introduction 44
6.4.2 Classification with the time series data 44
6.4.3 Classification with the SAX strings 45
7 SUMMARY 47
8 FUTURE WORK 47

9 REFERENCES 48
APPENDIX A 51
APPENDIX B 53
APPENDIX C 56
APPENDIX D 62

1 INTRODUCTION

1.1 Data mining and Machine learning

In our days we use many applications on our devices and on the web, that use data mining, machine learning and information retrieval algorithms. From speech recognition to spam filtering, these two computer science fields play a big role in our daily interactions.

A machine learning algorithm aims to predict the answer to a problem from a set of data. Nowadays, these algorithms can make complex mathematical calculations on big data with good accuracy and efficiency. Many of our day-to-day activities are powered by machine learning algorithms like web search results, handwriting, sign language and image recognition, ads on web pages and mobile devices, weather forecasting, self-driving cars, speech translation.

Data mining and machine learning are two distinct fields. While machine learning models reaches a decision based on some knowledge or information, data mining uses statistics to discover information and/or patterns [1]. In general, machine learning can be described by the term *classification* and data mining by the term *prediction.* Classification is the method to classify new data, based on a training set with already known labels. Prediction is the method to predict the labels of new, unknown data [2].

The classification schema begins with gathering the training data which are processed by classification algorithms. Then, a prediction is performed on a testing dataset that is validated by the classifier. Finally, the classifier assigns labels to the testing dataset [2]. A classifier is a function that gives a label to an unlabeled instance using internal data structures [40].

The most widely adopted machine learning methods are supervised learning (e.g. classification) and unsupervised learning (e.g. clustering). In this thesis we use supervised learning. In these type of algorithms, we train our machine with labeled inputs, to indicate the class labels of the output data. We know in advance the class labels of the training data.

A classifier H, given a training set of examples with the form: (x_i, y_i), for $i = 1, \dots, N$, where x_i is the value of the data and y_i is the class label of the data, must to know how to predict the class label $Y_i \in \{1, \dots, K\}$ of a new example vector X_i, given only the new vector of variables and not its class [37, 33].

On the contrary, in unsupervised learning the testing set cannot be classified based on the training set because the class labels of the data are unknown [2].

Once we decide which our training dataset will be, we must prepare the data by preprocessing them. A training set can be images, time series, stocks, signals etc. So, the data must be preprocessed to reduce noise, extract features and be normalized. In our case, we have images with leaves that are converted into time series. When a dataset contains images, the preprocessing is done using computer vision.

1.2 Computer vision and Digital Image Processing

Applications that utilize computer vision and image processing algorithms are very common. Making images more beautiful with special filters, identifying the person or the object on an image, iris, facial expression, gesture and body movement recognition are all achieved with image processing.

Digital image processing also applies in many fields of science: Medicine, astronomy, industry, geology, microscopes, satellites [12]. Plants attract the researchers' interest because plant classification is a common problem. With computer vision and machine learning, the plants can be classified easily and rapidly. Botanists can identify species and detect plant diseases easier.

In general, computer vision algorithms extract specified information from images. Computer vision is the science that allows the people to understand and see things in images and videos that cannot see with the naked eye [3]. It is a branch of artificial intelligence science that overlaps with digital image processing and allows computers to imitate the human eye and draw conclusions.

There are three levels of processing:

A low-level process, that takes an image as input and produces an image as a result. This level contains fundamental image processing operations such as noise reduction, image sharpening, contrast enhancement.

A mid-level process, that receives an image and produces a set of attributes. An operation of this level can be object recognition or identification, and segmentation.

A high-level process, that takes these attributes and produces an output that "makes sense". This level contains operations like scene understanding and autonomous navigation.

The first two levels, low and mid, can be characterized as the image processing level and the last one, high-level, is the image analysis, the computer vision level.

1.3 Time series

Time series is a term that is used to describe a sequence of points through time. They are more often used in statistics, but also in fields like finance, signal processing, forecasting and many more. Time series can show how a company's stock increased or decreased through a period of time, describing a behavior or trend. In plant classification time series are often used to describe the outline of a leaf or a part of its stem. In most cases, researchers use more than just a time series to describe a leaf. They extract additional information that characterize each leaf's species in a better way.

1.4 Information Retrieval

The field of Information Retrieval (IR) deals with extracting information from data. Large amounts of information are stored every day, so it is important to be able to find knowledge in databases reliably and efficiently, in order to evaluate or answer a query. The most commonly used application of information retrieval in our everyday lives is in web search engines. Web search engines use IR techniques to find results on the web that match best with the user's question. An information retrieval system ensures that each document in a database has a description, which is used to evaluate the matching grade with the given question and show the corresponding result [10].

2 RELATED WORK

2.1 Introduction

This section is dedicated to other researches that deal with plant classification. We will provide an overview of their research and the information we used to compose this thesis. Any classification and image processing method that is referenced, will be explained in the next chapter.

2.2 Flavia

Flavia is a leaf recognition program and algorithm for plant classification using a probabilistic neural network [4]. We used the image dataset from Flavia and the algorithm they used to pre-process the leaf images. Their system can classify 32 plant species mostly from Asia and North America, with an average accuracy more than 90% for all of them. Their program starts by asking the user to input a leaf image of a plant acquired via digital camera or scanner. Then, the leaf image appears on the screen and the user is prompted to select the two terminals of the longest and main venation of the leaf by clicking the image. Then the computer will show from which plant species this leaf is.

2.2.1 Image preprocessing

At first, the acquired image is processed with an algorithm to produce the perimeter of the leaf as a black curve on a white background. The image is converted from RGB to grayscale. To convert the RGB value of a pixel into its grayscale they use the following equation

$$gray = 0.2989 * R + 0.5870 * G + 0.1140 * B \quad (\text{Eq. } 1)$$

Then, the image is made binary through the histograms of each value of the RGB model. The histograms of each RGB value are divided with the total number of leaves (3000). The image is thresholded at the value that is the average lowest point between the two peaks in every color's histogram (242), by replacing all pixels that have luminance greater than 0.95 (242/255=0.949) with the value 1 and the others with 0. Additionally, the 0 and 1 values get swapped in order to have a black curve on a white background. Next, a rectangular averaging filter of size 3 × 3 is applied for noise filtering. Then, the image undergoes boundary enhancement, to get the margin of the leaf, applying a Laplacian filter of the following 3 × 3 spatial mask:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

In the following procedure, the 5 basic features -diameter, physiological length, physiological width, leaf perimeter and leaf area- are extracted. The physiological length is the feature that user

selects interactively. Also, 12 digital morphological features (DMFs) are extracted such as smooth factor, form factor, aspect ratio, rectangularity, narrow factor, perimeter ratio of diameter, perimeter ratio of physiological length and physiological width and 5 vein features. Then, a statistical procedure called Principal Component Analysis (PCA) is used to reduce the dimensionality of the input matrix. PCA can achieve simplification, data reduction, outlier detection, classification, prediction on a data matrix [5]. In Flavia's case, PCA prepares the extracted features from the leaf image for a Probabilistic Neural Network (PNN).

2.2.2 Plant classification

"Neural networks are frequently employed to classify patterns based on learning from examples" [6]. PNN is a neural network that is used often for pattern recognition problems. Wu et al. [4] mention, "The network classifies input vector into a specific class because that class has the maximum probability to be correct". PNN is fast on training and has a simple structure.

In their research they used 1800 leaf images to train their system, most of them from China. They also used 10 images of each plant species for the testing set. They claim that the accuracy of their program can be increased by extracting more features from the leaf images.

2.3 Leafsnap

Leafsnap [7] is a mobile application that uses computer vision to automatically identify and recognize plant species. Their system can identify leaves efficiently, even if the background of the image contains non-leaf objects. Their dataset consists of 184 trees in the Northeastern United States.

2.3.1 Image preprocessing

Their first step is to make a classifier that tells if the object of the image is a leaf or not (leaf/no-leaf classification) and is a very important procedure, since some users can take a photo of a leaf which contains branches, other leaves or the lighting is not helpful to segment the image. To build the classifier they use a Support Vector Machine that uses an RBF kernel as a classification function. They also adjust the input image by resizing and rotating it. If the image cannot be used, the app informs the users that the input is invalid and gives instructions to take a better image. The classifier is very fast and helpful, so that in case of unsuitable input the user just retakes the picture and the system does not fail.

The system uses color-based segmentation. The writers of [7] claim that this approach is more effective and faster, compared to segmentation boundary methods that focus on the shape or other features of a leaf, because of the various leaf shapes; some leaves are complex (main leaf with leaflets) or they can be found in groups (e.g., pine needles). In other words, they segment the images separating the foreground from the background. It is very important that their method adapts to variances of the color and lighting of a leaf.

The segmentation starts with converting the RGB image to HSL (Hue, Saturation, Lightness), but only the saturation value is used. The segmentation of some leaves can be problematic and therefore they apply weights in the pixels of the image. In that way, when the pixels outside and inside of the region of interest have different weights, the segmentation has better results. Then, the image is thresholded and segmented with an Expectation Maximization (EM) algorithm that separates the foreground from the background. The result is an image with black background and white pixels that indicate the area of the leaf so there are two groups in an image, the leaf and the background. They then remove false positive regions which can be caused by irrelevant objects, shadows and uneven backgrounds. The following step contains the detachment of the stem of the leaf. If there is a stem in the picture, removing it must not alter the connected components in the background or in the leaf. Some of the segmented images contain "holes" in the leaf area or in the background, which are filled to avoid later complications.

They noticed that the most common errors in the segmentation of the leaf are when there are shadows in the background made by the leaf, or when there is too much light in the leaf area.

The next procedure is about the curvature of each leaf. They distinguish the leaves of four different species based on their curvature. They use two histograms of curvature that describe two different scales of an area of the leaf. The first measure is determined by "the fraction of the circle's area contained inside the object", where the center of this circle is a point in the leaf's outline. The second measure is "the fraction of the circle's perimeter contained inside the object". They use these measures to extract a curvature image from the segmented image. This curvature image "shows curvatures for each contour point along the x-axis and at different scales along the y-axis" and they use each row to compute histograms of curvature values to create a feature called Histograms of Curvature over Scale (HoCS). This shape retrieval approach has many advantages and guarantees better results.

2.3.2 Species identification

A nearest neighbor search is used in order to identify each leaf's species. The algorithm uses the HoCS feature as a query. Their database consists of HoCS extracted features from 23,915 lab images taken by high-quality camera and 5,192 field images captured by mobile devices. The field images are not "clean" thus containing different amounts of blur, light and they were captured from different viewpoints. It is very important that the search database contains real-world images because this is the key to effectiveness. They had several problems measuring species that have many different leaf shapes and their curvatures vary. In that case, the app shows to the user the top 25 matching species and the user does the identification. However, the best result is to show as few matching species as possible. The writers of [7] mention that "The correct match is within the top 5 results for 96.8% of queries using curvature histograms".

Currently, the team of Leafsnap has an application for mobile devices that run iOS. Since the launch of their app in May 2011, nearly a million users downloaded Leafsnap. They also are discussing for improvements and launching the application for Android.

2.4 Swedish leaf dataset and PRICoLBP

The writers of "*Pairwise Rotation Invariant Co-occurrence Local Binary Pattern"* [8] used their algorithm and eight datasets to create applications for texture classification, material recognition, flower recognition, leaf recognition, food recognition and scene classification. For the leaf recognition application, they used the Swedish leaf dataset that contains 15 plant species and is used very often to similar applications. It has 75 leaf images for each species, 25 of them are used for training and the rest of them for testing. The images are very clear and with high-quality. Some of them are slightly rotated. This research uses also shape-based methods. This can be challenging because firstly, a lot of species have very similar leaf shape, and secondly, some images are taken from a different viewpoint, so leaves from the same species may differ a lot.

The researchers of [8] indicate the importance of designing effective features. In their paper, they propose a transform invariance in spatial co-occurrence feature introducing their patent, the pairwise rotation invariant co-occurrence local binary pattern (PRICoLBP) feature.

2.4.1 Leaf recognition

PRICoLBP works as a descriptor in which the researchers "encode two LBP features collaboratively while preserving the relative orientation angle between them". The local binary pattern operator (LBP), is a descriptor that encodes texture information. As the writers explain: "For a pixel A in an image, its LBP code is computed by thresholding its circularly symmetric n neighbors in a circle of radius r with the pixel value of the central point and arranging the results as a binary string." Pairwise rotation invariance is based on Pairwise Transform Invariance (PTI). Given *A* and *B* that are two points in an image and each one has a determined orientation, the uniform pattern of B can be computed according to the orientation of A. Then, as a result they get two co-occurrence patterns. They used this descriptor and SVM to evaluate the Swedish leaf dataset and the performance of their system was significantly better than other researches. The descriptor PRICoLBP decreased the classification error by far, with the same classifier. Contrary to other shape-based descriptors, their descriptor PRICoLBP is insensitive to the inaccurate shape extraction caused by partial damage. That is a common problem in leaf images and in images that have noisy background.

2.5 Symbolic representation of time series

There are many ways to represent time series, the most common of which are the Discrete Fourier Transform (DFT), Piecewise Linear Approximation, Haar Wavelet and Adaptive Piecewise Constant Approximation [9]. There are representations that symbolize the time series data, but these methods do not allow lower bounding distance measures [9]. Keogh et al. [9] are introducing SAX (Symbolic Aggregate approXimation), their approach of symbolic representation of time series that allows the low-bounding feature.

2.5.1 SAX

Their procedure takes as input a raw time series and transforms it into the Piecewise Aggregate Approximation (PAA) representation, which is a dimensionality reduction technique proposed and introduced by Keogh et al. in [31]. Then, they symbolize the PAA representation into a discrete string. Their algorithm can offer dimensionality reduction due to the symbolic representation of the time series. Additionally, their approach guarantees lower bounding, which means, laconically, reducing the computational cost of a search.

At first, the writers of [9] explain the PAA dimensionality reduction: "to reduce the time series from *n* dimensions to *w* dimensions, the data is divided into *w* equal sized "frames". The mean (average) value of the data falling within a frame is calculated and a vector of these values becomes the data-reduced representation", and *w* as the number of PAA segments representing the time series. Secondly, they apply a transformation to achieve discretization, which means that each symbol will appear with equal probability. Then, based on the Euclidean distance, which is the most common distance measure for time series, they define a function that measure the distance between two SAX representations. This distance measure can be calculated by "looking up the distances between each pair of symbols, squaring them, summing them, taking the square root and finally multiplying by the square root of the compression rate" [9].

2.5.2 Data mining using SAX

Keogh et al. [9] tried SAX, implementing some clustering and classification methods and then comparing the results with the Euclidean distance and other proposed symbolic approaches.

The first clustering evaluation they present is hierarchical clustering. In this data mining method, the best distance measure is considered as the one which creates the most natural groups to the data. They state that "SAX is superior, since it correctly assigns each class to its own subtree" because of the "smoothing effect of dimensionality reduction". The results were very similar with those Euclidean distance produced on various datasets.

In the second method, they perform partitional clustering using the k-means algorithm. They claim that working with a symbolic approximation gives better results than working with the raw data.

They present the results using two classification methods. First, using the nearest neighbor classification, SAX beats the Euclidean distance and the other approaches once more. This success of SAX "is probably due to the smoothing effect of dimensionality reduction". Secondly, they perform decision tree classification comparing SAX to the Regression Tree (RT) and their results were competitive with those of the RT.

Additionally, the authors claim that their approach can reduce the numerosity of the time series data for some applications. Given that the most applications must deal with very long time series, they perform sliding window subsequence extraction to extract smaller parts of the SAX word and count, for example, how many times a substring appears in the original string. In that way the actual length of the string can be reduced by removing the duplicate substrings. Also, the occurrences of any removed substring or its position in the original string can be retrieved when

needed by sliding to the right and testing to see if the next window is also mapped to the same word until the word changes.

Summarizing, SAX is shown to be competitive with, if not superior to other representations. Keogh et al. [9] introduced "the first dimensionality reduction, lower bounding, streaming symbolic approach in the literature".

2.6 SAX-VSM

The writers of [11] introduce a time series classification method that is based on Symbolic Aggregate approximation (SAX), a symbolic representation of time series, and Vector Space Model (VSM), which uses TF∗IDF (TF: term frequency and IDF: inverse document frequency) weighting scheme. Vector space model is the representation of a set of documents as vectors in a common vector space [39]. They propose an alternative to nearest-neighbor algorithm (1NN) that covers the disadvantages of 1NN. SAX-VSM can learn from a small training set, has a low computational complexity and high understandability.

Their algorithm commences using SAX to transform the original time series data from the given training dataset into combined collections of words, referred to as bags of words. Each class corresponds to a bag of words. They use the sliding window method that is proposed in [9] to extract overlapping subsequences and convert them to SAX words. To obtain the bag of words representation of the original time series, they are placing the SAX words in a collection unordered.

Then, the algorithm continues transforming these collections of words into class-characteristic vectors using the TF∗IDF weighting scheme. Applying the TF∗IDF scheme, the result is a TF∗IDF weight matrix, whose rows correspond to all SAX words found in all classes and each column indicates a class of the training dataset. This matrix is usually sparse, because the SAX words of one class do often not appear in other classes.

During the next step, the algorithm transforms the unlabeled time series into SAX words and then into a term frequency vector, as in the training phase. Using cosine similarity, SAX-VSM computes the values between the term frequency vector of the unlabeled data and the TF∗IDF weight vectors. The class of the unlabeled time series is the one whose vector yields the maximal cosine similarity value.

The algorithm was tested in various datasets in order to discover the best corresponding patterns. The authors present the advantages and superiority of their approach and their results on [11] by applying SAX-VSM in various datasets that contain time series. They show how their algorithm had discovered the best characteristic subsequences from each given time series dataset. The discovered patterns aligned exactly with previous work on each dataset.

3 DIGITAL IMAGE PROCESSING

3.1 Introduction

In image processing, an image can be defined mathematically by the two-dimensional function *f(x, y)*, where x and y are the two spatial coordinates. A digital image has finite number of elements which have a specified position and intensity. These picture elements are defined as pixels [12]. When an image is acquired from a digital camera, waves/signals are transferred from one part of the system to the other, and considering this, in image processing the image is considered a signal. There are colored, grayscale and binary images. A grayscale image has intensities that correspond to different levels of gray. A binary image has only two intensity values, one and zero. A colored image can have various formats. A commonly used format for colored images is RGB (red, green, blue) which has three vectors, one for each color component, and each pixel of the image is represented by three intensity values, one for each color. Another similar format is CMY (cyan, magenta, yellow) that is used mostly in printers. People do not describe a color shade based on the percentages of the fundamentals colors (red, green, blue) but referring to its hue, saturation and intensity. That's why the HIS (or also called HSL by hue, saturation, luminance) format is a very helpful color model in image processing.

A lot of popular applications (like social media and messaging applications) use color transformations and tone corrections to improve the appearance of an image. Some color transformations can be achieved by processing the histogram of the image for image enhancement and segmentation. A histogram is a graph that shows how many pixels of the image correspond to each different intensity value found in that image. With histogram equalization in RGB or HSI color space we can achieve darker or brighter images, with high or low contrast [12].

3.2 Filtering

In signal processing we filter signals to remove unwanted frequencies. A filter can let low values of frequency to pass and block frequencies above a specified value, or vice versa. In image processing, filtering can be performed in the frequency domain or in the spatial domain. Filters can be used to remove noise and smooth or blur images. Noise is very common in images, due to various random information that are produced when the image is captured. There are different types of noise, such as Gaussian, salt-and-pepper, Poisson and speckle [13]. In spatial domain, we move the filter from point to point in an image, usually from one "neighborhood" of a certain size to another. Additionally, an average image can be found to reduce noise of the original image [12]. Image averaging is very important in astronomy since the insufficient light causes noise in the captured images, rendering them useless for analysis.

A smoothing spatial filter, also called averaging or low-pass filter, can be applied to remove small details in an image, or extract objects (blurring) and bridge gaps in line or curves using filter masks. A sharpening spatial filter, or high-pass filter can help to enhance or highlight a detail of the image

that has been blurred. Image sharpening is applied to medical imaging, electronic printing, industrial inspection [12]. Additionally, there are spatial linear and non-linear filters that can be implemented to reduce noise, but in general, linear filters are used to suppress noise [13].

To sharpen an image, one can use isotropic filters, "a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation" as the authors of [12] says. The simplest isotropic derivative linear operator is the Laplacian and applying it in an image can increase the contrast at some locations of the image and enhance small details. For enhancement, the first derivatives can also be implemented using the magnitude of the gradient. The gradient is used often in industrial inspection for defects detection.

Filtering in spatial domain can also be implemented with the mean, median and the Wiener filter. Mean filter, or average, is a linear filter that replaces the values of the pixels of a neighborhood with the mean value of the pixels of the neighborhood. Median filter is a non-linear filter that is most often used for noise reduction and edge detection. It is very effective in the presence of salt-and-pepper noise [12]. Median first sorts the values of each neighborhood numerically and replaces all the pixels with the middle value of the sorted pixels [13].

In frequency domain, smoothing can be implemented with three types of low-pass filters: the ideal low-pass filter, the Butterworth low-pass filter and the Gaussian low-pass filter. Similarly, there are three types of sharpening frequency domain (high-pass) filters: the ideal, the Butterworth and the Gaussian high-pass filters. Laplacian can also be implemented in frequency domain and the results are identical [12].

In frequency domain, one-dimensional or two-dimensional Fourier Transform and its Inverse can also be used to process images. The Fourier transform *F(u)* of a continuous function *f(x)* of one variable, is defined by the equation:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux}\, dx \quad \text{(Eq. 2)}$$

where, $j = \sqrt{-1}$. Given the *F(u)*, we can obtain the *f(x)* by means of the inverse Fourier transform that is given by the equation

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux}\, du \quad \text{(Eq. 3)}$$

The discrete Fourier transform (DFT) is also significant. The Fourier transform of a discrete function of one variable, *f(x)*, where $x = 0,1,2,\ldots,M-1$, is defined by the equation

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-j2\pi ux/M} \qquad \text{for } u = 0,1,2,\ldots,M-1. \quad \text{(Eq. 4)}$$

Similarly, we can obtain *f(x)* given *F(u)*, using the inverse DFT

$$f(x) = \frac{1}{M}\sum_{u=0}^{M-1} F(u)e^{j2\pi ux/M} \quad \text{for } x = 0,1,2,\dots,M-1. \quad \text{(Eq. 5)}$$

The two-dimensional Fourier transform of a discrete function of two variables is defined as

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{uy}{N})} \quad \text{(Eq. 6)}$$

And its inverse as

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{ux}{M}+\frac{uy}{N})} \quad \text{(Eq. 7)}$$

DFT has become a significant procedure in signal and image processing because of the Fast Fourier Transform (FFT), which has a low computational cost compared to DFT and other operations [12].

Information extraction from images can also be performed with correlation and convolution. Both are operations in which each output pixel is the weighted sum of neighboring input pixels. More specifically, correlation is an operation in which, by sliding a filter in the image to have a different center element each time, it multiplies the corresponding pixels with the filter components (the weights in the correlation kernel). In the end, it sums the individual products. The difference between these two operations is that, in convolution the convolution kernel (the filter) is rotated 180 degrees.

The discrete convolution of two functions $f(x,y)$ and $h(x,y)$ of size M × N is denoted by $f(x,y) * h(x,y)$ and is defined by the expression

$$f(x,y) * h(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n) \quad \text{(Eq. 8)}$$

The correlation of two functions $f(x,y)$ and $h(x,y)$ is defined as

$$f(x,y) \circ h(x,y) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)h(x-m,y-n) \quad \text{(Eq. 9)}$$

3.3 Image processing techniques

In this chapter, the image processing techniques and operations that were applied in implementation will be presented. MATLAB was used as computing environment, so some MATLAB functions and operations will be analyzed.

An image can be rotated and resized in order to serve the needs of the researcher and improve the visual appearance of an image [17]. A rotation operator "performs a geometric transform which

maps a position *(x1, y1)* of a picture element in an input image onto a position *(x2, y2)* in an output image by rotating it through a user-specified angle θ about an origin *O*" as has been mentioned in [17].

In MATLAB, there is a function that can convert the RGB values of an image to corresponding grayscale values by forming a weighted sum of the R, G and B components same as in Eq. 1: $0.2989 * R + 0.5870 * G + 0.1140 * B$

These coefficients are identical to those used to calculate luminance (E'y) in Recommendation ITU-R BT.601-7 [14] using this formula:

$$0.299 * R + 0.587 * G + 0.114 * B \quad (\text{Eq. } 10)$$

Most applications that implement a similar procedure, use the above equation to convert an RGB image to grayscale.

To convert a grayscale image to binary, one can use thresholding. MATLAB has a function that thresholds the input image using either Otsu's method [15] or using real-time adaptive thresholding [16]. A threshold is a specific value that is used as comparison to check if the given value exceeds it or not. The purpose of thresholding in image processing is to classify the pixels of an image as "dark" or "light" as is mentioned in [16]. Thresholding provides an easy way to separate regions of interest in an image from the background (segmentation) [17].

3.4 Morphological operations

Very common in image processing is the utilization of morphological operations. The morphological operators often take as input a binary image and a structuring element, and process objects in the image based on characteristics of the shape of the element. They can also brighten the image or even reduce noise. The final output is another binary image [17]. There are many morphological operations, like erosion, dilation, closing, opening, thinning and thickening [18].

Dilation is the morphological operation that enlarges regions in an image and it is mostly used to fill holes within those regions.

Erosion is used to shrink regions of an image, so the boundaries erode and the holes in those regions are enlarged.

Opening works like erosion and is used to remove boundary pixels from the edges of regions, thus it preserves the foreground regions of interest and eliminates all the other regions of foreground.

Closing, on the contrary, is like dilation and enlarges the boundary regions in an image, thus it preserves the background regions of interest while eliminating all the other background regions.

Another morphological operator is Hit-and-Miss Transform, which takes as inputs a binary image and a structuring element and is used to look for the pattern of the structuring element in the input image.

Thinning is mostly used for edge detection and is related to the hit-and-miss transform. It takes as input a structuring element and tries to translate and match this element in the image to remove foreground pixels.

Thickening has as output an image that contains the input image and any additional foreground pixels determines by the input structuring element [18].

3.5 Segmentation

Segmentation, as was mentioned above, is the operation that separates regions in an image. This operation is very significant in image processing since many applications require the extraction or identification of objects that the image contains. In plant classification specifically, it is needed to extract objects, the leaf or the background of the image for example, to use them then in classification methods. Segmentation can be implemented using intensity thresholding [17], separating the foreground of an image based on the foreground intensity values that are different from the intensity values of the background. This kind of procedure, is effective mostly in images that have a distinct foreground and background. Put differently, when the histogram of the image has well defined peaks, the segmentation using thresholding can have good results. If this is not the case, a simple thresholding cannot guarantee a good segmentation and adaptive thresholding may be a better solution [17].

3.6 Edge detection

Edge detection is also a very significant operation in segmentation and image processing in general. Edges can be identified by the high intensity contrast in an image. Representing an image based on its edges can remarkably reduce the amount of data in the image, without losing important information. Due to the high intensity values of the edges, calculating the first and second derivatives of the image can highlight them and become more clear and distinct. The gradient edge detection is widely used and there are several kernels (also known as masks or small matrices) used as gradient edge detectors such as the Sobel, Roberts and Prewitt operators [17].

4 TIME SERIES DATA MINING

4.1 Preprocessing time series data

Before we begin processing our dataset, we should define our type of classification problem. It is easy to classify data into one or two classes (binary classification), but when the dataset is categorized into more than two classes, we have to deal with multiclass classification.

4.1.1 One-against-All

Multiclass classification is a common problem in data mining and machine learning. The most common solution to this problem is the one-versus-all (OvA, or one-against-all) approach [25]. One can simply classify among *K* binary classes instead of *K* classes. Specifically, for *K* classifiers, the k^{th} classifier is trained with the its samples as positive samples and the other $K - 1$ classifiers are trained with their samples as negative samples [26]. When testing an unknown example, the classifier that produces the maximum score wins and the example is characterized with the predicted, corresponding class label [26]. In other words, one-against-all method divides our problem into several sub-problems, in order to manage them methodically and to implement the preferred methods more easily. Implementing this technique in a multiclass classification can improve the results of a classification (accuracy, recall, precision) and reduce the program's runtime.

4.1.2 Time series normalization

To classify time series, we need to measure the distance and similarity between them with the preferred measurements, and to do so we must first preprocess the time series. Sometimes, two time series can be identical, with similar peaks and shapes but can have different offsets in the Y-axis, which can be a problem when measuring the distance between them [19]. The time series may seem the same, but the distance between them will show that they are very different. Therefore, time series normalization is a significant step in their analysis. The two time series must be shifted to the same offset and the distance will reveal the true similarity between them [19].

In this thesis, Z-score normalization was implemented in the raw time series data. Shalabi et al. [21] compared three normalization methods to preprocess data that will be used for data mining. Z-score normalization datasets gave one of the minimum number of leaf nodes (simplicity), high accuracy and short training time (tree growing time), but the preferred one was min-max normalization. Z-score normalization was also proposed as normalization technique in time series by Vlachos in [22] and by Senin and Malinchik in [11]. In z-normalization the values of a time series *Q* of length *N* are normalized based on the mean and standard deviation of *Q*. A value *x* of *Q* is normalized to *x'* by the equation:

$$x_i' = \frac{x_i - \bar{Q}}{\sigma_Q} \qquad \text{for } i = 1,2,\dots,N \quad \text{(Eq. 11)}$$

where $\bar{Q}$ is the mean value of the time series *Q* and σ_Q is the standard deviation of *Q*.

4.1.3 Dimensionality reduction techniques

Dimensionality reduction is also a very important for a dataset or database of time series due to high dimensionality of this type of data. For this reason, it is recommended to implement representation (decomposition) techniques that can reduce the dimensionality, thus the computational and storage cost, without losing significant information that describe the data.

Time series can be decomposed into sum of sine waves (coefficients which are the discrete Fourier transform of the time series) using Fourier analysis [20, 22]. This way we gain space (information reduction and compression), energy, lower computational cost and we can achieve a smoothed out result [22]. The Discrete Fourier transform (DFT) is a very common representation technique. The equation of DFT and its inverse have been mentioned above. Given the number of the coefficients we choose to compress the data, the loss of information is proportional.

First, we implement DFT (or FFT) in the original data and then decide how many coefficients of the transform we will keep. When we decompose a time series choosing a small number of coefficients (e.g. 2 or 5), the output time series is a curve that does not carry much information and is not similar with the original time series. On the contrary, using 20 or 40 coefficients, the output time series is very similar to the original, without losing important information. After choosing the desired number of coefficients, the transform must be reconstructed using the Inverse Discrete Fourier transform (IDFT) in order to have the final result.

Another dimensionality reduction technique is Singular Value Decomposition (SVD) and this procedure was implemented in this thesis to reduce the dimensionality of the matrix that contains the raw time series data. SVD is a global transformation. Specifically, this technique examines the entire data set and then rotates it in a way that the first axis has the maximum possible variance, the second axis has the maximum possible variance orthogonal to the first axis, the third axis has the maximum possible variance orthogonal to the first two axis, etc. [23]

Using linear algebra, a real matrix *A* can be factorized into a product of three matrices. Given an *m* x *n* matrix *X* and a rank *r* (the number of linearly independent rows or columns), the equation for singular value decomposition of *X* according to [24] is the following:

$$X = USV^T \quad \text{(Eq. 12)}$$

where *U* is an *m* x *n* matrix, *S* is an *n* x *n* matrix and V^T is an *n* x *n* matrix. The columns of *U* are called the left singular vectors, $\{u_k\}$, and form an orthonormal basis, so that $u_i \cdot u_j = 1$ when i = j, and $u_i \cdot u_j = 0$ otherwise. The rows of V^T are called the right singular vectors, $\{v_k\}$, and form an orthonormal basis for the gene transcriptional responses. The elements of S are only nonzero on the diagonal, and are called the singular values. For dimensionality reduction, we are interested in the g_i transcriptional response gene. The SVD equation is a linear combination of the eigengenes $\{v_k\}$ for g_i and is defined by

$$g_i = \sum_{k=1}^{r} u_{ik} s_k v_k \quad \text{for } i = 1, \dots, m. \quad \text{(Eq. 13)}$$

The i^{th} row of *U*, $g_i{}'$ contains the coordinates of the i^{th} gene in the coordinate system (basis) of the scaled eigengenes, $s_k v_k$. If $r < n$, the transcriptional responses of the genes may be captured using a reduced number of variables, using gene $g_i{}'$ instead of g_i. This procedure can be implemented for dimensionality reduction [24].

In this thesis the data science software platform RapidMiner was used to implement some procedures and operations. RapidMiner has an SVD operator which can remove unnecessary attributes that are linearly dependent.

Lin et al. [31] proposed a promising approach (SAX) for representing time series symbolically that allows dimensionality reduction, numerosity reduction and lower bounding [31]. SAX can reduce a time series of arbitrary length *n* to a string of arbitrary length *w*, with $w < n$ (typically, $w \ll n$). The strings have a certain alphabet and it is of arbitrary length *a*, where $a > 2$.

SAX uses an intermediate representation called Piecewise Aggregate Approximation (PAA) between the raw time series data and the strings. The data are transformed into the PAA by the following equation:

$$\overline{c_i} = \frac{w}{n} \sum_{j=\frac{n}{w}(i-1)+1}^{\frac{n}{w}i} c_j \quad \text{(Eq. 14)}$$

where a time series *C* of length *n* is represented by the vector $\bar{C} = \overline{c_1}, \ldots, \overline{c_w}$ of length *w*. The above equation calculates the i^{th} element of $\bar{C}$. The dimensionality reduction is done by dividing the original time series into *w* equal sized "frames" (PAA coefficients). Particularly, "the mean value of the data falling within a frame is calculated and a vector of these values becomes the data-reduced representation" as Lin et al. [31] explain. Additionally, their algorithm contains a z-normalization of the time series before converting the data to the PAA representation, to have a mean of 0 and a standard deviation of 1. Once the time series have been normalized, they have a Gaussian distribution and are approximately normal. That can be assured with a normal probability plot that indicates an approximate straight line. We need to have an approximately normal time series so that the algorithm can produce equally probable symbols and determine the "breakpoints" (a sorted list of numbers) that will produce *a* areas with equal size under a Gaussian curve. Specifically, the breakpoints $B = \beta_1, \beta_2, \ldots, \beta_{a-1}$ in the area under a normal Gaussian curve from β_i to β_{i+1} to be equal to 1/*a*, with β_0 and β_a defined as -∞ and ∞, respectively [31]. After having obtained the PAA coefficients and the breakpoints, they compare each coefficient with the breakpoints to turn them into symbols. That is, "all PAA coefficients that are below the smallest breakpoint are mapped to the symbol "a", all PAA coefficients that are greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol "b", etc." as explained in [31]. Obtaining the PAA approximation $\bar{C}$ to a word $\hat{C}$, described in an equation:

$$\widehat{c_i} = alpha_j \quad iff \quad \beta_{j-1} \leq \overline{c_i} < \beta_j \quad \text{(Eq. 15)}$$

where $alpha_j$ the j^{th} element of the alphabet and the word $\hat{C} = \widehat{c_1}, \ldots, \widehat{c_w}$.

4.2 Distance and similarity measures

Once the dataset has been processed and analyzed, we must define a distance measure we want to implement, to compare the time series data. This way, we can then proceed to the classification and mining of the time series.

4.2.1 Euclidean distance

As mentioned in [27], Euclidean distance is the most straightforward similarity measure for time series. This similarity measure is easy on implementation and can work well on large datasets or databases [27]. On the contrary, Euclidean distance is sensitive to noise and to small variations in the time axis [28]. We can calculate the distance between two time series *Q* and *C* of length *n* using the Euclidean distance (ED):

$$ED(Q,C) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \quad \text{(Eq. 16)}$$

where $Q = q_1, q_2, \dots, q_i, \dots, q_n$ and $C = c_1, c_2, \dots, c_i, \dots, c_n$ [29].

4.2.2 Dynamic time warping

In order to have a more accurate result in small data sets, we can use Dynamic Time Warping (DTW) [27]. This similarity measure is superior over the Euclidean distance because it can define the true distance and similarity between two time series. DTW is based on the Euclidean distance.

According to [30], suppose we have two time series *Q* and *C* of length *n* and *m* respectively, where:

$Q = q_1, q_2, \dots, q_i, \dots, q_n$ and $C = c_1, c_2, \dots, c_j, \dots, c_m$

We construct an *n*-by-*m* matrix where the (i^{th}, j^{th}) element of the matrix the distance $d(q_i, c_j) = (q_i - c_j)^2$ (based on the Euclidean distance) between the two points q_i and c_j to align the two sequences (time series). Each matrix element (i,j) corresponds to the alignment between the points q_i and c_j. This alignment in time is determined by a warping path $W = w_1, w_2, \dots, w_k, \dots, w_K$ of length *K*, where $\max(m,n) \leq K < m + n - 1$, which is "a contiguous set of matrix elements that defines a mapping between *Q* and *C*", as defined by [30]. The warping path must start and finish in the "diagonally opposite corner cells of the matrix" [30], thus $w_1 = (1,1)$ and $w_K = (m,n)$. Additionally, the allowable steps in the warping path to adjacent cells have the following restriction: for $w_k = (a,b)$ then $w_{k-1} = (a',b')$ where $a - a' \leq 1$ and $b - b' \leq 1$.

Finally, the points in the warping path must be monotonically spaced in time, and thus given a point $w_k = (a,b)$, then $w_{k-1} = (a',b')$ where $a - a' \geq 0$ and $b - b' \geq 0$. There are many possible warping paths, but we choose the one that follows the above three restrictions and minimizes the warping cost, which is:

$$DTW(Q,C) = min\left\{\sqrt{\sum_{k=1}^{K} w_k}\Big/_{K}\right. \quad \text{(Eq. 17)}$$

"The K in the denominator is used to compensate for the fact that warping paths may have different lengths", as [30] is mentioning. Using dynamic programming, the path can be found as:

$$\gamma(i,j) = d(q_i, c_j) + \min\{\gamma(i-1, j-1), \gamma(i-1, j), \gamma(i, j-1)\} \quad \text{(Eq. 18)}$$

The above iterative method defines "the cumulative distance $\gamma(i,j)$ as the distance $d(i,j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements" [30].

Summarizing, Euclidean distance may be equally efficient and sometimes better in large datasets, but DTW is far more accurate in small datasets. However, the computational cost of DTW is higher, thus a single DTW calculation it has a time complexity of *O(nw)* compared to the Euclidean that has a time complexity of *O(n)* [27].

4.2.3 MINDIST

In [31] Lin et al. proposed a distance measure for their symbolic approach for representing time series, that is based on the Euclidean distance. Given two time series *Q* and *C* of length *n* and their PAA representations $\bar{Q}$ and $\bar{C}$ of length *w*, we can obtain a lower bounding approximation of the Euclidean distance using the following equation:

$$DR(\bar{Q}, \bar{C}) \equiv \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (\bar{q_i} - \bar{c_i})^2} \quad \text{(Eq. 19)}$$

Then given the words $\hat{Q}$ and $\hat{C}$ that have been generated from the PAA representations, we can have the approach of the authors of [31] defined by:

$$MINDIST(\hat{Q}, \hat{C}) \equiv \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist(\hat{q_i}, \hat{c_i})^2} \quad \text{(Eq. 20)}$$

In [31] they have explained and proved that MINDIST lower-bounds the Euclidean distance.

4.3 Data mining tasks

Most researchers use a certain number of data mining tasks to solve their problems. In this thesis we implemented the task called classification. The most common data mining tasks are:

- Classification
- Indexing
- Prediction
- Clustering
- Estimation
- Summarization
- Anomaly detection

In classification, the system assigns an unlabeled time series to one of two classes, for binomial (binary) classification, or one of many classes, for multiclass classification [9]. The system must be trained in order to label the unlabeled data. The goal is to build a model that contains a training set that has classified examples for each class that will be applied to unlabeled data to classify them [34]. To have this training set, we must split our dataset into two subsets, the training set and the testing set. The training set will be used to train our classifier and the testing set will be used to validate that the classifier predicts with accuracy. For this purpose, we use cross-validation. One type of cross validation is *k*-fold cross-validation, which splits randomly a dataset *D* into *k* mutually exclusive subsets $D_1, D_2, \dots, D_k$ of approximately equal size. The classifier is applied *k* times [40]. Another type of cross-validation is leave-one-out cross-validation, in which the number of folds is equal to the number of the examples in the dataset [41]. So, the classifier is applied once for each example, using all the other examples for training and using the selected example for testing [41].

In the end the classifier must be evaluated by its *accuracy*, *precision* and *recall*. "The accuracy of a classier C is the probability of correctly classifying a randomly selected instance" explains the author of [40]. In [42], the author presents accuracy as "the fraction of the system's interesting/uninteresting predictions that agree with the user's assessments".

Precision is the performance term that measures the retrieval specificity, which is the proportion of the retrieved items that the user judged as relevant/right.

Recall measures the retrieval coverage, that is "the proportion of the set of relevant items that is retrieved by the system" as explained in [42].

In indexing, the system can find the most similar time series in the dataset or database to the query time series, using a distance/similarity measure [9]. The time series in the database must be preprocessed and simplified using some dimensionality reduction technique/compression method or a different representation of the time series data. This way, the runtime of the search will be lower. The query must also be simplified. A set of time series will be presented as result [22]. Faloutsos has presented in [32] an interesting approach of an indexing and feature extraction algorithm called GEMINI (GEneric Multimedia INdexIng) which is claimed to have a high speed of search, but the accuracy of the results is corresponding to the distance measure [32].

In prediction, the unlabeled data are classified based on predicted values and behaviors [34]. Using training examples and historical data for these examples, a model is built to explain the observed behavior. Applying the model to current inputs, we have a prediction of future behavior as a result.

In clustering, the system makes natural groupings (clusters) of the time series in a dataset using a similarity or distance measure [9]. Clustering does not rely on predefined classes and examples like classification [34].

In estimation, the system comes up with a value for unknown continuously valued outcomes, such as income or height [34]. While classification deals with problems whose answer is yes or no (discrete variables), estimation deals with continuous variables, like estimating the number of children in a family, a household's total income or the life expectancy of a person.

In summarization, the system creates an approximation of a time series *Q* with *n* data-points, which fits on a single page, a computer screen etc. [9].

Finally, in anomaly detection, the system finds anomalies or an "interesting/ unexpected" behavior in a time series *Q* given a model with a "normal" behavior [9].

4.4 Data mining algorithms

4.4.1 *k*-nearest neighbor (*k*NN)

A common and widely used classification algorithm is *k*-nearest neighbor (*k*NN), which is easy to implement. It classifies the unlabeled data finding groups (clusters) of *k* examples in the training set that are closest to the unlabeled data, and then finds which class in the neighborhood (cluster) has the larger number of examples to assign its label to the test data [33]. The model must contain a training dataset, a similarity or distance measure to compute the distance between the data, and the *k* value that is the number of nearest neighbors that *k*NN will use. More specifically, the distance of the unlabeled data to the labeled data is computed, then the *k*-nearest neighbors are identified, and the class labels of these neighbors are used to assign the class label of the unlabeled data. The *k*-NN classification algorithm is implemented as follows:

Given as input a training dataset *D* of *k* training objects and a test object $z = (x', y')$, where x' is the data of the test object and y' its class, the algorithm computes the distance $d(x', x)$ (or similarity) between *z* and every object $(x, y) \in D$, where *x* is the data of the training object and *y* its class, and selects the set $D_z \subseteq D$ of the *k* closest training objects to *z*. Finally, the test object is classified based on the majority voting:

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D_z} I(v = y_i) \quad \text{(Eq. 21)}$$

where v is a class label, y_i is the class label for the i^{th} nearest neighbor, and $I(\cdot)$ is "an indicator function that returns 1 if its argument is true and 0 otherwise" [33]. A better approach instead of

majority voting is to weigh its object's vote by its distance, so we have the weight factor: $w_i = 1/d(x', x_i)^2$ and the last step of the algorithm will be alternated as follows:

$$y' = \underset{v}{\operatorname{argmax}} \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i) \quad \text{(Eq. 22)}$$

The value *k* must be chosen wisely, because if it is too large then the cluster may include unnecessary examples from other classes. Moreover, building this algorithm's model may be cheap, but has high computational cost. In [33] it is recommended to use the cosine similarity as distance measurement to classify documents and avoid the Euclidean distance when our data are described with high dimensionality.

4.4.2 Support vector machines (SVM)

SVM is one of the most accurate algorithms, and as is analyzed in [33], it works well with high-dimensional data and it requires few examples for training. In a binary classification problem, SVM is able to find the best classification function by maximizing the margin between the two classes, to distinguish members of the one class from another in the training dataset. This margin "is defined as the amount of separation between the two classes as defines by the hyperplane" [33]. In a linear classification function, a hyperplane *f(x)* is the one "that passes through the middle of the two classes, separating them" [33]. A SVM classifier attempts to maximize the following function, with respect to $\vec{w}$ vectors and the constant *b*, which define the hyperplane, to ensure on finding the maximum margin hyperplanes

$$L_P = \frac{1}{2}\|\vec{w}\| - \sum_{i=1}^{t} a_i y_i (\vec{w} \cdot \vec{x_i} + b) + \sum_{i=1}^{t} a_i \quad \text{(Eq. 23)}$$

where *t* is the number of training examples, and a_i the Lagrange multipliers and are non-negative numbers such that the derivates of L_P, the Lagrangian, with respect to a_i are zero.

4.4.3 Decision trees

Decision trees can be used for both classification and prediction, and are useful for data exploration and modeling. The authors of [34] describe a decision tree as a structure that divide a large set of records into smaller sets with successive division, by applying decision rules. The small sets (child nodes) become more and more like one another, until we reach the root node of the decision tree. To build a decision tree, first a record is entered in the root node and then the root node applies a question/test to determine which child node the record will meet next. The child node that has been selected will apply a test to determine the next child node that will be visited by the record etc. until the record arrives to a leaf node. Each leaf node has its unique path to the root node, that is an expression of the rule that is used to classify the records. For this reason, many different leaf nodes may make the same classification, but for different reasons. A decision tree grows by repeatedly splitting the data into smaller sets, according to a function of a single input field, in a way that each new level of the tree (each new generation of nodes) has greater purity than its ancestors with respect to the binary, categorical variable. The input field that has been chosen must

make the best split, which is a split that separates the records into groups in which a single class predominates, that increases the purity of the groups and that creates nodes of similar size. For estimation, it is better to use regression trees, that are more suitable for continuous variables [34]. Note that, regression is the process that using one value of a pair of correlated variables *X* and *Y*, predicts the other value of the pair [34].

4.4.4 Random forests

Random forests are an improved approach where a set of growing trees vote for the most popular class [35]. So, a random forest is a classifier that contains tree-structured classifiers $\{h(x, \Theta_k), k = 1, \dots, \}$, where Θ_k is a random vector, independent of its past random vectors, but with the same distribution, and x is an input vector (a set of random trees). Each tree classifier will vote for the most popular class at the input vector x. A margin function measures the area to which the average number of votes for the right class at the random vector *X*, *Y* exceeds the average vote for any other class. "The larger the margin, the more confidence in the classification" [35]. The margin function given a group of classifiers $h_1(x)$, $h_2(x)$,…, $h_K(x)$ and a training set drawn at random from the distribution of the random vector *X*, *Y*, is defined by:

$$mg(X,Y) = av_k I(h_k(X) = Y) - max_{j \neq Y} av_k I(h_k(X) = j) \quad \text{(Eq. 24)}$$

where *I(·)* is the indicator function. Also, the generalization error is defined by:

$$PE^* = P_{X,Y}(mg(X,Y) < 0) \quad \text{(Eq. 25)}$$

where the subscripts *X*, *Y* indicate that the probability is over the *X*, *Y* space. So, for a random forest, the margin function is defined by:

$$mr(X,Y) = P_\Theta(h(X,\Theta) = Y) - max_{j \neq Y} P_\Theta(h(X,\Theta) = j) \quad \text{(Eq. 26)}$$

Random forests are accurate for classification and regression when the right kind of randomness (bagging, random features, etc.) is chosen [35].

4.4.5 Gradient boosting machines or gradient boosting/boosted trees

Gradient boosting trees is a robust method that requires little data preprocessing and adjustment of parameters. It is an iterative algorithm that combines functions with high prediction error to produce a high accuracy prediction rule [36] minimizing the squared-error [37]. In other words, gradient boosting is an ensemble of either classification or regression tree models. The goal of the gradient boosting machines algorithm is to reconstruct the unknown functional dependence $x \xrightarrow{f} y$ from a training dataset (x_i, y_i), for $i = 1, \dots, N$, with our estimate $\hat{f}(x)$ such that the chosen loss function $\Psi(y, f)$ is minimized (where $x_i = (x_1, \dots, x_K)$ an input vector and y_i its class label, and the chosen base-learner $h(x, \theta)$) [38]. So, the algorithm starts initializing [36, 38]:

$$\hat{f}(x) = y, \hat{f}(x) = \underset{f(x)}{\operatorname{argmin}} \Psi(y, f(x)) \quad \text{(Eq. 27)}$$

At each iteration t, $1 \leq t \leq T$, the algorithm computes the negative gradient $g_t(x_i)$, for $i = 1, \dots, N$, as the following and chooses a new base-learner function $h(x, \theta_t)$ [36, 38]:

$$g_t(x_i) = -\left[\frac{\partial \Psi\left(y_i, f(x_i)\right)}{\partial f(x_i)}\right]_{f(x)=f_{t-1}(x)} \quad \text{for } i = 1, \ldots, M. \quad \text{(Eq. 28)}$$

Then, the algorithm finds the best gradient step-size ρ_t:

$$\rho_t = \underset{\rho}{\operatorname{argmin}} \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f_{t-1}}(x_i) + \rho h(x_i, \theta_t)\right] \quad \text{(Eq. 29)}$$

The output of the algorithm is:

$$\widehat{f_t}(x) = \widehat{f_{t-1}}(x) + \rho_t h(x, \theta_t) \quad \text{(Eq. 30)}$$

According to the research of the authors of [38], the algorithm provided excellent accuracy at solving classification and regression problems and is easy on model application [36].

4.4.6 Naive Bayes

Naive Bayes is a comprehensible classification method using the Bayes' theorem for solving supervised classification problems and it is very important due to its simplicity to be built and its fast application on huge datasets [33]. Given a training set with labeled examples categorized in two classes *i* = 0, 1, we construct a score/threshold *t* that, if an unlabeled instance has larger score than the threshold, it is associated with objects from class 1, otherwise (if the score is smaller than the threshold) with objects from class 0. According to [33], we define $P(i|x)$ to be the probability that an object with vector of variables $x = x_1, \ldots, x_p$ of length p belongs to the class *i*. Using elementary probabilities "we can decompose $P(i|x)$ as proportional to $f(x|i)P(i)$, where $f(x|i)$ is the conditional distribution of *x* for class *i* objects, and $P(i)$ is the probability that an object will belong to class *i* if we know nothing further about it". So, for this problem a suitable ratio/score would be:

$$\frac{P(1|x)}{P(0|x)} = \frac{f(x|1)P(1)}{f(x|0)P(0)} \quad \text{(Eq. 31)}$$

Naive Bayes assumes that the components of the vector *x* are independent, so:

$$f(x|i) = \prod_{i=0}^{1} \prod_{j=1}^{p} f(x_j|i) \quad \text{(Eq. 32)}$$

Given the independence assumption, the ratio of Eq. 31 can become:

$$\frac{P(1|x)}{P(0|x)} = \frac{\prod_{j=1}^{p} f\left(x_j|1\right)P(1)}{\prod_{j=1}^{p} f(x_j|0)P(0)} = \frac{P(1)}{P(0)} \prod_{j=1}^{p} \frac{f\left(x_j|1\right)}{f\left(x_j|0\right)} \quad \text{(Eq. 33)}$$

We want to produce a ratio which is monotonically related to $P(i|x)$, and so we take the logarithms of Eq. 33 and in the end, we have the following:

$$\ln\frac{P(1|x)}{P(0|x)} = \ln\frac{P(1)}{P(0)} + \sum_{j=1}^{p} \ln\frac{f(x_j|1)}{f(x_j|0)} \quad \text{(Eq. 34)}$$

The naive Bayes model is widely used for text classification and is very effective.

5 INFORMATION RETRIEVAL TECHNIQUES

5.1 Introduction

Based on [39], information retrieval, as a field of study, is finding an answer to a query from inside large collections of data. Usually, the system finds text documents that satisfy the information need (what the user needs to know) from databases stored in computers. We deal with Boolean retrieval models, in which we can pose any query with terms that are combined with Boolean expressions, such the operators AND, OR and NOT. These models view each document as a set of words.

A document can be a book or a chapter of a book, and many of them compose a collection that will be used to retrieve data from. Boolean queries retrieve a set of matching documents, but in case of large document collections, a score needs to be determined for each document, that represents how good of a match is each document for the query. Therefore, a document that mentions a term of the query more often than the other documents must receive a higher score.

5.2 TF-IDF weighting

To find the score of each document in a collection, we have to assign a weight to each term in the document depending on the term(s) of the query [39]. This weight depends on the number of occurrences of each term in the document. This weighting scheme is called term frequency. Term frequency is denoted as $tf_{t,d}$ and is the total number of occurrences of a term *t* in a document *d*. But we need another feature for each term, to examine if the term is important enough in respect to all the documents of the collection. Hence, the document frequency (df_t) is calculated. It is the number of the documents that contain a term *t*. Then, the weight of a term can be calculated using its df and the inverse document frequency (idf) of a term *t* can be defined by the equation:

$$idf_t = \log \frac{N}{df_t} \quad (\text{Eq. } 35)$$

where *N* is the total number of documents in a collection. So, we can say that a term is rare if its idf is high, and if the term is frequent its idf is low. Combining the definitions of term frequency and inverse document frequency, we have the following equation:

$$tf - idf_{t,d} = tf_{t,d} \times idf_t \quad (\text{Eq. } 36)$$

which is the weight of a term *t* in a document *d*. TF-IDF is very high when a term is frequent in a small collection of documents, lower when a term is rare in a document or occurs in many documents, and very low when a term occurs in almost all documents.

5.3 Other information retrieval operations

In contrast to Boolean retrieval that treats each term as a word with a specific order in the document, there is another information retrieval model that uses the number of occurrences of each term to describe it. This model is called *bag of words* and ignores the ordering of the words in a document, and so, many documents with the same words may seem identical [39].

To define the similarity between two documents we compute the cosine similarity [39]. The similarity of two documents d_1 and d_2 is computed by the cosine similarity of their vectors $\vec{V}(d_1)$ and $\vec{V}(d_2)$ as defined by the equation:

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{\left|\vec{V}(d_1)\right|\left|\vec{V}(d_2)\right|} \quad \text{(Eq. 37)}$$

where the numerator represents the inner product of the two vectors and the denominator is the product of their Euclidean lengths [39]. The inner product of two vectors $\vec{x}$ and $\vec{y}$ of length M is defined as:

$$\vec{x} \cdot \vec{y} = \sum_{i=1}^{M} x_i y_i \quad \text{(Eq. 38)}$$

and the Euclidean length of a document vector $\vec{V}(d)$ with M components is defined as:

$$\left|\vec{V}(d)\right| = \sqrt{\sum_{i=1}^{M} \vec{V}_i^2(d)} \quad \text{(Eq. 39)}$$

Finally, to measure and evaluate the effectiveness of a system in information retrieval, we also calculate precision and recall as in classification. According to [39], "precision is the fraction of retrieved documents that are relevant" and is defined by:

$$Precision = \frac{\#(relevant\ retrieved\ items)}{\#(retrieved\ items)} = P(relevant|retrieved)$$

"Recall is the fraction of relevant documents that are retrieved" [39] and is defined by:

$$Recall = \frac{\#(relevant\ retrieved\ items)}{\#(relevant\ items)} = P(retrieved|relevant)$$

When an item is retrieved and relevant, it is denoted by *tp* (true positive), and when an item is retrieved but not relevant, it is denoted by *fp* (false positive). *fn* (false negatives) denotes the items that are relevant and not retrieved, and *tn* (true negatives) denotes the items that are non-relevant and not retrieved. Based on that, we have the following equations of precision, recall and accuracy, and the contingency table [39]:

	Relevant	Nonrelevant
Retrieved	True Positive (TP)	False positives (FP)
Not retrieved	False negative (FN)	True negatives (TN)

$$Precision = TP/TP + FP$$

$$Recall = TP/TP + FN$$

$$Accuracy = (TP + TN)/(TP + FP + FN + TN)$$

6 IMPLEMENTATION

6.1 The dataset

The implementation was done in MATLAB and RapidMiner. The dataset used is the raw data from the research of Wu et el. [4], and contains digital images of leaves from Asia and North America. Our dataset contains 50 leaves from each species of the 32 species/plant classes. Each image contains the main leaf, without the stem, with a white background. Figures 1 to 4, show some examples of leaf images of the raw dataset with their Latin names [4]. Additionally, to have more samples, the images have been rotated 90 degrees and -90 degrees, tripling the size of the dataset.

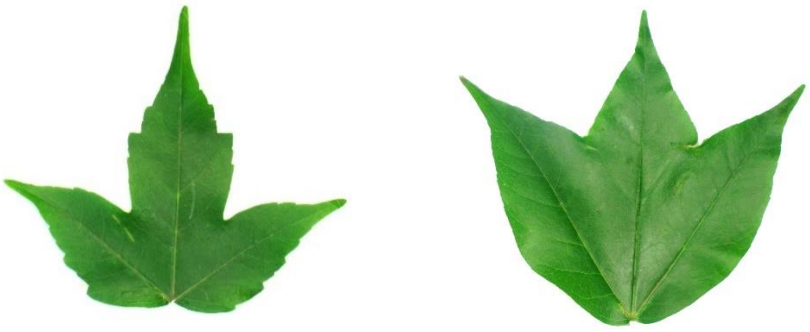

Figure 1. Leaf examples from the species “Acer buergerianum”, a species of maple.

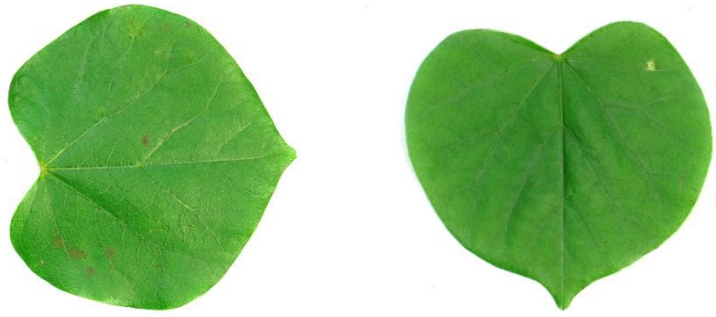

Figure 2. Leaf examples from the species “Cercis chinensis”, commonly known as Chinese redbud.

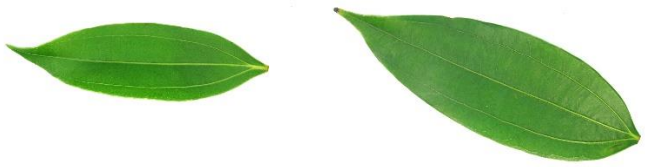

Figure 3. Leaf examples from the species “Cinnamomum japonicum”, commonly known as Japanese cinnamon.

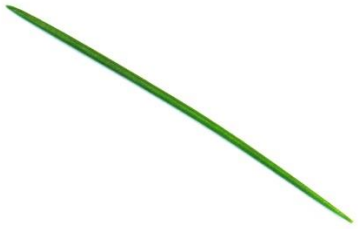

Figure 4. Leaf examples from the species “Cedrus deodara”, a species of cedar.

6.2 Image preprocessing

The image preprocessing was done in MATLAB. Each leaf image is converted to a time series and to do so, an algorithm was used to extract the perimeter of the leaf. Guided by the image preprocessing algorithm of [4], the outline of each leaf was extracted successfully. First, the input image was resized with the function imresize(), to have image vectors with lower dimensionality, making them easier to handle. The final image is 50% smaller than the initial. Secondly, the resized image is converted from RGB to grayscale with the function rgb2gray(). Then, the grayscale image is converted to binary using im2bw() with a threshold of level 90% (0.9). That means that all the pixels of the gray image that are greater than that level were replaced with the value 1 (white) and all the other pixels were replaced with the value 0 (black). Next, an average filter was applied for smoothing. The filter filled some small holes and eliminated dark spots in the white background. Figures 5 and 6, show the corresponding steps on a leaf from species “Acer palmatum”, commonly known as Japanese maple.

Figure 5. The initial RGB test image (left) and the grayscale image (right).

Figure 6. The initial binary image (left) and the smoothed binary image (right).

Next, a Laplacian filter was implemented and multidimensional filtering using convolution to enhance the image. Finally, the negative of the image was used to define the perimeter of the leaf.

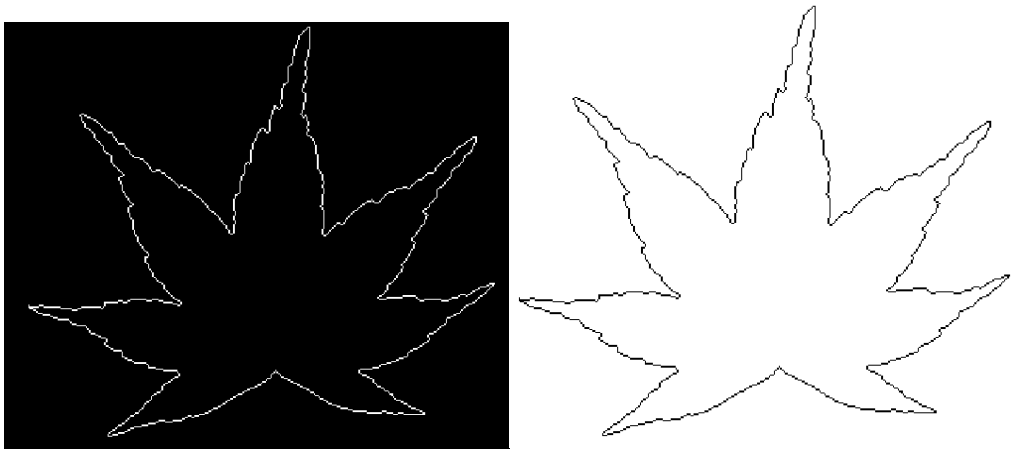

Figure 7. The enhanced image (left) and the final image that show the boundary of the leaf.

Now, we need to find the center of the leaf in order to calculate the distances from the center to each point of the outline of the leaf. The outline contains only black points (value 0) and keeping the coordinates of each point, we can find the "center" of the leaf. In the end, using the Euclidean distance (Eq. 16), we can measure for every point of the outline its distance from the center. The result will be a plot/curve that will be used as time series to describe each leaf. It should be noted that the vectors of the leaves, and hence of the time series, had different dimensions because each leaf is unique. For this reason, we had to decide a fixed size of the time series to prevent any later problems. We decided to keep only 300 points from the perimeter of the leaf and calculate only

300 distances from the center of the leaf. This way, the dimensionality is also greatly reduced and that will be very useful in when we implement classification algorithms. Figure 8 shows the two versions of the time series of the above leaf. This dataset is very appropriate for extracting time series from the leaves due to the noticeable foreground and noise-less background.

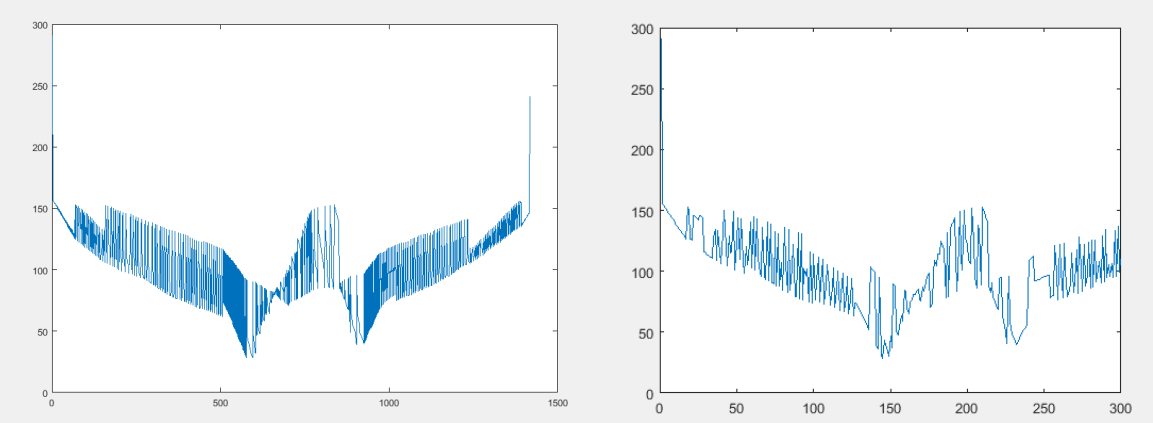

Figure 8. The time series of the Japanese maple leaf with 1418 points (left) and the corresponding time series with 300 points (right).

6.3 Time series processing

This step includes the normalization, decomposition and numerosity reduction of the time series. Z-normalization was implemented in the time series with Eq. 11 as it was proposed in [11] and [22]. In MATLAB this is defined as *(X - mean(X))/std(X)*, where *X* is the time series from the previous step, image preprocessing. Some leaves were identical, but in different levels in the y-axis, and so, by implementing normalization, we realized that the leaves can be compared more accurately and correctly. This was the last step for processing the time series and the data are ready to be an input to a classification model. Two classification models were used. One takes as input the normalized time series data and the other the time series data represented by SAX strings. For the latter, the time series processing had to continue by implementing Fast Fourier Transform and its inverse. Figure 12 shows how the distance between two time series has been reduced due to normalization, and figure 11 shows the distance between the time series without normalization. To test this, we used the Dynamic Time Warping (function dtw() in MATLAB) distance measurement. The images and their time series that were used appear in figures 9 and 10.

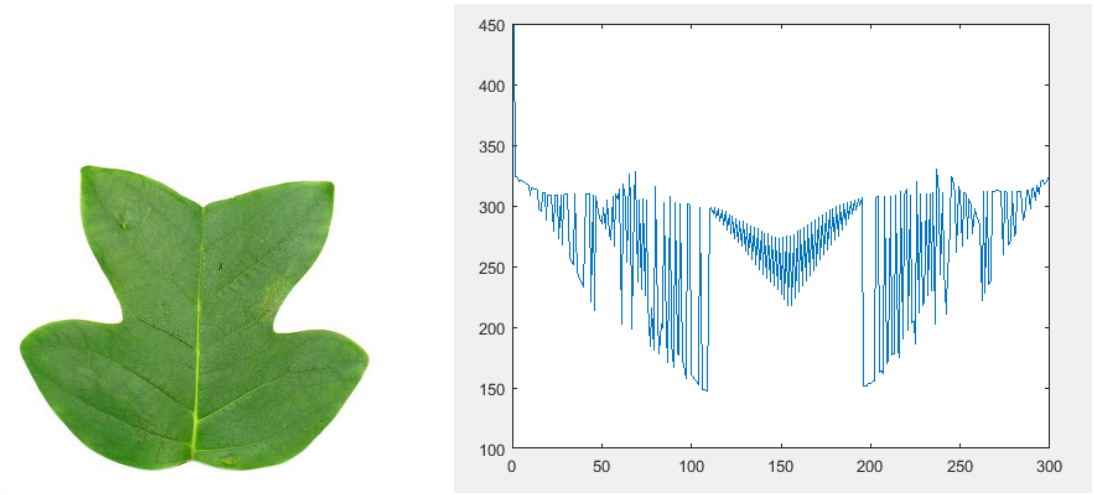

Figure 9. The original image (left) and the corresponding time series (right).

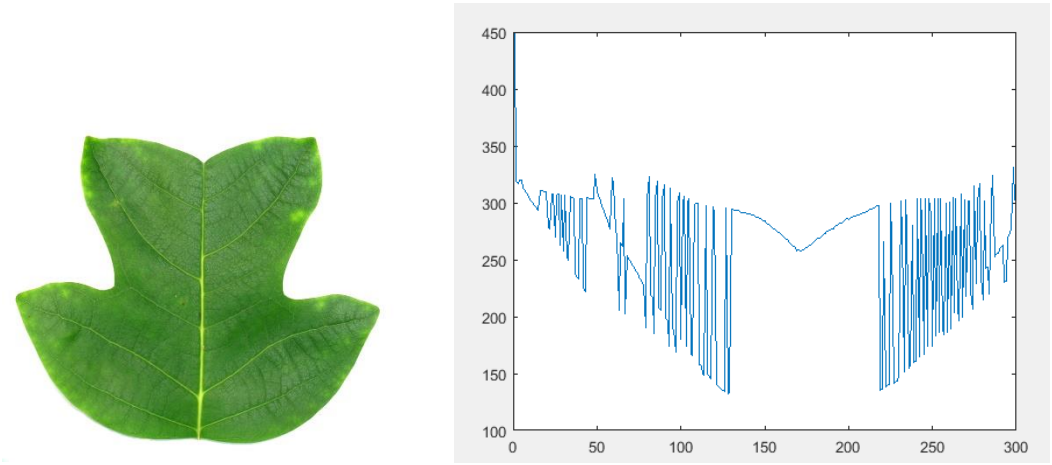

Figure 10. The original image (left) and the corresponding time series (right).

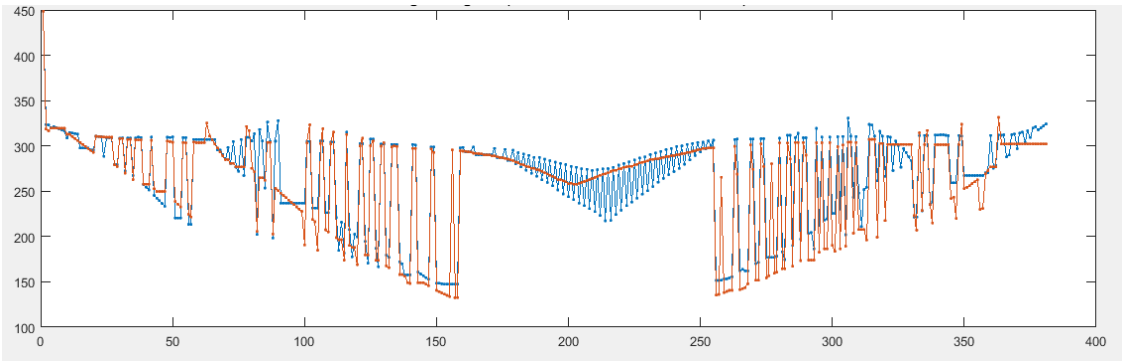

Figure 11. Comparing the two un-normalized time series with DTW with distance: 5115.9318

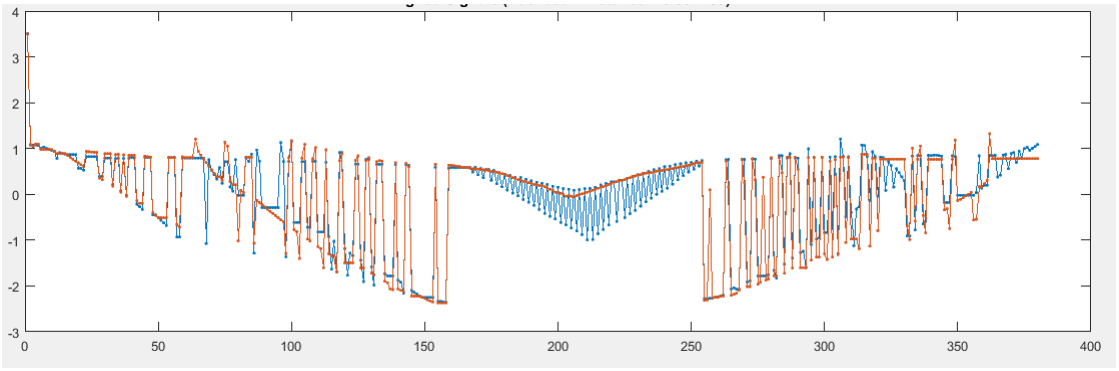

Figure 12. Comparing the two normalized time series with DTW with distance: 79.928

We notice that the difference is quite important, so for better results, normalization is necessary. We will use the data that have been produced so far to train and test the first classification model that handles only time series as inputs.

We will continue with processing the time series to prepare the data for the second classification model. Once the time series were normalized, a Fast Fourier Transform was implemented for decomposition of the time series into a sum of sine waves. We kept the first 40 coefficients and with Inverse Fast Fourier Transform we reconstructed the signal/time series. In figures 13 and 14 there is an example of a normalized time series and the corresponding final time series by keeping the first 5, 20 and 40 coefficients.

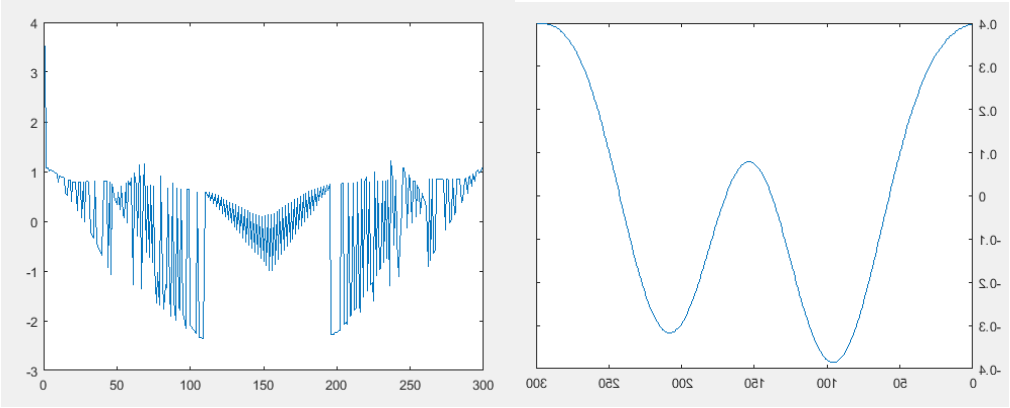

Figure 13. The time series (left) and the decomposed keeping 5 coefficients (right).

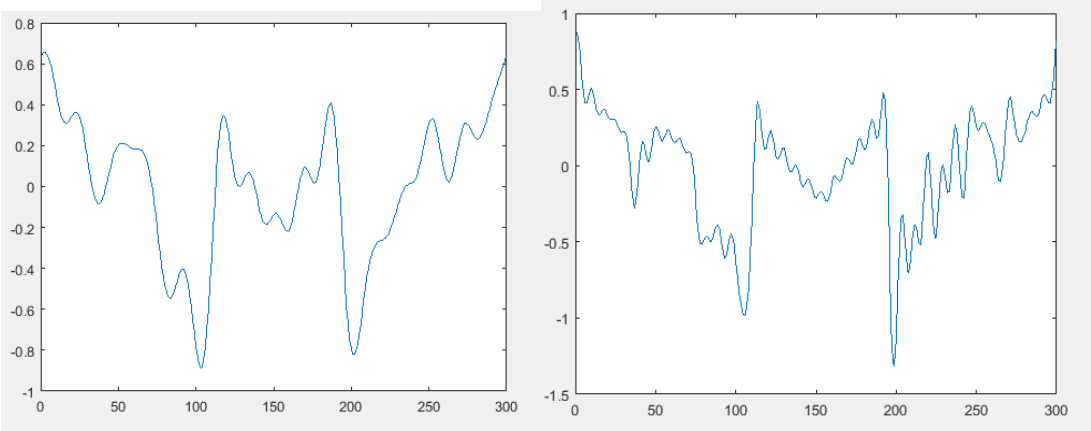

Figure 14. The decomposed time series keeping the first 20 coefficients (left) and keeping the first 40 coefficients (right).

As we can see, by keeping few coefficients we lose important information and so, keeping the first 40 coefficients, we achieve data compression without losing important information from the time series. Now, we can use the processed time series for the symbolic representation.

The code that was used to represent the time series as strings, was from the research of Keogh et al. [31] who proposed SAX (Symbolic Aggregate approximation). As mentioned in previous chapters, the algorithm takes as input the time series, the desired length of the output string and the alphabet size. First, it transforms the time series into the desired number of PAA (Piecewise Aggregate Approximation) segments with Eq. 14 for dimensionality reduction. Then, "translates" the segments/coefficients into symbols using cut points depending on the alphabet size. Note that, the algorithm has a restriction in the number of segments. The number of segments must be divisible by the data/time series length. Figure 15 shows the above processed time series converted into a PAA representation with alphabet size 3 and 10 coefficients, and the generated symbols according to the segments.

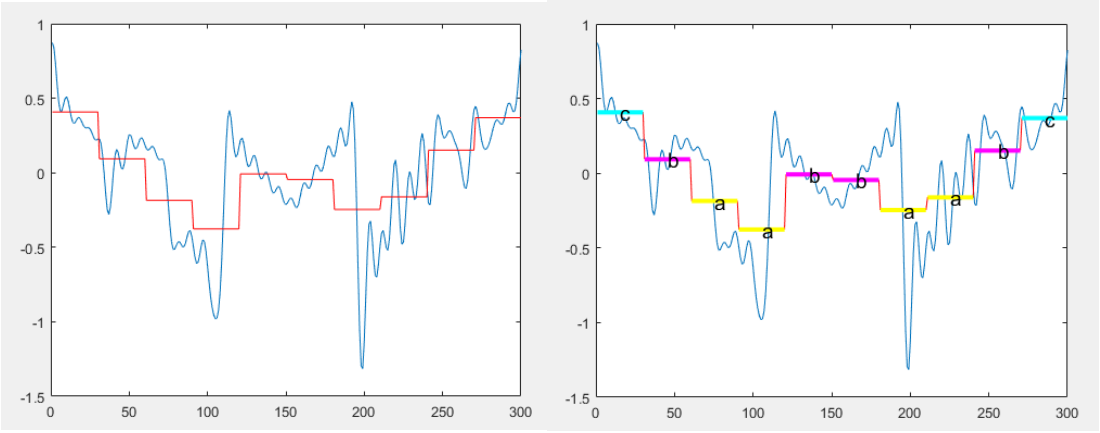

Figure 15. The PAA representation of the time series (left) and the symbolic one (right).

The symbolic representation is more accurate and with more detail if we use a larger number of coefficients. Additionally, when we use a larger alphabet, the strings tend to be less frequent in the whole dataset and that is helpful for the classification algorithms afterwards. Figures 16 and 17 show the symbolic representation of the time series using 50 segments and alphabet size 10.

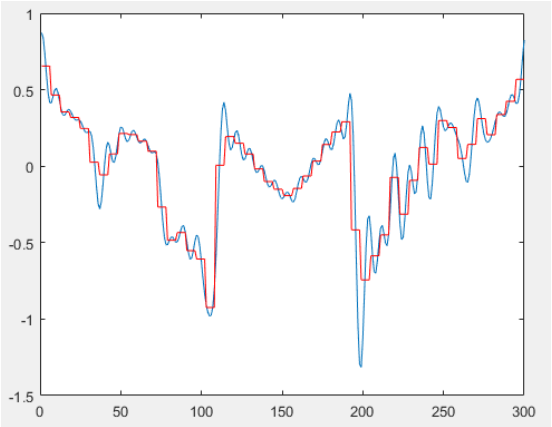

Figure 16. The PAA representation of the time series with 50 segments and alphabet size 10.

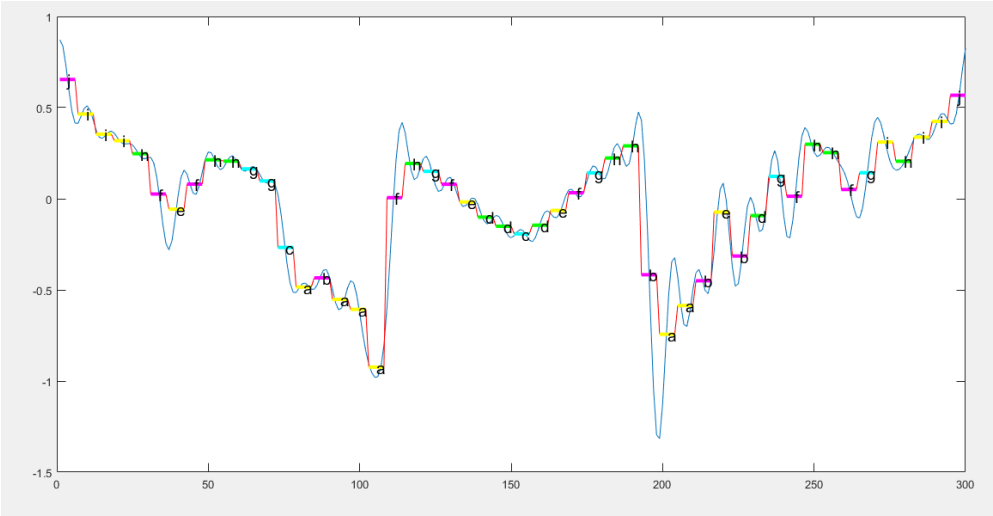

Figure 17. The symbolic representation using 50 segments and alphabet size 10.

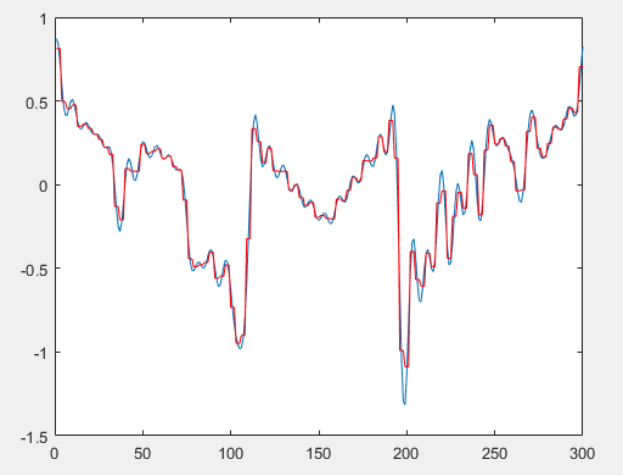

Figure 18. The PAA representation using 100 segments and alphabet size 10.

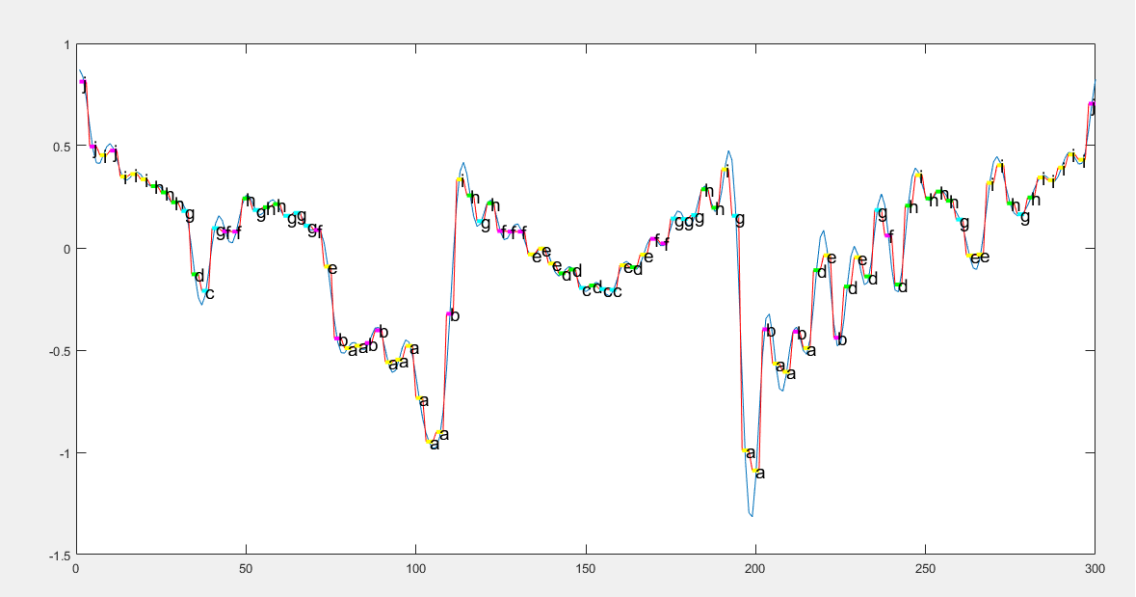

Figure 19. The symbolic representation using 100 segments and alphabet size 10.

In the last two figures (18 and 19), we can see the symbolic representation that will be used to train and test the classification model. All the time series were converted into strings with length 100 and with alphabet size 10. As mentioned, we used another two samples of each image, 90 and -90 degrees rotated versions of it. When the images are rotated 90 and -90 degrees, their time series were different, but when rotated 180 degrees the time series were similar.

6.4 The classification models

6.4.1 Introduction

In this thesis were implemented two classification models in RapidMiner. One takes as data input the raw time series from every leaf image of the dataset. All these time series have length 300 and are z-normalized. The other classification model takes as input data the strings that have been generated with the SAX algorithm.

6.4.2 Classification with the time series data

We tested our first model with the k-NN and Gradient Boosted Trees classification methods. Our problem is a multi-class classification problem, so we used the one-against-all technique. We did not test our model using 32 classes, but only two and we applied the classification model 32 times (one for every class) using two classes each time. One class had the label of the current species we were testing, and the rest were referred to with the class label "Other". Additionally, we used an operator in RapidMiner that performs dimensionality reduction based on Singular Value Decomposition. Without SVD, the k-NN algorithm was extremely slow using both the Euclidean distance and Dynamic Time Warping distance. k-NN using DTW distance had the biggest runtime followed by k-NN using the Euclidean distance. Gradient Boosted Trees, on the contrary was really fast. In the training phase, the 90% of the dataset was used and K-fold cross-validation was applied. The model created in the training phase was applied to a test set, the remaining 10% of the dataset. Considering accuracy, recall and precision, the classification model was evaluated and the best algorithm for our purpose was Gradient Boosted Trees using 100 trees with maximal depth 5. The average accuracy with Gradient Boosted Trees is 97.58%, the average recall is 50.03% and the average precision is 63.07%. Wu et al. [4], extracted certain features from each leaf image from their dataset. They used a probabilistic neural network for their classification model and their average accuracy is 90.312% [4]. Figure 20 shows the scatter chart for the time series data of each class, represented by two variables using Singular Value Decomposition (SVD).

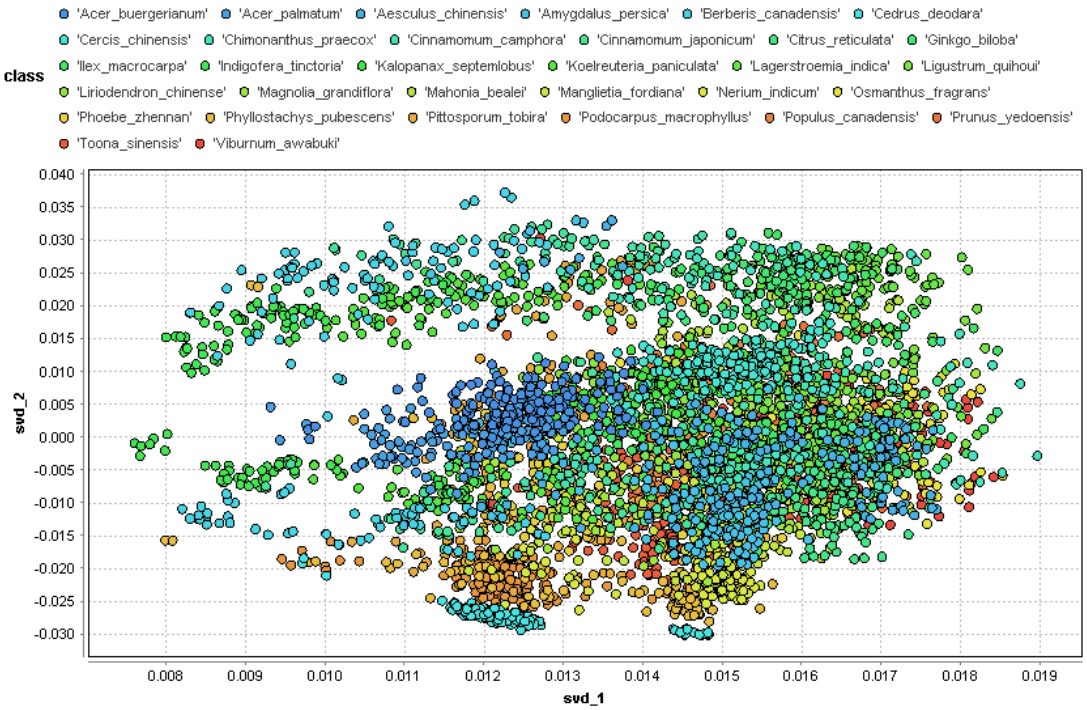

Figure 20. The scatter chart that displays the time series of all 32 classes represented by two variables generated by SVD.

6.4.3 Classification with the SAX strings

In this model we used also the one-against-all technique. The model takes as input the strings generated by the SAX algorithm that represent the time series. Then, we used an operator in RapidMiner that made vectors of four characters from the SAX strings using the TF-IDF scheme. Next, an operator assigned weights to the vectors by calculating their relevance based on information gain, and another operator selected the vectors that had the top 40 highest weights with respect to the input weights. A K-fold cross-validation was implemented as well. Decision tree, random forest, support vector machine, gradient boosted trees and naive Bayes were the classification algorithms used for testing the model. Decision trees, random forests and SVM were very fast but a big disappointment regarding their accuracy, recall and precision. On the contrary, gradient boosted trees and naive Bayes gave better results and were very fast. Gradient Boosted Trees used 100 trees with maximal depth 5 and achieved an average accuracy of 96.19%, an average recall of 62.27% and an average precision of 33.86%. With naive Bayes the average accuracy is 69.93%, the average recall 67.04% and the average precision a very low 8.18%. In [11], the authors used another leaf dataset and implemented their classification and extraction of characteristic patterns model, SAX-VSM that has been analyzed in a previous chapter. Their algorithm achieved an accuracy of 89%. Figure 21 shows the scatter chart for the SAX strings data of each class, represented by two variables using Singular Value Decomposition (SVD).

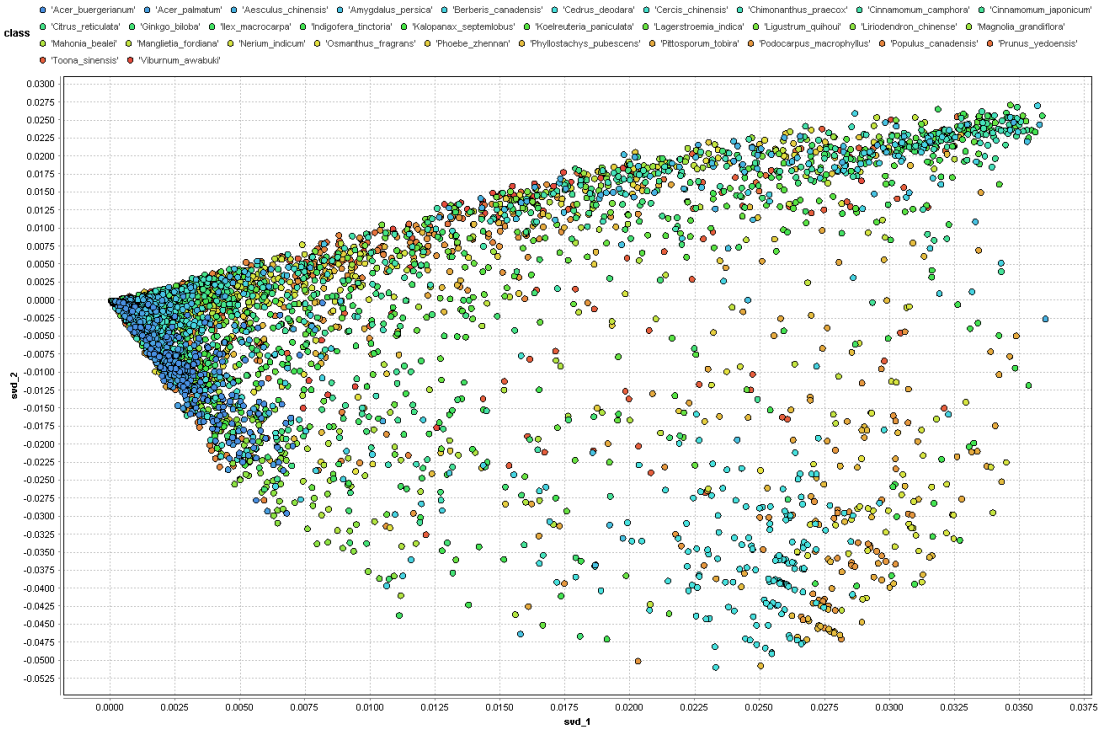

Figure 21. The scatter chart that displays the SAX words of all 32 classes represented by two variables generated by SVD.

7 SUMMARY

The algorithm that was implemented to extract the outline of a leaf image worked successfully for all 50 leaves of the 32 species. The outline of each leaf was converted to a unique time series. An algorithm was implemented to normalize and process the time series in order to achieve dimensionality, numerosity and computational cost reduction. Using SAX, the time series data were transformed into unique words. 90% of the data was used for training and the 10% was used for testing each model. The models proposed in this thesis achieved an accuracy of 97.58%, using the raw time series data, and an accuracy of 96.19%, using time series data represented by SAX strings. The recall and precision of each model have room for improvement.

8 FUTURE WORK

The research can be improved by adding more examples in the raw dataset and by extracting more features that describe a leaf image well. The research can be expanded by adding images with noisy background and by implementing an algorithm suitable to extract features from this type of images. Additionally, the user interface can be improved upon, to provide the user with results. An indexing method can be implemented as well, to predict the class label of the leaf that the user wants to know. Finally, the application could be improved by adding functionality to identify leaves from images that aren’t part of the dataset.

9 REFERENCES

[1] Clarke, B., Fokoue, E. and Zhang, H.H., 2009. *Principles and theory for data mining and machine learning*. Springer Science & Business Media.

[2] Han, J., Pei, J. and Kamber, M., 2011. *Data mining: concepts and techniques*. Elsevier.

[3] Shrestha, B., 2010. Classification of plants using images of their leaves. *Appalachian State University*.

[4] Wu, S.G., Bao, F.S., Xu, E.Y., Wang, Y.X., Chang, Y.F. and Xiang, Q.L., 2007, December. A leaf recognition algorithm for plant classification using probabilistic neural network. In *Signal Processing and Information Technology, 2007 IEEE International Symposium on* (pp. 11-16). IEEE.

[5] Wold, S., Esbensen, K. and Geladi, P., 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems*, *2*(1-3), pp.37-52.

[6] Specht, D.F., 1990. Probabilistic neural networks. *Neural networks*, *3*(1), pp.109-118.

[7] Kumar, N., Belhumeur, P.N., Biswas, A., Jacobs, D.W., Kress, W.J., Lopez, I.C. and Soares, J.V., 2012. Leafsnap: A computer vision system for automatic plant species identification. In *Computer Vision–ECCV 2012* (pp. 502-516). Springer, Berlin, Heidelberg.

[8] Qi, X., Xiao, R., Li, C.G., Qiao, Y., Guo, J. and Tang, X., 2014. Pairwise rotation invariant co-occurrence local binary pattern. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *36*(11), pp.2199-2213.

[9] Lin, J., Keogh, E., Lonardi, S. and Chiu, B., 2003, June. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery* (pp. 2-11). ACM.

[10] Kantor, P.B., 1994. Information retrieval techniques. *Annual Review of Information Science and Technology (ARIST)*, *29*.

[11] Senin, P. and Malinchik, S., 2013, December. Sax-vsm: Interpretable time series classification using sax and vector space model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on* (pp. 1175-1180). IEEE.

[12] Masters, B.R., Gonzalez, R.C. and Woods, R., 2009. Digital image processing. *Journal of biomedical optics*, *14*(2), p.029901.

[13] Patidar, P., Gupta, M., Srivastava, S. and Nagawat, A.K., 2010. Image de-noising by various filters for different noise. *International journal of computer applications*, *9*(4).

[14] Series, B.T., 2011. Studio encoding parameters of digital television for standard 4: 3 and wide-screen 16: 9 aspect ratios.

[15] Otsu, N., 1979. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, *9*(1), pp.62-66.

[16] Bradley, D. and Roth, G., 2007. Adaptive thresholding using the integral image. *Journal of Graphics Tools*, *12*(2), pp.13-21.

[17] Fisher, R., Perkins, S., Walker, A. and Wolfart, E., 1994. Hypermedia image processing reference. *Department of Artificial Intelligence, University of Edinburgh*.

[18] Pratt, W.K., 2013. *Introduction to digital image processing*. CRC Press.

[19] Keogh, E. and Kasetty, S., 2003. On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, *7*(4), pp.349-371.

[20] Bloomfield, P., 2004. *Fourier analysis of time series: an introduction*. John Wiley & Sons.

[21] Al Shalabi L, Shaaban Z, Kasasbeh B. Data mining: A preprocessing engine. Journal of Computer Science. 2006 Sep;2(9):735-9.

[22] Vlachos, M., 2005. A practical time-series tutorial with matlab. In *European conference on machine learning*.

[23] Keogh, E., Chakrabarti, K., Pazzani, M. and Mehrotra, S., 2001. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, *3*(3), pp.263-286.

[24] Wall, M.E., Rechtsteiner, A. and Rocha, L.M., 2003. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis* (pp. 91-109). Springer US.

[25] Har-Peled, S., Roth, D. and Zimak, D., 2003. Constraint classification for multiclass classification and ranking. In *Advances in neural information processing systems* (pp. 809-816).

[26] Aly, M., 2005. Survey on multiclass classification methods. *Neural Netw*, *19*.

[27] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X. and Keogh, E., 2008. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, *1*(2), pp.1542-1552.

[28] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D. and Keogh, E., 2003, August. Indexing multi-dimensional time-series with support for multiple distance measures. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 216-225). ACM.

[29] Batista, G.E., Wang, X. and Keogh, E.J., 2011, April. A complexity-invariant distance measure for time series. In *Proceedings of the 2011 SIAM International Conference on Data Mining* (pp. 699-710). Society for Industrial and Applied Mathematics.

[30] Keogh, E.J. and Pazzani, M.J., 2001, April. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM International Conference on Data Mining* (pp. 1-11). Society for Industrial and Applied Mathematics.

[31] Lin, J., Keogh, E., Wei, L. and Lonardi, S., 2007. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, *15*(2), pp.107-144.

[32] Faloutsos, C., 1996. *Searching multimedia databases by content* (Vol. 3). Springer Science & Business Media.

[33] Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A., Liu, B., Philip, S.Y. and Zhou, Z.H., 2008. Top 10 algorithms in data mining. *Knowledge and information systems*, *14*(1), pp.1-37.

[34] Berry, M.J. and Linoff, G., 1997. *Data mining techniques: for marketing, sales, and customer support*. John Wiley & Sons, Inc..

[35] Breiman, L., 2001. Random forests. *Machine learning*, *45*(1), pp.5-32.

[36] Guelman, L., 2012. Gradient boosting trees for auto insurance loss cost modeling and prediction. *Expert Systems with Applications*, *39*(3), pp.3659-3667.

[37] Trai Dietterich, T.G., Ashenfelter, A. and Bulatov, Y., 2004, July. Training conditional random fields via gradient tree boosting. In *Proceedings of the twenty-first international conference on Machine learning* (p. 28). ACM.

[38] Natekin, A. and Knoll, A., 2013. Gradient boosting machines, a tutorial. *Frontiers in neurorobotics*, *7*.

[39] Christopher, D.M., Prabhakar, R. and Hinrich, S.C.H.Ü.T.Z.E., 2008. Introduction to information retrieval. *An Introduction To Information Retrieval*, *151*, p.177.

[40] Kohavi, R., 1995, August. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai* (Vol. 14, No. 2, pp. 1137-1145).

[41] Sammut, C. and Webb, G.I. eds., 2011. Encyclopedia of machine learning. Springer Science & Business Media.

[42] Alvarez, S.A., 2002. An exact analytical relation among recall, precision, and classification accuracy in information retrieval. Boston College, Boston, Technical Report BCCS-02-01, pp.1-22.

APPENDIX A

Results of Image preprocessing algorithm

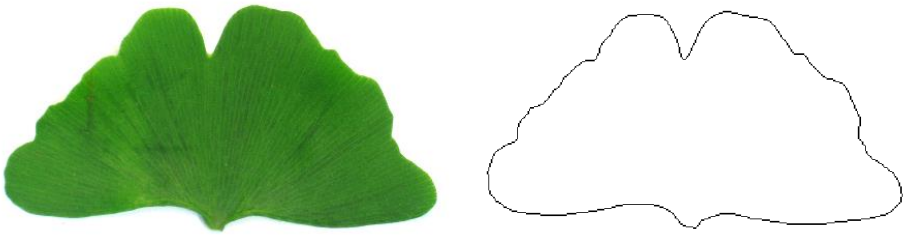

Figure A.1: A leaf from Ginkgo biloba (left) and its outline (right)

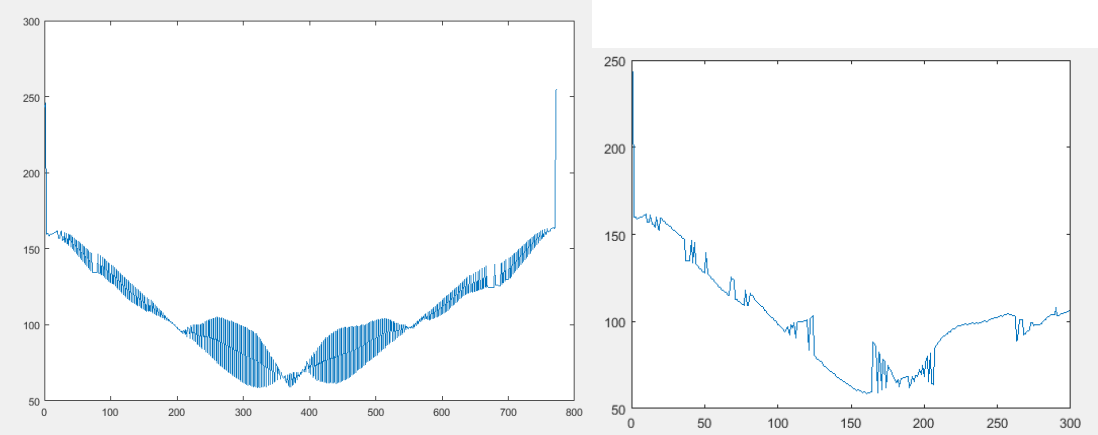

Figure A.2: The above leaf's time series with 773 points (left) and with 300 (right).

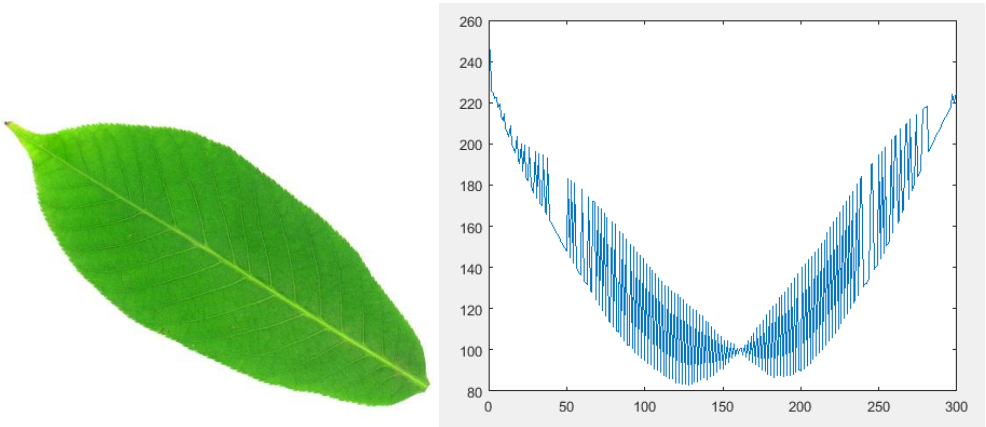

Figure A.3: The original leaf image (left) and its perimeter final time series (right).

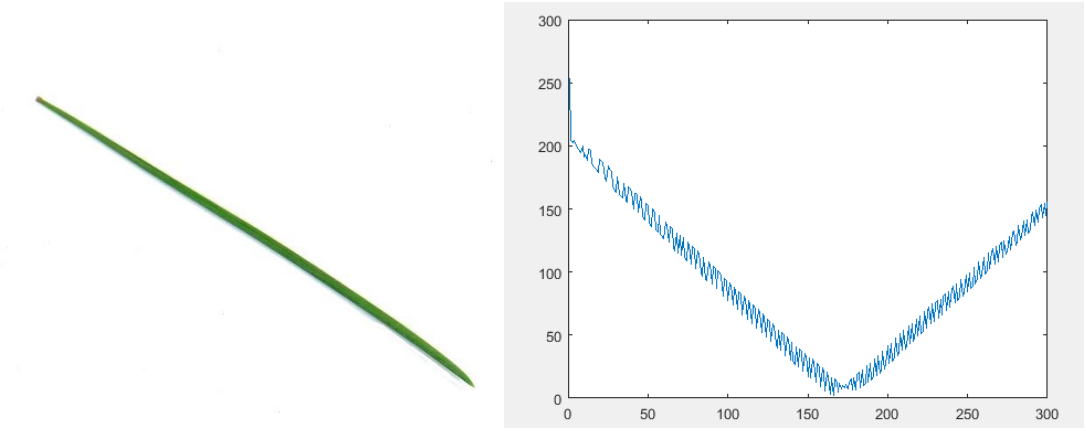

Figure A.4: The original image (left) and the corresponding time series (right).

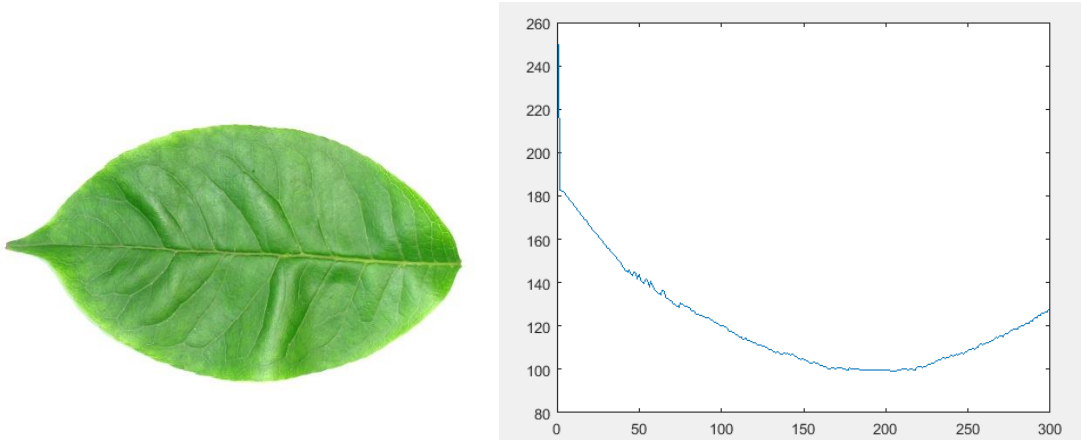

Figure A.5: The original image (left) and the corresponding time series (right).

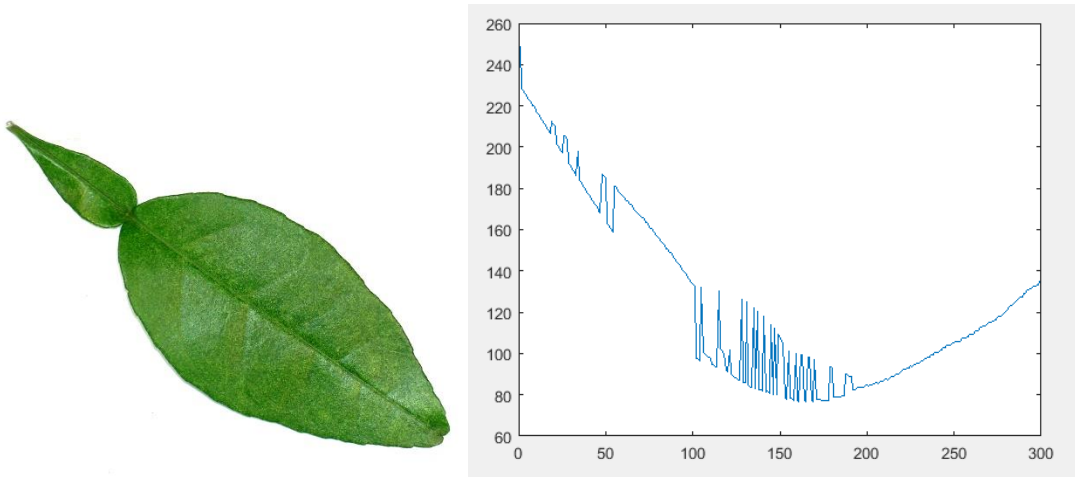

Figure A.6: The original image (left) and the corresponding time series (right).

APPENDIX B

Results of the symbolic representation of the processed time series

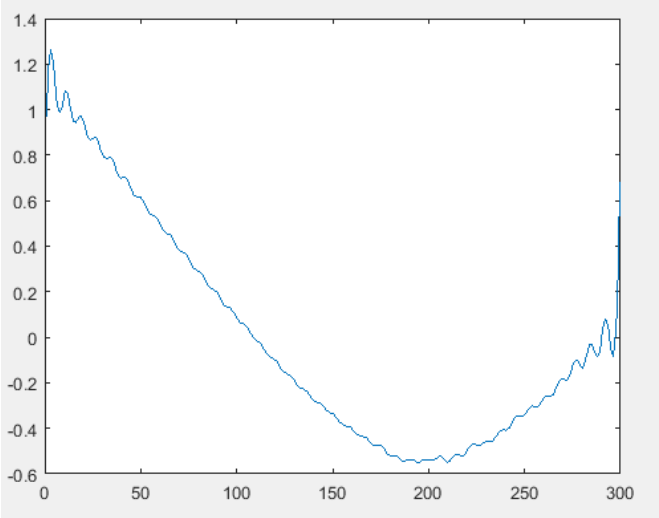

Figure B.1. A leaf from the species Berberis canadensis, commonly known as American barberry, (left) and the corresponding processed time series (right).

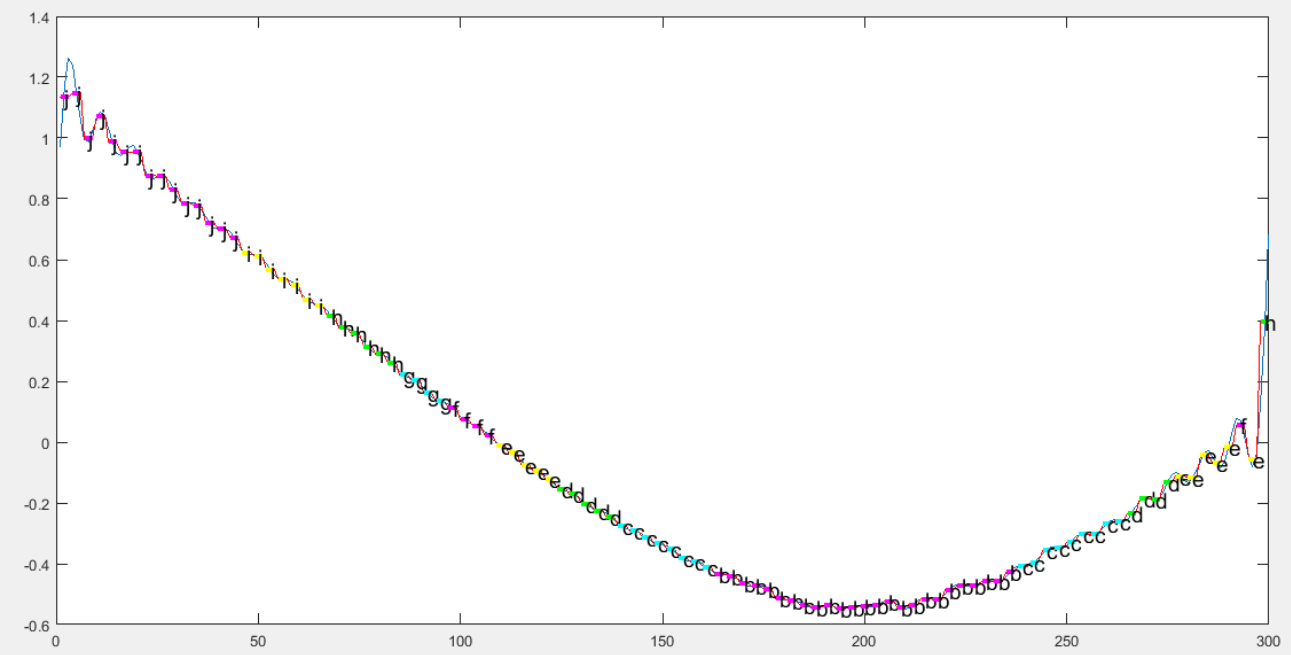

Figure B.2. The symbolic representation of the above time series. The result is a string with 100 characters.

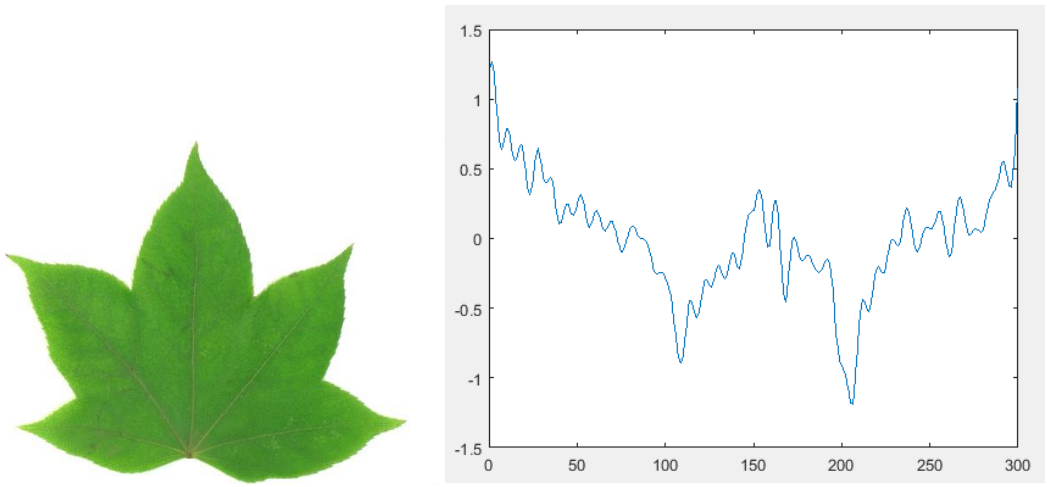

Figure B.3. A leaf from the species Kalopanax septemlobus (left) and the corresponding time series (right).

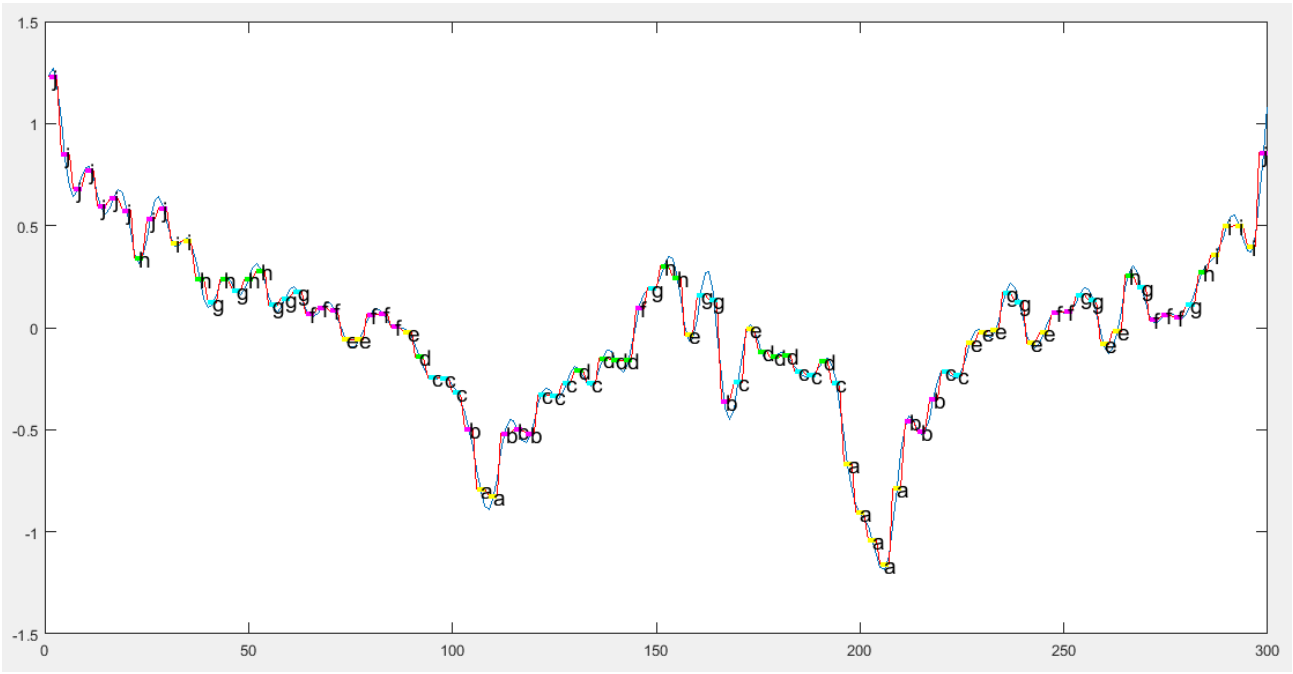

Figure B.4. The symbolic representation of the above time series.

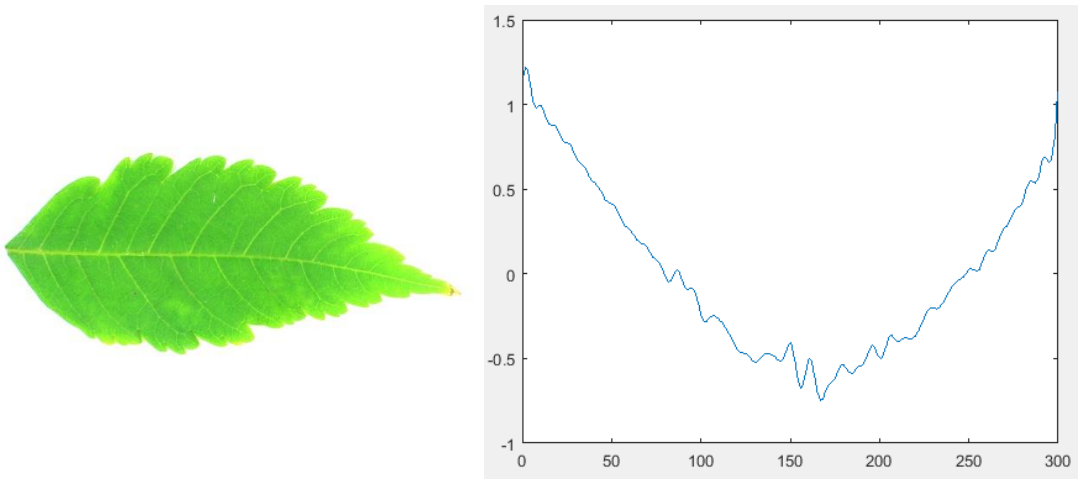

Figure B.5. A leaf from the species Koelreuteria paniculata (left) and the corresponding time series (right).

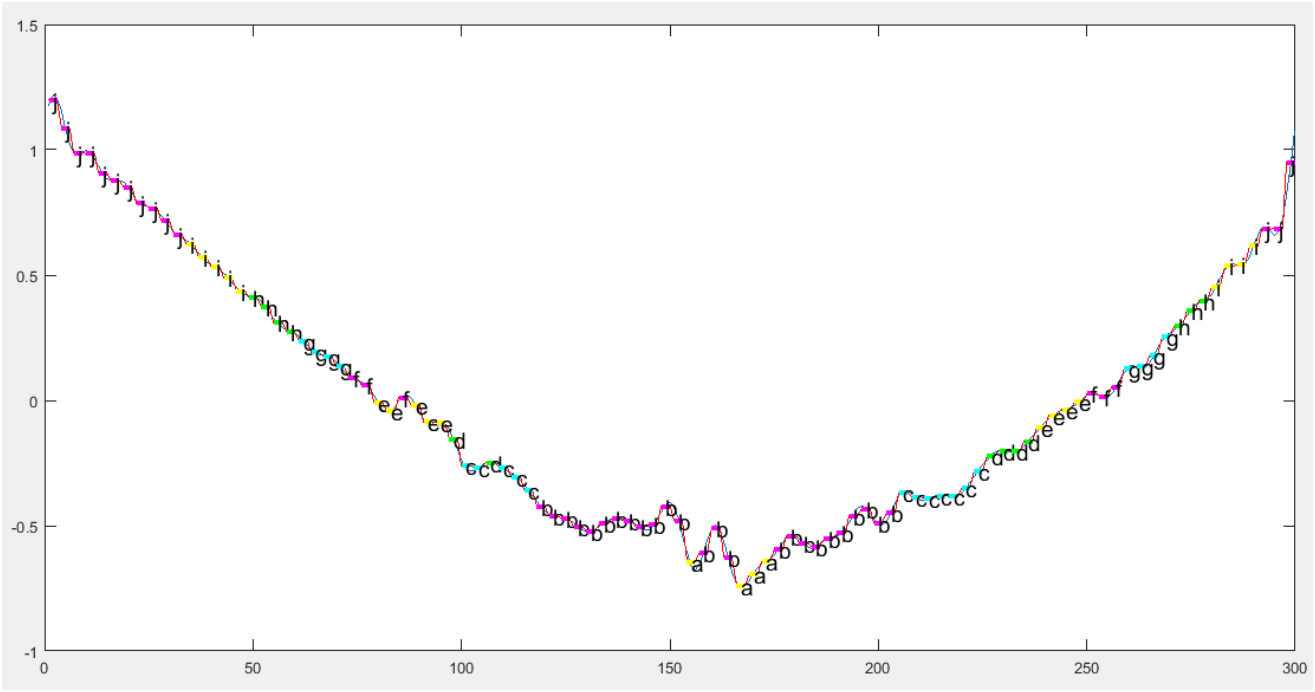

Figure B.6. The symbolic representation of the above time series.

APPENDIX C

Results from the interactive GUI

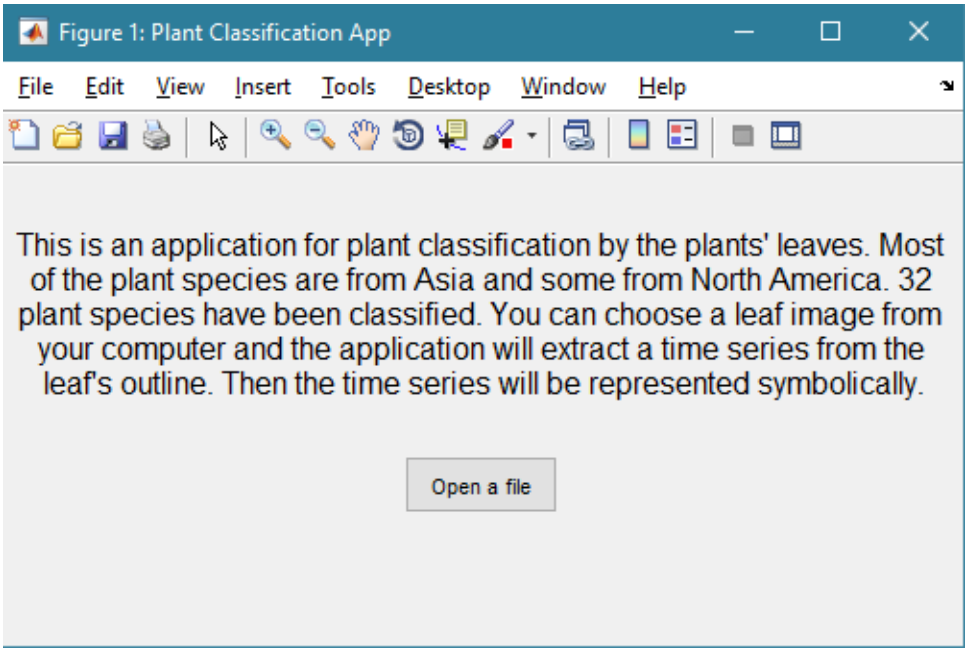

Figure C.1. The main screen of the application's GUI.

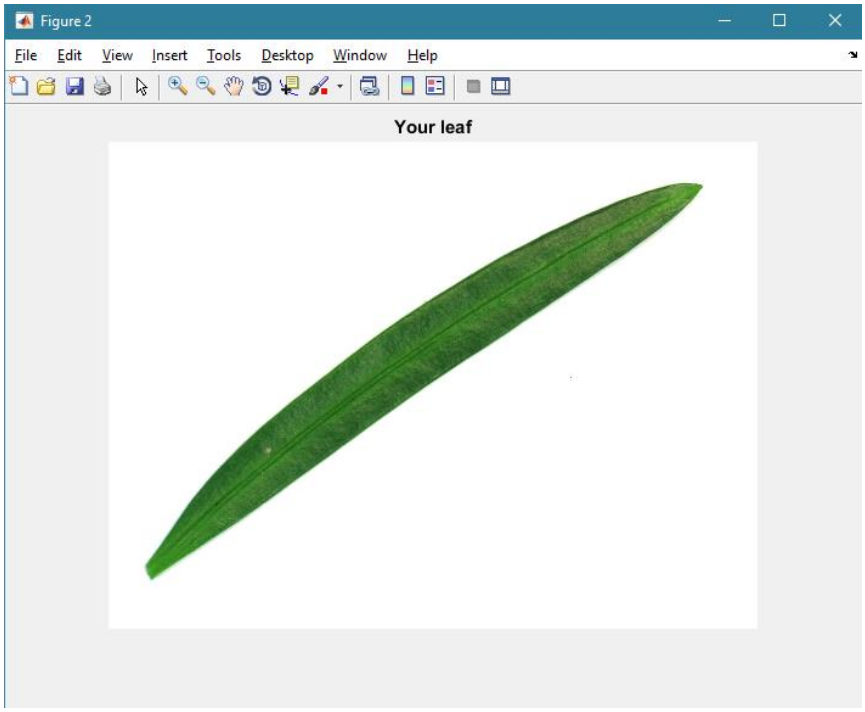

Figure C.2. The leaf that user has chosen, appears on the screen.

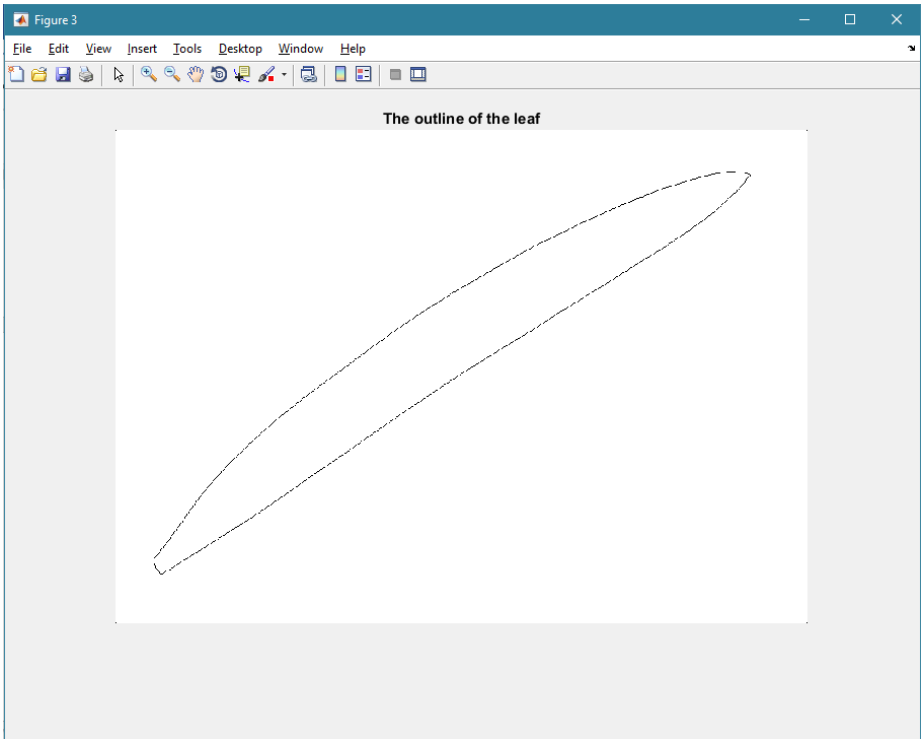

Figure C.3. The perimeter of the chosen leaf appears on the screen.

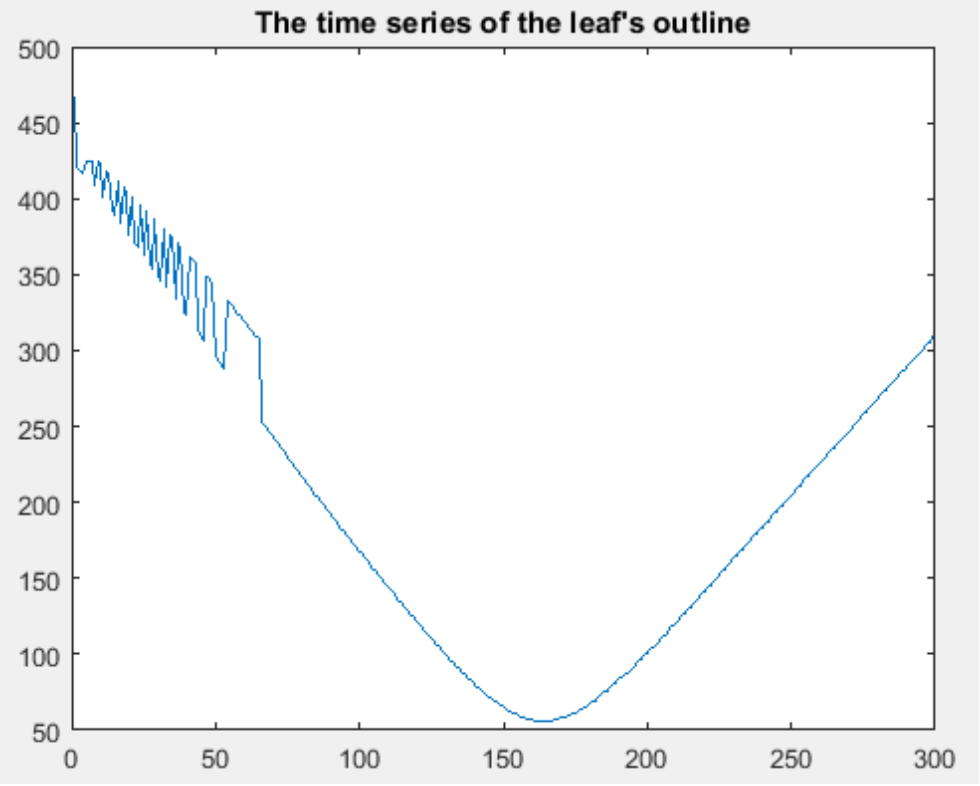

Figure C.4. A representation of the initial time series appears on the screen.

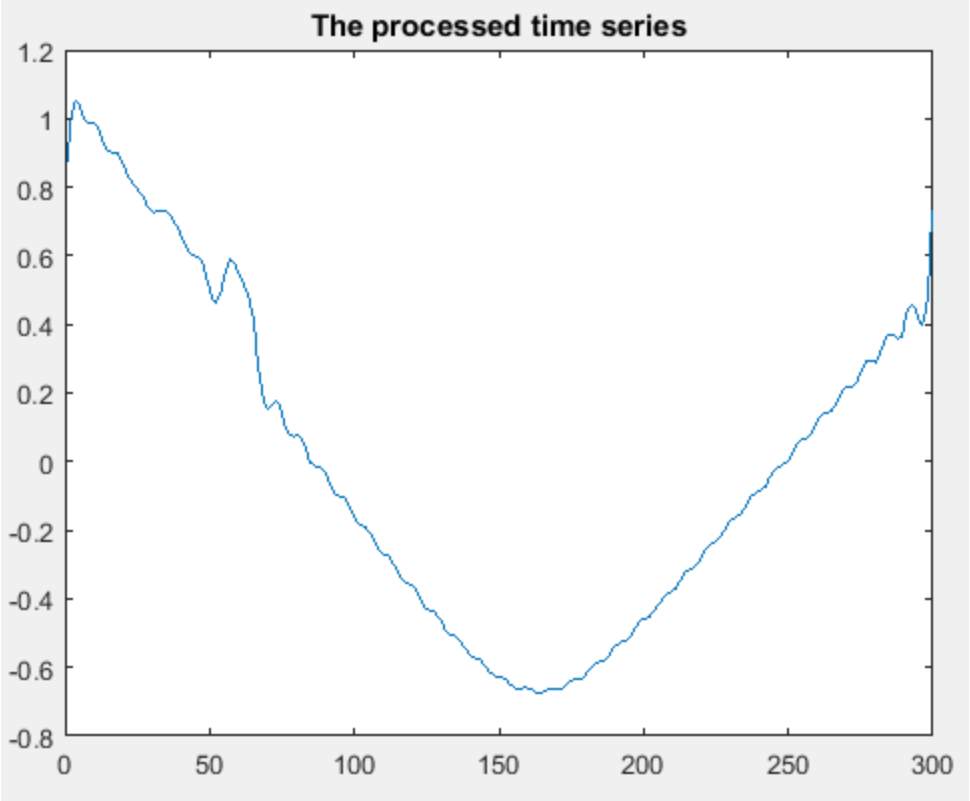

Figure C.5. A representation of the processed time series appears on the screen.

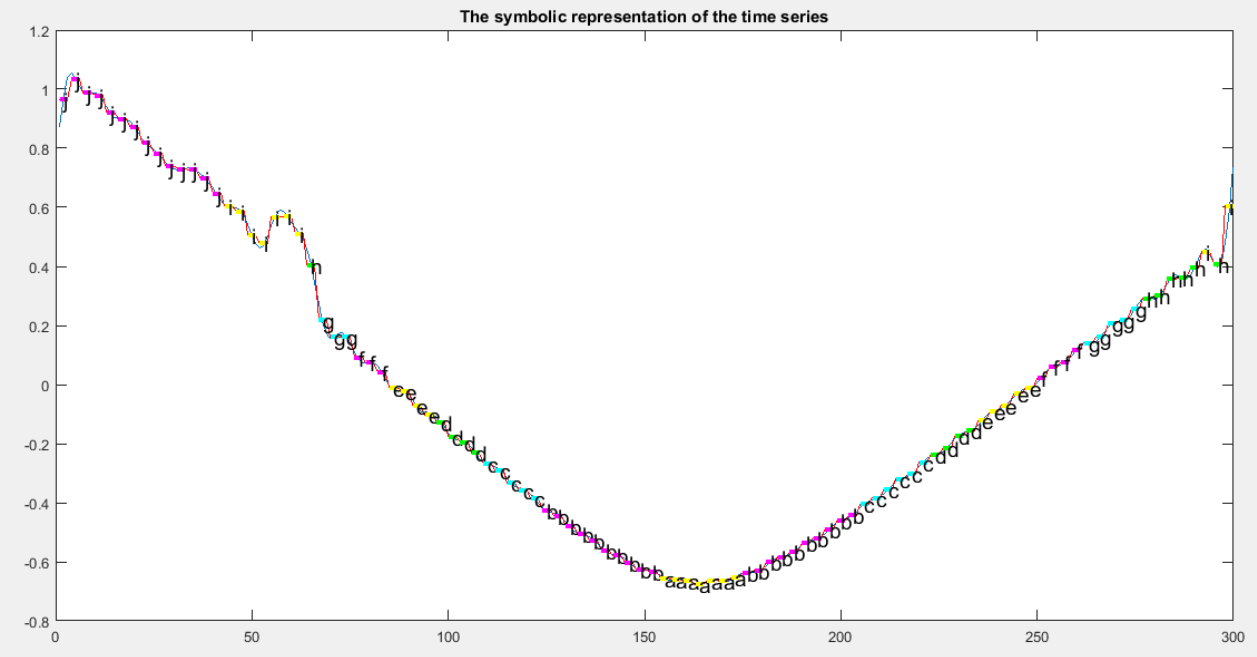

Figure C.6. The symbolic representation of the processed time series appears on the screen.

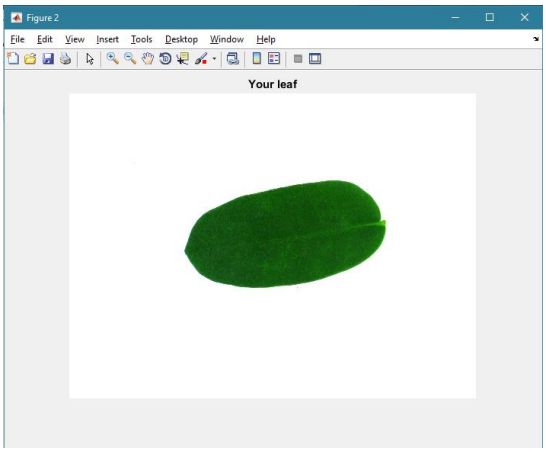

Figure C.7. Another example of the application.

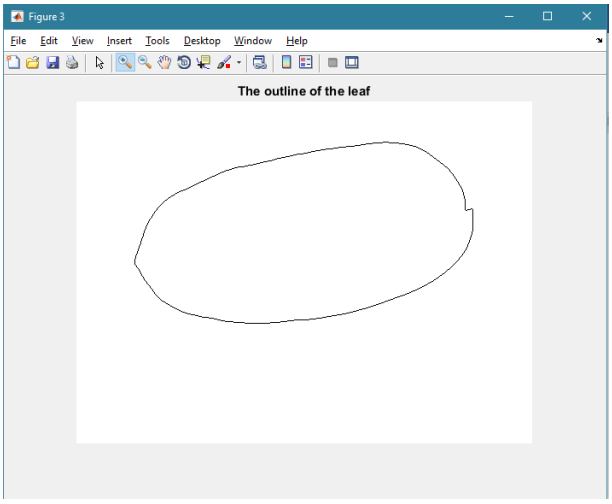

Figure C.8

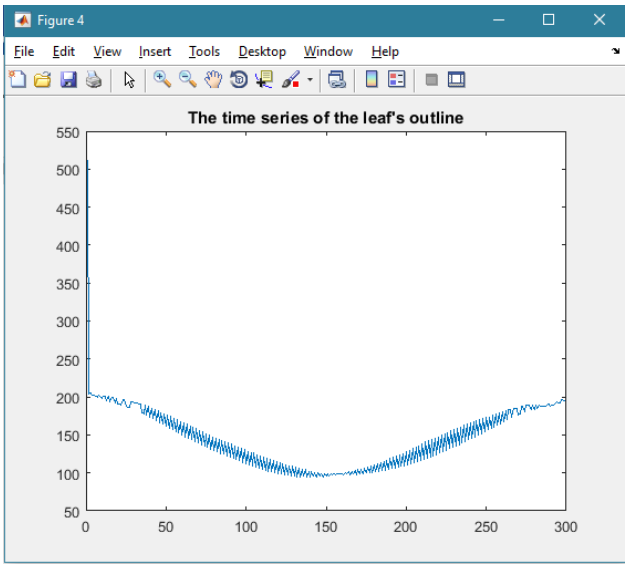

Figure C.9

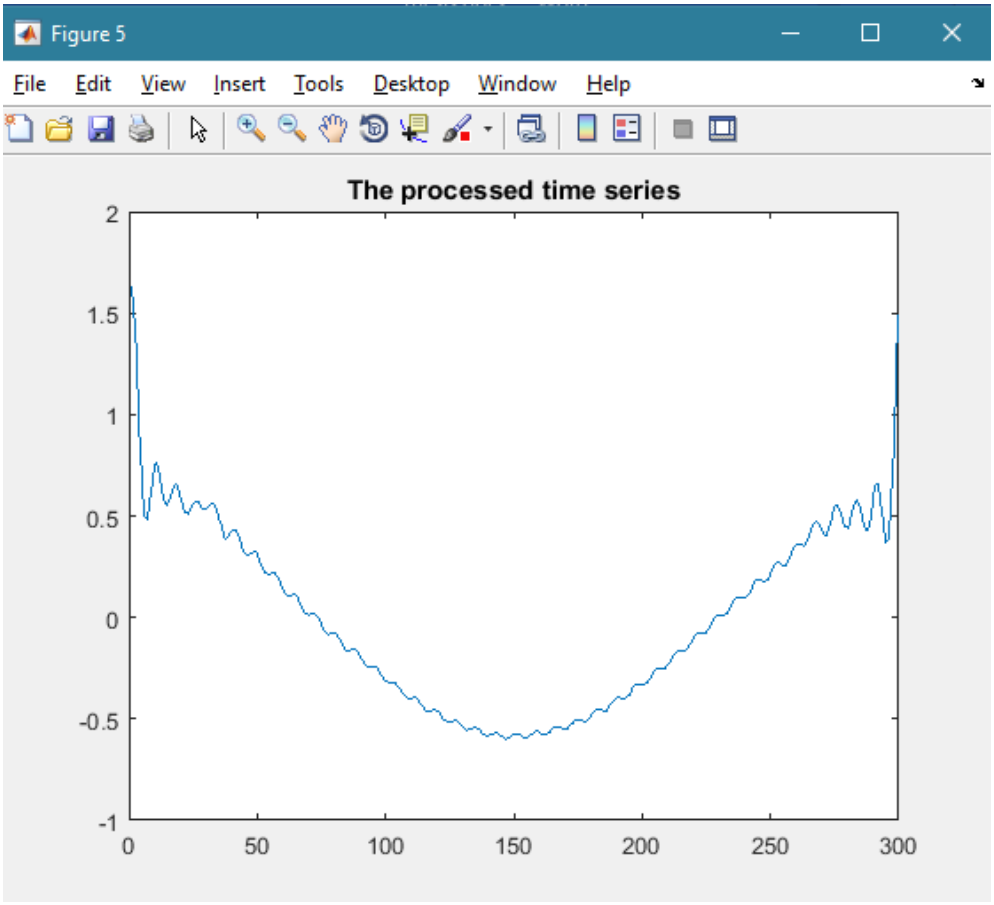

Figure C.10

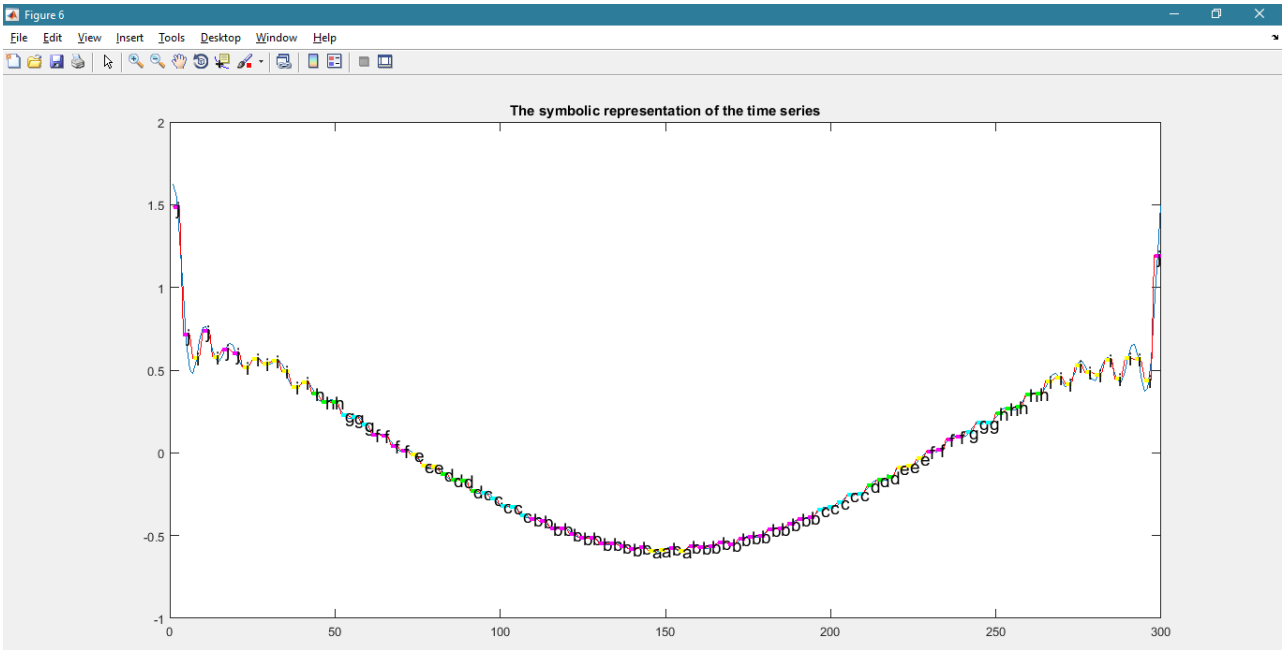
Figure 6
File Edit View Insert Tools Desktop Window Help
The symbolic representation of the time series
2
1.5
1
0.5
0
-0.5
-1
0
50
100
150
200
250
300

Figure C.11

APPENDIX D

Implemented methods

```
%Wu, S.G., Bao, F.S., Xu, E.Y., Wang, Y.X., Chang, Y.F. and Xiang, Q.L., 2007, December.
%A leaf recognition algorithm for plant classification using probabilistic neural network.
%In Signal Processing and Information Technology,
%2007 IEEE International Symposium on (pp. 11-16). IEEE.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ outline ] = LeafPerimeter( leaf )
    gray_leaf = rgb2gray(leaf);
    binary_leaf = im2bw(gray_leaf, 0.9);
    H1 = fspecial('average');
    smooth_leaf = imfilter(binary_leaf, H1);
    H2 = fspecial('laplacian');
    enhance_leaf = imfilter(smooth_leaf, H2, 'conv');
    outline = ~enhance_leaf;
    figure; imshow(outline);
    title('The outline of the leaf');
end
```

Figure D.1. A function that takes as input the leaf image and gives as output the leaf's outline. Code copyrighted by Wu et al. [4].

```
function [ distances ] = LeafCenter( perimeter, perim_prec )
    [points_x, points_y] = find(perimeter~=1);
    center_x = sum(points_x)./length(points_x);
    center_y = sum(points_y)./length(points_y);
    distances = sqrt((points_x-center_x).^2 + (points_y-center_y).^2);
    step = floor(length(distances)/perim_prec);
    max = length(distances) - mod(length(distances), perim_prec);
    distances = distances(1:step:max);
end
```

Figure D.2. A function that takes as input the outline of the identified leaf and the desired dimensionality of the time series that will be produced. The output is the time series.

```
%Vlachos, M., 2005. A practical time-series tutorial with matlab.
%In European conference on machine learning.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ final_time_s ] = preProcessing( initial_time_s )
    norm = (initial_time_s - mean(initial_time_s))/std(initial_time_s);
    ft = fft(norm);
    ft(40:end) = 0;
    reconstructed = real(ifft(ft));
    final_time_s = timeseries(reconstructed);
end
```

Figure D.3. This function takes as input a time series and process it by normalizing and decomposing it. Code copyrighted by Vlachos [22].

```
% Copyright (c) 2003, Eamonn Keogh, Jessica Lin, Stefano Lonardi, Pranav Patel, Li Wei. All rights reserved.
function [sax_string] = sax(data)
    seq_len = length(data);
    numCoeff = 100;
    alphabet_size = 10;

    % nseg must be divisible by data length
    if (mod(seq_len, numCoeff))
        disp('nseg must be divisible by the data length. Aborting ');
        return;

    end;

    % win_size is the number of data points on the raw time series that will be mapped to a
    % single symbol
    segLen = floor(seq_len/numCoeff);
    figure; plot(data);

    % special case: no dimensionality reduction
    if seq_len == numCoeff
        PAA = data;
    % Convert to PAA.  Note that this line is also in timeseries2symbol, which will be
    % called later.  So it's redundant here and is for the purpose of plotting only.
    else
        PAA = mean(reshape(data, segLen, numCoeff));
    end

    % plot the PAA segments
    PAA_plot = repmat(PAA, segLen, 1);
    PAA_plot = reshape(PAA_plot, 1, seq_len)';

    hold on;
    plot(PAA_plot,'r');

    % map the segments to string
    str = timeseries2symbol(data, seq_len, numCoeff, alphabet_size);
```

Figure D.4.1. A function that converts a time series into a SAX string using the PAA method. Code copyrighted by Keogh et al.

```
% get the breakpoints
switch alphabet_size
    case 2, cutlines  = [0];
    case 3, cutlines  = [-0.43 0.43];
    case 4, cutlines  = [-0.67 0 0.67];
    case 5, cutlines  = [-0.84 -0.25 0.25 0.84];
    case 6, cutlines  = [-0.97 -0.43 0 0.43 0.97];
    case 7, cutlines  = [-1.07 -0.57 -0.18 0.18 0.57 1.07];
    case 8, cutlines  = [-1.15 -0.67 -0.32 0 0.32 0.67 1.15];
    case 9, cutlines  = [-1.22 -0.76 -0.43 -0.14 0.14 0.43 0.76 1.22];
    case 10, cutlines = [-1.28 -0.84 -0.52 -0.25 0. 0.25 0.52 0.84 1.28];
    otherwise, disp('WARNING:: Alphabet size too big');
end;

% draw the gray guide lines in the background
%guidelines = repmat(cutlines, 1, seq_len);
%plot(guidelines', 'color', [0.8 0.8 0.8]);
hold on

%pause;

color = {'g', 'y', 'm', 'c'};
symbols = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j'};

% high-light the segments and assign them to symbols
for i = 1 : numCoeff

    % get the x coordinates for the segments
    x_start = (i-1) * segLen + 1;
    x_end   = x_start + segLen - 1;
    x_mid   = x_start + (x_end - x_start) / 2;

    % color-code each segment
    colorIndex = rem(str(i),length(color))+1;

    % draw the segments
    plot([x_start:x_end],PAA_plot([x_start:x_end]), 'color', color{colorIndex}, 'linewidth', 3);


    % show symbols
    text(x_mid, PAA_plot(x_start), symbols{str(i)}, 'fontsize', 14);
end

sax_string = symbols(str);
```

Figure D.4.2. A function that converts a time series into a SAX string using the PAA method. Code copyrighted by Keogh et al.

```
% Copyright (c) 2003, Eamonn Keogh, Jessica Lin, Stefano Lonardi, Pranav Patel.  All rights reserved.
function [symbolic_data, pointers] =  sax_modified(data, N, n, alphabet_size, NR_opt)

if nargin < 4
    disp('usage: sax_modified(data, window_len, num_segment, alphabet_size, [numerosity_reduction_option]');
    return;
end

if alphabet_size > 20
    disp('Currently alphabet_size cannot be larger than 20.  Please update the breakpoint table if you wish to do so');
    return;
end

if nargin < 5
    NR_opt = 2;
end

win_size = floor(N/n);   % win_size is the number of data points on the raw time series that will be mapped to a single symbol

pointers         = [];        % Initialize pointers,
symbolic_data = zeros(1,n);   % Initialize symbolic_data with a void string, it will be removed later.
all_string = zeros(length(data)-N+1,n);


% Scan accross the time series extract sub sequences, and converting them to strings.
for i = 1 : length(data) - (N -1)

    if mod(i, 1000) == 0
        disp(num2str(i));
    end

    % Remove the current subsection.
    sub_section = data(i:i + N -1);

    % Z normalize it.
    sub_section = (sub_section - mean(sub_section))/std(sub_section);
```

Figure D.5.1. A function called timeseries2symbol converts the input time series into a sequence of symbols, according to an alphabet size and a desired length of the output string. Code copyrighted by Keogh et al.

```
    % take care of the special case where there is no dimensionality reduction
    if N == n
        PAA = sub_section;

    % N is not dividable by n
    else
        if (N/n - floor(N/n))
            temp = zeros(n, N);
            for j = 1 : n
                temp(j, :) = sub_section;
            end
            expanded_sub_section = reshape(temp, 1, N*n);
            PAA = [mean(reshape(expanded_sub_section, N, n))];
        % N is dividable by n
        else
            PAA = [mean(reshape(sub_section,win_size,n))];
        end
    % Convert to PAA.
    %else
   %     PAA = [mean(reshape(sub_section,win_size,n))];
    end

    current_string = map_to_string(PAA,alphabet_size);          % Convert the PAA to a string.

    % no numerosity reduction: record everything
    if NR_opt == 1
        symbolic_data    = [symbolic_data; current_string];     % ... add it to the set...
        pointers         = [pointers ; i];                      % ... and add a new pointer.

    % with numerosity reduction: record a string only if it differs from its leftmost neighbor
    elseif NR_opt == 2

        if ~all(current_string == symbolic_data(end,:))             % If the string differs from its leftmost neighbor...
            symbolic_data    = [symbolic_data; current_string];     % ... add it to the set...
            pointers         = [pointers ; i];                      % ... and add a new pointer.
        end;
```

Figure D.5.2. The timeseries2symbol function. Code copyrighted by Keogh et al.

```
    % advanced numerosity reduction: record a string only if its mindist to the last recorded
    % string > 0
    elseif NR_opt == 3

        % always record the first string
        if i == 1
            symbolic_data    = [symbolic_data; current_string];     % ... add it to the set...
            pointers         = [pointers ; i];                      % ... and add a new pointer.

        % subsequent strings
        else

            % we only need to check if two sliding windows have different strings (if they are
            % the same then their mindist is 0)
            if ~all(current_string == symbolic_data(end,:))            % If the string differs from its leftmost neighbor...

                % Here we're doing a simplified version of mindist. Since we're only interested
                % in knowing if the distance of two strings is 0, we can do so without any extra
                % computation.  Since only adjacent symbols have distance 0, all we have to
                % do is check if any two symbols are non-adjacent.
                if any(abs(symbolic_data(end,:) - current_string) > 1)
                    symbolic_data    = [symbolic_data; current_string];     % ... add it to the set...
                    pointers         = [pointers ; i];                      % ... and add a new pointer.
                end
            end
        end

    else


        % we only need to check if two sliding windows have different strings (if they are
        % the same then their mindist is 0)
        if ~all(current_string == symbolic_data(end,:))            % If the string differs from its leftmost neighbor...
            if any(abs(symbolic_data(end,:) - current_string) > 1)
                if ~all(sign(diff(current_string)) >= 0) & ~all(sign(diff(current_string)) <= 0)
                    symbolic_data    = [symbolic_data; current_string];     % ... add it to the set...
                    pointers         = [pointers ; i];                      % ... and add a new pointer.
                end
            end
        end
    end

end;

% Delete the first element, it was just used to initialize the data structure
symbolic_data(1,:) = [];
```

Figure D.5.3. The timeseries2symbol function. Code copyrighted by Keogh et al.

```
%---------------------------------------------------------------------------------------------------------------------------------------------------
%---------------Local Functions---------------------Local Functions---------------Local Functions---------------------Local Functions-----
%---------------------------------------------------------------------------------------------------------------------------------------------------

function string = map_to_string(PAA,alphabet_size)

string = zeros(1,length(PAA));

switch alphabet_size
        case 2, cut_points  = [-inf 0];
        case 3, cut_points  = [-inf -0.43 0.43];
        case 4, cut_points  = [-inf -0.67 0 0.67];
        case 5, cut_points  = [-inf -0.84 -0.25 0.25 0.84];
        case 6, cut_points  = [-inf -0.97 -0.43 0 0.43 0.97];
        case 7, cut_points  = [-inf -1.07 -0.57 -0.18 0.18 0.57 1.07];
        case 8, cut_points  = [-inf -1.15 -0.67 -0.32 0 0.32 0.67 1.15];
        case 9, cut_points  = [-inf -1.22 -0.76 -0.43 -0.14 0.14 0.43 0.76 1.22];
        case 10, cut_points = [-inf -1.28 -0.84 -0.52 -0.25 0. 0.25 0.52 0.84 1.28];
        case 11, cut_points = [-inf -1.34 -0.91 -0.6 -0.35 -0.11 0.11 0.35 0.6 0.91 1.34];
        case 12, cut_points = [-inf -1.38 -0.97 -0.67 -0.43 -0.21 0 0.21 0.43 0.67 0.97 1.38];
        case 13, cut_points = [-inf -1.43 -1.02 -0.74 -0.5 -0.29 -0.1 0.1 0.29 0.5 0.74 1.02 1.43];
        case 14, cut_points = [-inf -1.47 -1.07 -0.79 -0.57 -0.37 -0.18 0 0.18 0.37 0.57 0.79 1.07 1.47];
        case 15, cut_points = [-inf -1.5 -1.11 -0.84 -0.62 -0.43 -0.25 -0.08 0.08 0.25 0.43 0.62 0.84 1.11 1.5];
        case 16, cut_points = [-inf -1.53 -1.15 -0.89 -0.67 -0.49 -0.32 -0.16 0 0.16 0.32 0.49 0.67 0.89 1.15 1.53];
        case 17, cut_points = [-inf -1.56 -1.19 -0.93 -0.72 -0.54 -0.38 -0.22 -0.07 0.07 0.22 0.38 0.54 0.72 0.93 1.19 1.56];
        case 18, cut_points = [-inf -1.59 -1.22 -0.97 -0.76 -0.59 -0.43 -0.28 -0.14 0 0.14 0.28 0.43 0.59 0.76 0.97 1.22 1.59];
        case 19, cut_points = [-inf -1.62 -1.25 -1 -0.8 -0.63 -0.48 -0.34 -0.2 -0.07 0.07 0.2 0.34 0.48 0.63 0.8 1 1.25 1.62];
        case 20, cut_points = [-inf -1.64 -1.28 -1.04 -0.84 -0.67 -0.52 -0.39 -0.25 -0.13 0 0.13 0.25 0.39 0.52 0.67 0.84 1.04 1.28 1.64];
        otherwise disp('Error! alphabet_size is too big');
end;

for i = 1 : length(PAA)
    string(i) = sum( (cut_points <= PAA(i)), 2 );        % order is now: a = 1, b = 2, c = 3..
end;
```

Figure D.5.4. A local function in timeseries2symbol. Code copyrighted by Keogh et al.