Tampere University

Tuomo Hartikainen

FEATURE SELECTION IN HIGH-DIMENSIONAL FEATURE SPACES FOR TREE SPECIES CLASSIFICATION FROM QUANTITATIVE STRUCTURE MODELS

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Examiners: Assoc. prof., Pasi Raumonen
D.Sc. (Tech.), Markku Åkerblom
November 2020

ABSTRACT

Tuomo Hartikainen: Feature selection in high-dimensional feature spaces for tree species classification from quantitative structure models
Master of Science Thesis
Tampere University
Master's Programme in Science and Engineering
November 2020

The possibility for computation of thousands of structural tree features from Quantitative Structure Models (QSMs) has unlocked new potential for statistical tree species classification of trees. Previously it has only been done using a dozen or so features and the classification accuracy has been limited by those features' distinctiveness in the species that are classified. Since every possible combination of those features could be tested in practice, the focus has been on testing and optimizing different classification methods. With thousands of features it is no longer possible to test all feature combinations and thus the focus of this work is on feature selection.

A method for selecting multiple feature sets for classification of each species separately was developed. The method, called decimated input ensembles for one vs all classification (DIE1VA), is a filter-wrapper hybrid that generates a user-defined number of feature sets for each class to perform One-vs-All classification and determine the final prediction by majority voting. Initial feature sets are generated based on filter rankings and the ones with the highest accuracy are refined by adding features one by one if they improve accuracy. The method is able to utilize more of the abundance of data available than just selecting one set of features to classify all species and achieves higher accuracy as a result. The method's parameters affect its computational time and the resulting accuracy of the classifier.

In about 24 hours the method selected feature sets that could be used to classify five tree species with total accuracy of 91.5% and producer accuracies of at least 64% for each species compared to 80% total accuracy or 57% producer accuracies on the same data when choosing from 17 features. Common finnish species Pine, Spruce and Birch could be classified with 98.5% accuracy after about an hour of feature selection. Five tropical species could be classified with 84.9% accuracy in 4 hours and 20 minutes.

A few of the parameters of the method were tested with different values but due to having many parameters that also affect each other in some ways, optimizing and researching the method thoroughly for different kinds of applications and data would require more time. There are also ways to improve the method that could be worth researching, such as changing certain parameters or observation weights during the method's execution. This study proves that in some cases there is added value in having thousands of features instead of a dozen or so to choose from for classification of trees and the flexibility of the method in addition to the accuracies that it achieves mean that it could be used in a number of practical applications.

Keywords: feature selection, high-dimensional feature spaces, classification, quantitative structure models, species recognition

TIIVISTELMÄ

Tuomo Hartikainen: Ominaisuuksien valinta korkeadimensioisissa ominaisuusavaruuksissa puulajien luokitteluun kvantitatiivisista rakennemalleista
Diplomityö
Tampereen yliopisto
Teknis-luonnontieteellinen tutkinto-ohjelma
Marraskuu 2020

Kvantitatiiviset rakennemallit mahdollistavat tuhansien rakenteellisten ominaisuuden laskemisen puista, joka puolestaan avaa uusia mahdollisuuksia puiden aikaisempaa tarkempaan tilastolliseen luokitteluun. Aikaisemmissa tutkimuksissa puiden luokitteluun on käytetty noin tusinaa ominaisuutta ja luokittelutarkkuutta on rajannut se, kuinka hyvin ne erottavat kullekin lajille ominaiset piirteet. Koska kaikki mahdolliset kombinaatiot näistä ominaisuuksista on ollut mahdollista testata käytännössä, tutkimus on keskittynyt testaamaan ja optimoimaan eri luokittelumetodeja. Tuhansien ominaisuuksien tapauksessa on mahdotonta testata kaikkia mahdollisia kombinaatioita ja siksi tämä työ keskittyy ominaisuuksien valintaan.

Kehitetty metodi valitsee useamman ominaisuusjoukon jokaisen luokan luokitteluun. Metodi, nimeltään desimoidut syötekokonaisuudet yksi vastaan kaikki luokitteluun on suodatin-kääre hybridi, joka tuottaa käyttäjän määrittelemän määrän ominaisuusjoukkoja jokaiselle luokalle suorittamaan yksi vastaan kaikki luokittelun, jonka tulokset määrittelevät lopullisen ennustuksen enemmistöäänestyksellä. Alustavat ominaisuusjoukot tuotetaan suodatinsijoitusten perusteella ja niitä, joilla on suurin tarkkuus, jalostetaan lisäämällä ominaisuuksia yksi kerrallaan jos ne parantavat tarkkuutta. Täten metodi käyttää enemmän dataa hyödyksi kuin jos käytettäisiin vain yhtä ominaisuusjoukkoa kaikkien luokkien luokitteluun, saavuttaen korkeamman luokittelutarkkuuden sen seurauksena. Metodin parametrit vaikuttavat sen käyttämään laskennalliseen aikaan ja lopullisen luokittelijan tarkkuuteen.

Noin 24 tunnissa metodi valitsee ominaisuusjoukot, jotka luokittelevat viisi puulajia 91.5% tarkkuudella niin, että jokainen yksittäinen laji luokitellaan vähintään 64% tarkkuudella. Tämä on huomattava parannus aikaisempiin tutkimuksiin, joissa 17 ominaisuudesta valitsemalla saavutettiin 80% kokonaistarkkuus tai vähintään 57% tarkkuus jokaiselle lajille samaa dataa käyttäen. Suomessa yleiset lajit mänty, kuusi ja koivu saatiin luokiteltua 98.5% tarkkuudella tunnissa. Viisi trooppista lajia saatiin luokiteltua 84.9% tarkkuudella 4 tunnissa ja 20 minuutissa.

Muutamien metodin parametrien vaikutusta testattiin, mutta koska parametrejä on monta ja ne myös vaikuttavat toisiinsa, perusteellinen metodin tutkiminen ja optimointi eri sovelluksille ja datalle vaatisi enemmän aikaa. Metodia voi vielä parantaa tavoilla jotka voisivat olla tutkimuksen arvoisia, esimerkiksi joitain parametrejä voisi muuttaa metodin suorituksen aikana. Tämä tutkimus todistaa, että joissain tapauksissa siitä että luokitteluun voi valita tuhansista ominaisuuksista reilun tusinan sijaan on lisäarvoa puiden luokitteluun. Kehitetyn metodin joustavuuden ja sen saavuttamien tarkkuuksien ansiosta sitä voitaisiin käyttää monenlaisissa sovelluksissa.

Avainsanat: ominaisuuksien valinta, korkeadimensioiset ominaisuusavaruudet, luokittelu, lajintunnistus, kvantitatiiviset rakennemallit

PREFACE

Developing a method for automatic tree species classification proved to be an interesting and worthwhile challenge and I am grateful for the opportunity to do it paid for my home town university. A great deal of work that was done in the field of tree species classification prior to this work contributed to making it easy to focus on what was essential to classification in high-dimensional feature spaces, namely feature selection. This thesis was written between April and November 2020 in Tampere University.

The biggest enabler to this thesis was my instructor, manager and examiner Pasi Raumonen, who offered the job to do it and guided me throughout the process of doing this thesis. Large parts of the text explaining QSMs in Chapter 3 were provided by him as well as the MATLAB script that computes over 13000 features from QSMs. In addition to Pasi Raumonen, Markku Åkerblom also provided lots of comments and suggestions for improvement on the text and is largely responsible for it being more scientific than it would have otherwise. Dr. Kim Calder (Ghent University), Prof. Raisa Mäkipää (Natural Resources Institute Finland) and Dr. Olivier Martin-Ducup (AMAP, University Montpellier) provided the data sets used in the experiments.

In Tampere, 19th November 2020

Tuomo Hartikainen

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

C	set of cannot-link constraints
E	allowed error in support vector machines
H_j	nearest hits in ReliefF
$I(x, y)$	mutual information of two random variables x and y
M	set of must-link constraints
M_{SVM}	margin used in support vector machines
$M_j(C)$	nearest misses of an observation of class C in ReliefF
R_i	randomly selected observation
S	subset of features that we are seeking
Z	reduced data space used in Fisher score
Ω	set of all features
Ω_S	set of features not yet selected by mRMR
$\tilde{\boldsymbol{\mu}}$	overall mean vector in reduced data space
$\tilde{\boldsymbol{\mu}}_k$	mean vector of kth class in reduced data space
$\mathbf{I}$	the identity matrix
$\tilde{\mathbf{S}}_b$	between-class scatter matrix
$\tilde{\mathbf{S}}_t$	total scatter matrix
$\mathbf{z}_i$	the ith observation
ν	number of degrees in freedom in χ^2 distribution
ρ	correlation
σ	standard deviation
a	number of fitted parameters in χ^2 distribution
c	number of classes
d	a distance metric
f	numerical feature
h	classification variable
k	number of nearest neighbors
m	user-defined parameter of how many times ReliefF process should be repeated

m_{HFS}	size of the block in HFS
n	number of feature subsets for each class's One-vs-All classifier
n_C	number of observations of class C
n_{HFS}	number of top feature sets that are combined with the rest to form new feature sets in HFS
n_{obs}	number of observations
n_b	number of subsequent blocks HFS is executed for
n_c	number of candidate features to look for before choosing
n_f	number of features
n_k	number of observations of kth class
n_s	number of selected features
p, P	probability (in the context of classification, probability of correct classification)
x_e	end point of a reference distribution
x_m	middle point of a reference distribution
x_s	starting point of a reference distribution
DBH	diameter at breast height
DIE1VA	decimated input ensembles for one vs all classification
DR	dimension reduction
FE	feature extraction
FS	feature selection
HFS	hybrid method for feature selection
Inf	infinity
kNN	k-nearest neighbors
MID	mutual information difference
MIQ	mutual information quotient
mRMR	minimum redundancy maximum relevance algorithm
NaN	not a number
PC	principal component
PCA	principal component analysis
PPV	positive predictive value
QSM	quantitative structure model
SVM	support vector machine

1 INTRODUCTION

Statistical classification allows us to categorise new observations based on data we already have about previous observations where the category is known. It can be used to automate tasks and sometimes even achieves a higher accuracy in categorising than experts of the field. The data we have about observations consists of the same set of features that have been measured for every observation. This set of features is called the feature space and every observation is represented by a point in that space. The categories are called classes and predicting the class of a new observation based on the data from observations where the class is known is called classification.

One example of statistical classification is classifying tree species based on features computed from quantitative structure models (QSMs). In previous research the approach has been used to define a small number of features and to test and optimize a few classifiers that classify all of the species [1, 2]. Since there have been a small number of features (15-17) it has been possible to test all possible feature combinations in reasonable time. But since QSMs make it possible to compute thousands of features, the approach of this thesis is to focus on developing a feature selection (FS) method for high-dimensional feature spaces, where testing all feature combinations is no longer possible due to possible time constraints. Having thousands of features instead of a dozen or so (assuming that the features are sensible) offers great potential to improve classification accuracy if we are able to select the most relevant features.

The main objectives of this thesis are the following. First is to develop a useful FS and classification method for tree species classification from high-dimensional QSM feature spaces. Second is to test and evaluate the method with data that has been previously used in [1, 2] and see if, and by how much, classification accuracy improved by selecting from thousands of features instead of a dozen or so.

In general there are three kinds of FS methods, filters, wrappers and embedded methods [3]. Filters use intrinsic qualities in the data to rank the features fast, but do not consider the classification method used when selecting features. Wrappers evaluate the performance of candidate subsets of features with the chosen classification method, which makes them slower, but able to reach higher accuracies than filters in practice. In high-dimensional feature spaces it is practically necessary to use some kind of a filter in order to keep time consumption sensible, but wrappers can be combined with filters to achieve higher accuracies than with filters alone [4].

The abundance of features offers many different perspectives through which we can look at tree species classification in the form of possible feature sets that we can select for the classifier to use. Instead of there being one clearly optimal set of features to use in classifying all trees for example, we will see that there are many feature sets that are close to optimal in terms of the accuracy and that can correctly classify some of the observations that the optimal set misclassifies. The feature sets that are best suited for distinguishing a particular species from the rest also differ between species.

Taking all of this into account, we will present a method that utilizes more data to achieve a higher classification accuracy by selecting multiple feature sets for classifying each species instead of selecting one subset of features for classifying all of the species. The method is a filter-wrapper hybrid that uses so-called One-vs-All classifiers that are built to distinguish one species from the rest, classifying observations either as that species or *other*. For each species, the same number of feature sets are selected and the One-vs-All classifiers vote on the final prediction using those feature sets.

The theoretical background of feature selection and classification methods used in this thesis is presented in Chapter 2. The method and the data used in the experiments is presented in Chapter 3. In Chapter 4 the results of the experiments using the method on different datasets with different parameters, classification methods and filters are presented. In Chapter 5 the results are analyzed and compared to previous work and the implications and possibilities for further research are discussed. Lastly, conclusions are presented in Chapter 6.

2 THEORETICAL BACKGROUND

To be able to statistically classify objects, we need to have information about them as well as already classified members of possible classes they belong to. To get data for classification, we measure some *features* of the objects we want to classify. All of the features measured from one object constitutes an *observation*, and a collection of observations is called a *sample*. If the number of features we measure is n_f, we can think of observations as points in a n_f-dimensional space called a *feature space*. With n_{obs} observations, the data can be represented by a $(n_f \times n_{\text{obs}})$-matrix.

When it comes to classification problems in high-dimensional feature spaces (thousands of features), extracting the most useful information or selecting the most relevant features is an important step, because using all of the features increases computational time and can (and usually does) degrade accuracy [4]. This phenomenon is also known as the *curse of dimensionality* [4]. *Dimension reduction* (DR) methods are used to reduce the number of features so that the classification can be done efficiently and accurately [4].

There are generally two kinds or DR methods: *feature selection* (FS) and *feature extraction* (FE) [5, 6]. In FS dimensionality is reduced by selecting a subset of the initial features to be used by the classifier. The rest of the features are discarded. The problem is determining which features are the most relevant to the classification problem at hand. One benefit of this is that the selected features are clearly defined, and could provide insight into what are the structural features that differentiate certain species of trees, for example. In FE the original feature space is projected into a lower dimensional subspace. As such, some information about most of the original features is also present in the new subspace, but the extracted features are often linear combinations of the original features and thus harder to interpret. Most FE methods have high time complexity and are not as viable for data sets with thousands of features. The most common FE method — and the one used in our experiments — is called *principal component analysis* (PCA).

2.1 Principal Component Analysis

Principal component analysis transforms a data set consisting of interrelated variables to a new set of variables called *principal components* (PCs), which are uncorrelated and retain all of the variation in the data set. The PCs are ordered in terms of the variation they retain, such that starting from the first PC we can add PCs until some threshold percentage of variation we want retained is explained by as few PCs as possible. Let us represent the data by an $n_f \times n_{\text{obs}}$ matrix $\mathbf{X}$ where each row represents a feature and each column represents an observation. First we find a vector α_1 of n_f constants $\alpha_{11}, \alpha_{12}, \ldots, \alpha_{1n_f}$ that maximizes the variance in the data when $\mathbf{X}$ is mapped to the first principal component

$$\mathbf{t}_1 = \alpha_1 \mathbf{X}. \tag{2.1}$$

Then we subtract the first principal component from the data matrix $\mathbf{X}$

$$\mathbf{X}_1 = \mathbf{X} - \mathbf{X}\alpha_1\alpha_1^T \tag{2.2}$$

and find the vector α_2 that is orthogonal to α_1 and maximizes the variance in $\mathbf{t}_2 = \alpha_2\mathbf{X}_1$. The process repeats so that the ith principal component is the vector

$$\mathbf{t}_i = \alpha_i\mathbf{X}_{i-1} = \alpha_i(\mathbf{X}_{i-2} - \mathbf{X}\alpha_{i-1}\alpha_{i-1}^T) \tag{2.3}$$

with maximum variance so that α_i is orthogonal to $\alpha_1, \alpha_2, \ldots, \alpha_{i-1}$ [7]. Singular value decomposition can be used to find PCs.

2.2 Feature Selection

There are generally three kinds of FS methods, *filters*, *wrappers* and *embedded methods* [3]. Filters use weighing functions to evaluate feature relevancies and select the top ranked features [4]. They are computationally fast, but do not consider the learning algorithm used in classification [4]. Wrappers on the other hand evaluate candidate subsets of features based on their performance with the target learning algorithm, choosing the best subset for each learning algorithm, but with hundreds or even thousands of features they require far too much time to evaluate all possible subsets [4]. Some methods combine filters and wrappers to get the best of both worlds: reasonable computational time with higher accuracy when combined with a specific classifier [4]. Embedded methods perform feature selection as part of constructing the classifier [3].

Another way to divide FS methods is into *supervised*, *unsupervised* and *semi-supervised* methods. Supervised methods use supervision information (usually class labels) of each observation in addition to the measured data. Unsupervised methods do not use any supervision information. This information is usually very useful for FS and in cases where we have it, supervised methods commonly outperform unsupervised ones [8]. However, sometimes getting supervision information is difficult or expensive, in which case we have to resort to unsupervised or semi-supervised methods (using supervision information for the portion of observations for which we have it). We will focus on supervised FS methods, since the information was available for us.

Since using some kind of a filter is practically necessary to keep computational time reasonable in applications that deal with high-dimensional data, we will first test a few filters and then see if classification accuracy can be improved by combining them with wrappers in Section 2.2.6. Filters generally work by calculating a score for each feature that reflects its importance in determining the class of a sample. Features are then selected starting with the highest (or lowest) score. There are many ways to calculate such a score.

In addition to how well the mathematical formulas used in each method reflect aspects that are useful in whatever classification we are doing, the features of different filters (not to be confused with the features they are scoring) determine how they perform in different classification tasks with different types of data. These features also affect how the filters can be improved or combined with certain wrappers to produce better results. Examples of these kinds of features are what kind of supervision information is used if any, whether or not redundancies of features are taken into account, or features being ranked individually versus incrementally adding features to an ordered set based on which features have already been selected.

2.2.1 The χ^2 test

The χ^2 test can be used to measure how well the frequency distribution of a random sample of n_{obs} observations that has been classified into c mutually exclusive classes fits the expected distribution. It offers a way to perform FS using a value that quantifies how dependent a predictor variable (feature) is of the response (class). We can evaluate a score for each feature and select a certain amount of the ones with highest scores. Suppose that we have a null hypothesis which gives probabilities p_i $(i = 1, 2, \ldots, c)$ that an observation falls into the ith class. Then the expected number of observations belonging to each class are $m_i = np_i$, where

$$\sum_{i=1}^{c} p_i = 1, \qquad \sum_{i=1}^{c} m_i = n_{\text{obs}}. \tag{2.4}$$

Now suppose that as a result of a classification, there are observed numbers x_i $(i = 1, 2, \ldots, c)$ of members of each class. Pearson [9] proposed the quantity

$$X^2 = \sum_{i=1}^{c} \frac{(x_i - m_i)^2}{m_i} = \sum_{i=1}^{c} \frac{x_i^2}{m_i} - n_{\text{obs}} \tag{2.5}$$

as a test criterion for the null hypothesis being correct. When the null hypothesis holds, the limiting distribution of X^2 as $n_{\text{obs}} \to \infty$ and p_i remain fixed is the χ^2 distribution

$$P_\nu(\chi^2)d\chi^2 = \frac{1}{2^{\nu/2}(\frac{\nu}{2} - 1)!}(\chi^2)^{(\nu/2)-1} e^{-\frac{1}{2}\chi^2} d\chi^2, \tag{2.6}$$

where ν is the number of degrees of freedom in χ^2 [9]. In Pearson's χ^2 test for goodness of fit $\nu = c - a$, where a is the number of fitted parameters in the distribution e.g. when performing a χ^2 test for a single feature to determine its importance, $a = 1$. We can then compare X^2 to the χ^2 distribution with ν degrees of freedom to find out the level of confidence (p-value) it would allow us to have in our null hypothesis. This can be thought of as an importance score for a feature in terms of FS since the similarity in the theoretical probabilities and observed frequencies of the mutually exclusive classes is assumed to be based on actual measurable differences in that particular feature between different classes.

A drawback of the χ^2 test when performed on each feature on their own is that it does not take into account possible redundancies between top features. The reason many top features get high scores in the test is often because they correlate highly with some other top feature, especially in high-dimensional feature spaces, making them mostly redundant in the presence of the other feature, sometimes even resulting in lower classification accuracy. Theoretically the test could be performed on sets of more than one feature but it would result in the same problems of computational complexity that exist in wrappers. Nonetheless, the χ^2 test is a simple and fast way to perform FS that works in some cases.

2.2.2 Minimum redundancy maximum relevance algorithm

The minimum redundancy maximum relevance (mRMR) algorithm was developed to tackle the issue of redundancy in feature sets obtained by simply selecting a certain number of top features ranked by some filter method. The algorithm adds new features to the feature set by minimizing their redundancy to the already selected features while maximizing their relevance to the response [10]. For discrete variables, both redundancy and relevance are measured using the mutual information I of two random variables x and y, which is defined by

$$I(x,y) = \sum_{i,j} p(x_i, y_j) \log \frac{p(x_i, y_j)}{p(x_i)p(y_j)}, \tag{2.7}$$

where $p(x_i, y_j)$ is the joint probability of x_i and y_j, i.e. the probability of both x_i and y_j occurring at the same time (x_i and y_j are certain discrete values of x and y) and $p(x_i), p(y_j)$ are the marginal probabilities of x_i and y_j. A high value of $I(x,y)$ means that x and y are mutually dependent and if they are both features, the other one is highly redundant in the presence of the other. If y is the response variable, it means that x is highly relevant to the classification problem.

Let S denote the subset of features we are seeking. The minimum redundancy condition is

$$\min W_I, \quad W_I = \frac{1}{\mid S \mid^2} \sum_{i,j \in S} I(i,j), \tag{2.8}$$

where $I(i,j)$ represents the mutual information of the ith and jth features and $\mid S \mid$ is the number of features in S. Let h denote the classification variable. The maximum relevance condition is

$$\max V_I, \quad V_I = \frac{1}{\mid S \mid} \sum_{i \in S} I(h,i). \tag{2.9}$$

The mRMR algorithm optimizes the conditions in Equations (2.8) and (2.9) simultaneously by combining them into a single criterion function, for which there are two simple alternatives:

$$\max(V_I - W_I), \tag{2.10}$$

$$\max(V_I / W_I). \tag{2.11}$$

Let Ω_S denote the set of features not yet selected by the algorithm ($\Omega_S = \Omega - S$, where Ω is the set of all features). The conditions in Equations (2.8) and (2.9) can be simplified for the algorithm into

$$\max_{i \in \Omega_S} I(h, i), \tag{2.12}$$

$$\min_{i \in \Omega_S} \frac{1}{\mid S \mid} \sum_{j \in S} I(i, j), \tag{2.13}$$

where (2.12) is equivalent to Eq. (2.9) and (2.13) is an approximation of Eq. (2.8), where $\frac{1}{|S|^2}$ is replaced by $\frac{1}{|S|}$. The first feature is selected according to (2.12) and the rest are incrementally added to S according to (2.10) or (2.11) (called Mutual Information Difference (MID) and Mutual Information Quotient (MIQ) respectively) or variants thereof [10]. For continuous variables, Ding and Peng presented different conditions for minimum redundancy and maximum relevance in [10], but in practical applications, continuous variables can often be discretized which allows us to use the conditions defined above.

2.2.3 ReliefF

The ReliefF algorithm is based on the idea that the quality of a feature can be estimated by how well it distinguishes between observations that are near each other in the feature space [11]. The algorithm randomly selects an observation R_i and searches for k of its nearest neighbors from the same class and k nearest neighbors from each of the other classes. These neighbors are called nearest hits H_j and nearest misses $M_j(C)$, where $j = 1, 2, \ldots, k$ and C denotes a class. Features that give different class values for neighbors of the same class are penalized while those that give different class values to neighbors of different classes are rewarded.

The difference between values of a numerical feature f for two observations $\mathbf{z}_1$ and $\mathbf{z}_2$ is defined as

$$\text{diff}(f, \mathbf{z}_1, \mathbf{z}_2) = \frac{\mid \text{value}(f, \mathbf{z}_1) - \text{value}(f, \mathbf{z}_2) \mid}{\max f - \min f} \tag{2.14}$$

and it is also used to find the nearest neighbors by summing the differences over all features (also known as Manhattan distance) [11]. The quality estimate $W[f]$ is initialized as 0 for all features and updated according to

$$W[f] \leftarrow W[f] - \sum_{j=1}^{k} \frac{\text{diff}(f, R_i, H_j)}{m \cdot k} + \sum_{C \neq \text{class}(R_i)} \frac{P(C)}{1 - P(\text{class}(R_i))} \sum_{j=1}^{k} \frac{\text{diff}(f, R_i, M_j(C))}{(m \cdot k)}, \tag{2.15}$$

where m is a user-defined parameter for how many times the whole process should be repeated. The term $\sum_{C \neq \text{class}(R_i)} \frac{P(C)}{1-P(\text{class}(R_i))}$ weights each miss ($C \neq \text{class}(R_i)$) with the prior probability of its class ($P(C)$) occurring among all possible misses (divided by $1 - P(\text{class}(R_i))$) so that in each step the contributions of hits and misses are in the interval $[0, 1]$ [11].

2.2.4 Fisher score

The idea behind Fisher score is to minimize the distance between observations of the same class, while maximizing the distance between observations of different classes in the data space spanned by the selected features. Given n_s selected features, the Fisher score of the reduced data space $\mathbf{Z} \in \mathbb{R}^{n_s \times n_{\text{obs}}}$ is

$$F(\mathbf{Z}) = \text{tr}\{(\tilde{\mathbf{S}}_b)(\tilde{\mathbf{S}}_t + \gamma \mathbf{I})^{-1}\}, \tag{2.16}$$

where tr is the trace of a square matrix, defined as the sum of the elements on the main diagonal $\text{tr}\{\mathbf{A}\} = \sum_{i=1}^{n} a_{ii}$ and γ is a positive regularization parameter. $\tilde{\mathbf{S}}_b$ and $\tilde{\mathbf{S}}_t$ are called the between-class scatter matrix and total scatter matrix respectively, defined as

$$\tilde{\mathbf{S}}_b = \sum_{k=1}^{c} n_k (\tilde{\boldsymbol{\mu}}_k - \tilde{\boldsymbol{\mu}})(\tilde{\boldsymbol{\mu}}_k - \tilde{\boldsymbol{\mu}})^T \tag{2.17}$$

$$\tilde{\mathbf{S}}_t = \sum_{i=1}^{n_{\text{obs}}} (\mathbf{z}_i - \tilde{\boldsymbol{\mu}})(\mathbf{z}_i - \tilde{\boldsymbol{\mu}})^T, \tag{2.18}$$

where $\tilde{\boldsymbol{\mu}}_k$ and n_k are the mean vector and number of observations of the kth class respectively in the reduced data space, $\tilde{\boldsymbol{\mu}}$ is the overall mean vector of the reduced data, $\mathbf{z}_i$ is the ith observation and c and n_{obs} are the number of classes and observations respectively. A perturbation term $\gamma \mathbf{I}$ is added to $\tilde{\mathbf{S}}_t$ to make it positive semi-definite. In high-dimensional feature spaces choosing the best n_s features is a very challenging combinatorial optimization problem and to alleviate the difficulty, Fisher scores can be computed individually for each feature [12]. The Fisher score of the jth feature is

$$F(\mathbf{x}^j) = \frac{\sum_{k=1}^{c} n_k (\mu_k^j - \mu^j)^2}{\sum_{k=1}^{c} n_k (\sigma_k^j)^2}, \tag{2.19}$$

where μ_k^j and σ_k^j are the mean and standard deviation of the kth class corresponding to the jth feature, and μ^j is the mean of the whole data set corresponding to the jth feature.

After computing the Fisher score for each feature, we can select the top n_s features with the highest scores, but since the score for each feature is computed individually the resulting subset is suboptimal because the scores don't reflect the fact that two features with low individual scores might have a very high score when combined. Additionally, Fisher score does not take into account redundancy between features [12]. Despite this, it performs well in some classification tasks.

2.2.5 Constraint score

Unlike most supervised feature selection methods, constraint score does not use class labels directly as supervision information. Instead it uses pairwise constraints, which specify whether a pair of observations belong in the same class or not and can naturally be derived from labeled data. Constraint score can be used when no explicit class label information is available but we know whether or not two observations belong in the same class or in applications where it is more practical to consider pairwise constraints than to try to obtain class labels. Another advantage of pairwise constraints is that they can sometimes be obtained automatically.

Pairs of observations in matrix $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]$ form the sets of must-link constraints $M = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to the same class}\}$ and cannot-link constraints $C = \{(\mathbf{x}_i, \mathbf{x}_j) \mid \mathbf{x}_i \text{ and } \mathbf{x}_j \text{ belong to different classes}\}$. The score of the rth feature using constraints in C and M is evaluated by

$$C_r^1 = \frac{\sum_{(\mathbf{x_i},\mathbf{x_j}) \in M} (f_{ri} - f_{rj})^2}{\sum_{(\mathbf{x_i},\mathbf{x_j}) \in C} (f_{ri} - f_{rj})^2}, \text{ or} \tag{2.20}$$

$$C_r^2 = \sum_{(\mathbf{x_i},\mathbf{x_j}) \in M} (f_{ri} - f_{rj})^2 - \lambda \sum_{(\mathbf{x_i},\mathbf{x_j}) \in C} (f_{ri} - f_{rj})^2, \tag{2.21}$$

where f_{ri} denotes the value of the rth feature of the ith observation. Equation (2.21) has a regularization coefficient λ to balance the contributions of the two terms and in the typical case where the distance between observations from the same class is smaller than that in different classes it should be set to $\lambda < 1$ [13].

The intuition of constraint score is similar to that of Fisher score; if two observations are from the same class, they should be close to each other and if they are from different classes, they should be far away from each other. Thus, the 'best' features are the ones with the lowest scores. Like all other scores that are computed individually for each feature, Constraint score has the same drawbacks, i.e., not taking redundancies of features into account and possibly missing pairs of features that might have a high score when combined. Nevertheless, it has been found to perform well in comparison to other methods on some datasets, in particular achieving similar classification accuracies to Fisher score while using much less supervision information [13].

2.2.6 Combining filters with wrappers

The problem with many FS methods is that they do not take dependancies or interactions (whether positive or negative) between features into account. Even those that take redundancies of features into account (e.g. mRMR) often miss positive interactions where a pair of complementary features are highly relevant and useful for the classifier even though they might be useless on their own. To the best of my knowledge there are no widely known reliable ways to predict or quantify which features benefit from the inclusion of which other features and by how much (especially when there are so many possible combinations), but some researchers have developed ways to combine filters with wrappers to test different combinations of the features with high filter scores and re-rank features during FS based on already selected features [14, 15]. Usually these methods can improve classification accuracy somewhat at the expense of some computational time or reduce the number of features needed to achieve similar accuracies to just using filters.

Hybrid method for feature selection

The authors of [14] presented two ways to add features to the selected subset based on the order of their filter ranking depending on if classification accuracy improves by doing so. One of the methods in [14], here called hybrid method for feature selection (HFS), starts by ranking the features and setting a limit for classification accuracy by computing it using only the first ranked feature. Then it combines each of the top n_{HFS} features with every other feature so that all the possible subsets of two features where at least one of them is in the top n_{HFS} features are evaluated. Subsets that have classification accuracy below the limit are discarded and the rest are sorted in the order of their accuracy. This becomes the new ranking and the new limit is set as the accuracy of the first ranked subset. Then the n_{HFS} best subsets are combined with every other feature subset that was kept and the resulting subsets are evaluated. Subsets with accuracies below the limit are once again discarded, the rest of the subsets are sorted and the highest accuracy is set as the new limit. The process of combining, evaluating, discarding and sorting the new subsets and updating the limit repeats until accuracy does not improve anymore.

Consider an example of the process:
1. A filter is used to generate an initial ranking of the features, obtaining f_1, f_2, f_3, f_4, f_5.
2. The limit is set as the classification accuracy obtained by using only the first feature of the previous ranking (f_1), which is 31.2%.
3. Since $n_{\text{HFS}} = 2$, subsets are generated by combining f_1 and f_2 with each of the rest of the features and we end up with the following combinations and corresponding accuracies:
a) using feature f_1: (f_1, f_2 – 35.1%),(f_1, f_3 – 28.8%),(f_1, f_4 – 43.2%),(f_1, f_5 – 26.5%)
b) using feature f_2: (f_2, f_3 – 33.3%),(f_2, f_4 – 30.3%),(f_2, f_5 – 41.2%)
4. Ranking the subsets that had higher accuracy than the previous best subset leaves

the following: (f_1, f_4 – 43.2%),(f_2, f_5 – 41.2%),(f_1, f_2 – 35.1%),(f_2, f_3 – 33.3%) and the new limit is 43.2%.
5. New subsets are generated combining (f_1, f_4) and (f_2, f_5) with the rest of the subsets in the previous ranking and we end up with the following subsets and corresponding accuracies:
a) using subset (f_1, f_4): (f_1, f_4, f_2, f_5 – 52.3%),(f_1, f_4, f_2 – 48.5%),(f_1, f_4, f_2, f_3 – 42.5%)
b) using subset (f_2, f_5): (f_2, f_5, f_1 – 50.2%),(f_2, f_5, f_3 – 40.2%)
5. Ranking the subsets that had higher accuracy than the previous best subset leaves the following: (f_1, f_4, f_2, f_5 – 52.3%),(f_2, f_5, f_1 – 50.2%),(f_1, f_4, f_2 – 48.5%), the new limit is 52.3% and there is no more ways to combine these subsets to form new ones, so the selected subset is (f_1, f_4, f_2, f_5).

A higher value of n_{HFS} usually leads to higher accuracy but it also makes the method very time-consuming especially in high-dimensional feature spaces and if we have hundreds or thousands of features, even with a small value of n_{HFS}, the method would not be practical to use because of time constraints.

2.2.7 Other improvements

Another way to reduce the computational time of a wrapper or improve the accuracy is to re-rank the features based on already selected features during incremental feature selection. This way features that correlate with the already selected features will be at the end of the new ranking and features that are conditionally relevant to the class based on already selected features will be at the top of the ranking [15].

Using an ensemble of classifiers with different feature sets where the prediction is made by majority voting also improves accuracy and generalization [16]. Intuitively it makes sense that while pines for example have certain distinguishing features, not all pines are as easy to classify using the same set of features. The environment in which a tree grows affects some of its features, making those features more or less distinguishable than they would be in other conditions. Also, spruce and birch trees have different distinguishing features, so it would make sense to use different features for classifying each one versus all the other species.

All of these improvements (combining filters with wrappers, using HFS, re-ranking features and using an ensemble of classifiers with different feature sets) are essential to our method in either improving accuracy or reducing computational time.

2.3 Classification methods

Classification methods generally work by dividing a feature space into parts according to where observations belonging to different classes are located. The result is a model, where each element maps to a single class or multiple classes, and test points are then classified according to the partition depending on where they are located.

The quality of a classifier is usually measured as the accuracy by which it classifies observations, i.e., the portion of observations that are correctly classified (or *classification loss*, the portion that is misclassified), but it is not always the only relevant metric. In some cases depending on the application, we have to consider the individual accuracies of classifying observations to certain classes that are deemed important, or make sure that the accuracy is not too low for any one class. Another metric that can be used is the *positive predictive value* (PPV), which is the portion of predictions of a certain class that was correct. If classifying a certain species is deemed more important than some other species, observations of that species can be weighed so that their contribution to the classification loss is larger than for members of the other species.

Default class weights mean that every observation is regarded as equally important for the classifier (they all have an observation weight of $1/n_{\text{obs}}$, where n_{obs} is the total number of observations) and consequently every species' importance is proportionate to its representation, often making the classifier biased towards the dominant species depending on the degree of dominance. Balanced class weights mean that observations are weighed so that each class as a whole is equally important for the classifier regardless of its representation (observation weights for members of class C are $1/(n_C c)$, where n_C is the number of observations of class C and c is the number of classes, in which case the sum of weights of each class is $1/c$). In calculating the (weighed) accuracy, we add up the observation weights of every correctly classified observation. These weights can be used to guide a wrapper towards either maximizing total accuracy or the average producer accuracy (classification accuracy among true members of a particular class) across of all species.

Accuracy is usually estimated by partitioning the data set into a *training set*, which is used to train the model and a *test set* against which the model is tested. This can be done multiple times in what is called *cross-validation* to get a more reliable estimation that does not depend on the particular way the original data was divided. For example, 10-fold cross-validation means that the data is divided in 10 parts with $\frac{1}{10}$ of the observations each, and each of them is used once as a test set while the others are used as the training set. In this thesis all the accuracies are computed with 10-fold cross-validation and the reported accuracy is the average of the 10 test sets.

2.3.1 k-nearest neighbors

The k-NN algorithm classifies an unknown point as the class that has the most points among the k observations from the known data that are closest to the unknown point. Given a training set of vectors $(\mathbf{x_1}, \cdots, \mathbf{x_{n_{\text{obs}}}})$ where each vector is an observation, a corresponding vector of known classes $\mathbf{C} = [c_1, \cdots, c_{n_{\text{obs}}}]$ for each observation, and a distance metric d, the nearest neighbor to a new observation $\mathbf{x}$ is the $\mathbf{x_j}$ that satisfies

$$d(\mathbf{x_j}, \mathbf{x}) = \min d(\mathbf{x_i}, \mathbf{x}) \qquad i = 1, 2, \cdots, n_{\text{obs}} \tag{2.22}$$

and thus the new observation is classified as c_j, if we choose $k = 1$ [17]. Choosing a bigger k will make the classifier more robust to possible outliers that happen to be closest to the new point in a cluster otherwise dominated by samples from a different class. In that case, the k nearest neighbors determine the class of the new observation by majority voting. Different weighting schemes can be applied to give the votes of nearer points more weight, which might further improve the accuracy.

2.3.2 Support vector machine

A support vector machine (SVM) works by trying to find the optimal hyperplane that separates members of different classes in the feature space. The hyperplane divides the feature space into two parts, so naturally a single SVM performs binary classification. However, multiple SVMs can be combined to perform multiclass classification. Each SVM can vote between a pair of classes (one classifier for every possible pair), and new observations can be determined to belong to the class that gets the most votes out of these two-class SVMs. If, for example, there are 3 classes C_1, C_2 and C_3, we would have one classifier voting between each possible pair $(C_1, C_2), (C_1, C_3), (C_2, C_3)$ and if the votes are C_2, C_3, C_2 respectively, the final prediction would be C_2.

The optimal hyperplane is defined as the one with the biggest margin, i.e. the distance between the hyperplane and the closest point. This can be formulated as an optimization problem,

$$\max_{\beta_j, j=1,\ldots,n_f} M_{\text{SVM}} \tag{2.23}$$

$$\text{subject to} \quad \begin{cases} \sum_{j=1}^{n_f} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{n_f} x_{in_f}) \geq M_{\text{SVM}} \end{cases} \tag{2.24}$$

where $M_{\text{SVM}} \in \mathbb{R}$ is the margin, n_f is the number of variables, x_{ik} is the kth variable of the ith observation and $y_i \in \{-1, 1\}, i = 1, \ldots, n_{\text{obs}}$ [18]. In real world applications, classes are sometimes not separable and thus the optimization problem has no solution unless we introduce slack variables ϵ_i to allow a certain amount of error E:

$$\max_{\beta_{ij}, \epsilon_i, i=1,\ldots,n_{\text{obs}}; j=1,\ldots,n_f} M_{\text{SVM}} \tag{2.25}$$

$$\text{subject to} \quad \begin{cases} \sum_{j=1}^{n_f} \beta_j^2 = 1 \\ y_i(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{n_f} x_{in_f}) \geq M_{\text{SVM}}(1 - \epsilon_i) \\ \epsilon_i \geq 0; \ \sum_{i=1}^{n_{\text{obs}}} \epsilon_i \leq E \end{cases} \tag{2.26}$$

Large E values increase the penalty for misclassifications on the training data, which leads to tighter margins and thus fewer misclassified observations on the training data. This may result in overfitting to the training data and make the classifier generalize poorly to new observations [18].

3 MATERIALS AND METHODS

A quantitative structure model (QSM) of a tree is a model of the woody structure of the tree that describes quantitatively its basic topological (branching structure), geometric and volumetric properties. These include properties such as number of branches in total and in any branching order, the parent-child relations of the branches and lengths, volumes, and angles of individual branches and branch size distributions. A QSM consist of building blocks, which usually are some geometric primitives such as cylinders or cones, see Figure 3.1. The circular cylinder is often used for trees because it is the most robust choice and, in most cases, a very accurate choice for estimating diameters, lengths, directions, angles and volumes [19].

A QSM offers a compact representation of the tree from which we can compute a huge number of structural features. Features computed from QSMs have shown to be useful in tree species classification [1, 2]. The QSMs for this work were reconstructed with the TreeQSM [https://github.com/InverseTampere/TreeQSM, different versions were used for different datasets [20, 21]] method from terrestrial laser scanner data.

3.1 Computing structural tree features from QSMs

It is possible to compute thousands of different structural tree features from QSMs. For the purpose of this work a MATLAB script that computes about 13000 features from each QSM was written. The script is available in GitHub [DOI: 10.5281/zenodo.4244231, src/-compute_features.m]. We have basically three types of features: one type is simply some kind of "tree attribute", for example stem volume, mean branching angle, 1st-quartile value of branch diameter, mean area of 3rd-order branches, etc. Some of these are quite standard in forestry and easily interpretable but others are already quite imaginative and hard to interpret and certainly hard to measure manually.

The second type consists of possible ratios of the features of the first type, for example stem volume/mean branching angle. These include some standard measures such as stem volume/dbh (diameter at breast height) but mostly these are non-standard and hard to interpret, and to measure manually.

The third type is comparison of different distributions to different reference distributions. For example, let us take the volume distributed for different branching orders and we compare it to, e.g., a certain approximate Gaussian distribution or triangular distribution (defined over same branching orders) by computing the mean or the maximum difference

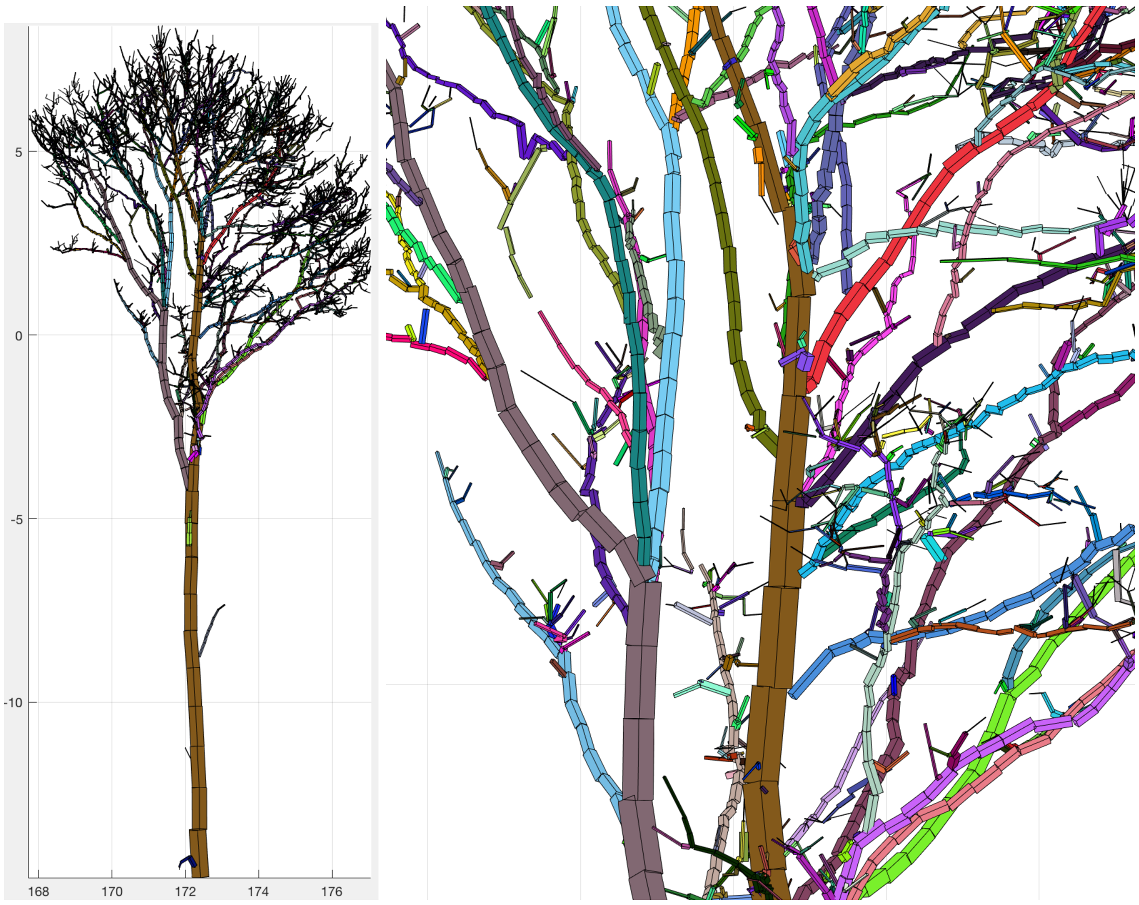


Figure 3.1. *A QSM of a Sycamore tree (left) and a close-up of the tree crown (right). It consists of circular cylinders and each color represents a branch.*

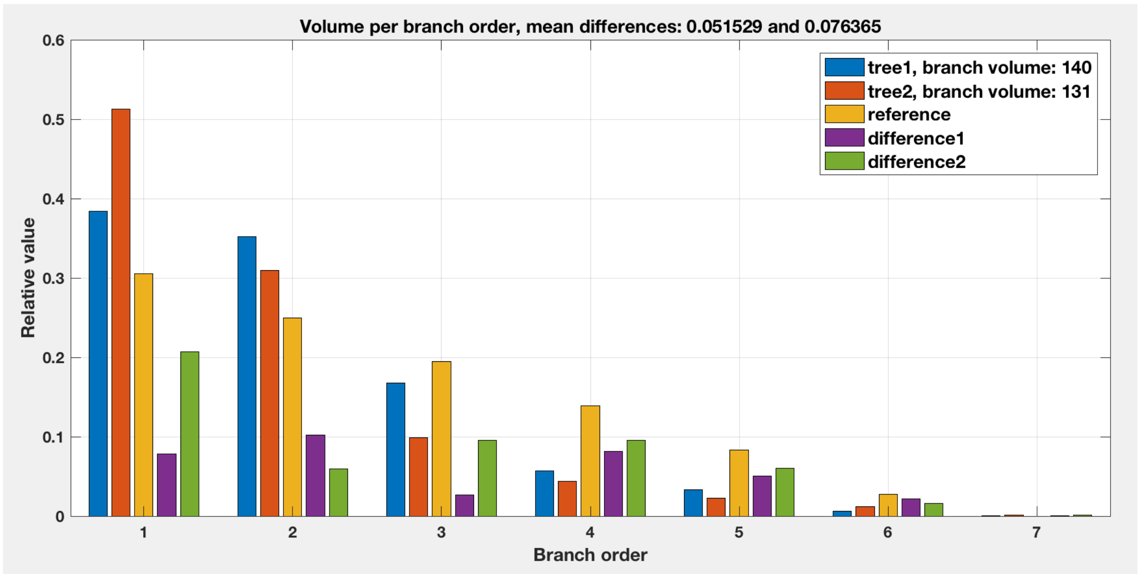


Figure 3.2. *An example of the third type of feature: the differences (difference1, difference2) of two observations (tree1, tree2) to a reference distribution (reference) are shown.*

in these distributions (the difference at each bin, then average of these or the maximum). The idea of these distribution comparisons is that we want to consider more information than just averages or medians: for example, two trees may have practically the same mean branching angle but the branching angle distributions may still be totally different and then when we compare these to different reference distributions they will show a difference, which was not in the average.

As another example, we can define certain vertical/height distributions and compare them to different reference distributions that cover the cases from bottom-heavy to uniform to top-heavy types and thus species that differ in “vertical profiles” should be distinguished with these types of distribution comparisons. An example of the third type of feature is shown in Figure 3.2.

The third type of features are named systematically and abbreviations are used to keep the names short. Volume (Vol), area (Are) and length (Len) of cylinders (Cyl) are distributed according to their diameter (Dia), height (Hei), azimuth (Azi) and zenith (Zen) angles. Triangle (triad), normal (normd) and uniform (unid) distributions are used as reference distributions. First, the distribution that is compared to a reference distribution is defined. For example, VolCylZen represents the distribution of the volumes of cylinders as a function of their zenith angle. Then the reference distribution is defined. For example, VolCylZen_triad_1_1_3 means that the previously defined distribution is compared to a triangle distribution where the starting point $x_s = 1$, middle point $x_m = 1$ and end point $x_e = 3$. Lastly, either mean or max specifies whether the value is the mean or maximum difference to the reference distribution. Different branch distributions are also compared. In addition to the distribution of volume, area and length, the distribution of the number of branches (Num) is also compared to reference distributions.

The relative heights, diameters and zenith angles of different percentages of bottom volume, area or length are also computed. For example, Rel_branch_Hei_bottom_30%_Vol means the relative branch height of the bottom 30% of branch volume. These are also computed for the 1st-order of branches, in which case there is branch1 in the name.

There are also a number of path-length based features. The path lengths from each branch’s tip and base to the tree’s base are calculated. From these, means, maximums, minimums, standard deviations and different ratios of those values are computed. Values for different percentiles of path lengths are calculated on their own and in relation to tip, base or tree height, for example 20%_base_path_length/base_height_ord1 means the 20th percentile of base path lengths divided by the base height of the first order of branches. Path lengths are also compared to reference distributions. Some of these features are also computed for first, second and third order (ord) branches separately.

Cylinder and branch volumes, areas and lengths as well as the number of branches between certain diameter and height classes and branch orders are also divided by other treedata. For example, AreCylDia_0.4_0.8/CrownRatio means the area of cylinders that have a relative diameter between 0.4 and 0.8 divided by the crown ratio, which is the height of the tree crown divided by the total tree height.

In previous research [1, 2], only a dozen or so features of the first and second type have been used and from those, every possible feature combination was tested. Our approach differs from the one used in [1, 2] in that it focuses on developing a method for FS out of necessity; while it is possible to test all possible feature combinations with 15 or 17 features to see which one performs the best, with thousands of features it would take a prohibitively long time.

3.2 Data

We had several datasets of QSMs of trees for which the species were known. The QSMs were obtained from UK, Finland and Cameroon; different kinds of environments, each with their own set of species. Each dataset has a different amount of species and trees and they are distributed differently, e.g. the data from UK has one clearly dominant species while the data from Cameroon does not.

3.2.1 Wytham woods, UK

The study area is a 1.4 ha plot in Wytham Woods (Oxford, UK) and it is dominated by five tree species: *Acer Pseudoplatanus* (ACERPS, Sycamore), *Fraxinus excelsior* (FRAXEX, European/Common ash) and *Crataegus monogyna* (CRATMO, Common hawthorn) constitute 88 % of the trees. Another 8 % is *Corylus avellana* (CORYAV, Common Hazel) and *Quercus robur* (QUERRO, Pedunculate/English oak). The dataset of QSMs consists of 755 identified and living trees of which 547, 81, 64, 37 and 26 trees are ACERPS, FRAXEX, CORYAV, QUERRO and CRATMO, respectively. More information about the plot, the species, the laser scanning and the QSMs can be found in [2]. The data was received from Dr. Kim Calder (Ghent University).

3.2.2 Punkaharju, Finland

This dataset consists of three single-species plots from Punkaharju, Finland. The plots consist of Silver birch (B, *Betula pendula Roth*), coniferous Scots pine (P, *Pinus sylvestris L.*) and Norway spruce (S, *Picea abies [L.] Karsten*) species. There are 358 birches, 457 pines and 276 spruces in our dataset of QSMs. More information about the plot, the species, the laser scanning and the QSMs can be found in [1]. The data was received from Prof. Raisa Mäkipää (Natural Resources Institute Finland).

3.2.3 Bouamir, Cameroon

This dataset consists of 368 tropical trees of 94 different species. The forest plot is from Bouamir, Cameroon. For our experiments we only included species that had at least 15 observations, which reduced the data to 120 trees of 5 species, *Greenwayodendron suaveolens* (Green), *Tabernaemontana crassa* (Taber), *Sorindeia grandifolia* (Sorin), *Uapaca guineensis* (Uapac) and *Strombosia pustulata* (Strom). The data was received from

Dr. Olivier Martin-Ducup (AMAP, University Montpellier).

3.3 Preprocessing

For each dataset, the preprocessing steps were the same (although some of them weren't necessary for some datasets), as follows. Based on the QSMs, 13567 features were computed for each tree. Any reciprocals of other features were removed. Trees that had only 0-values were removed and NaNs were located. Features and trees with too many NaNs ($> 10\%$ of all values) were removed. Constant features were removed and the features were scaled to the interval $[0, 1]$. Finally, features that correlated strongly ($|\rho| > 0.99$) with some other feature were removed until there were no more such correlations. The number of trees in each dataset before and after preprocessing is reported in Table 3.1.

For the Bouamir data, Inf and NaN-values were replaced by each species maximum and mean for the corresponding feature respectively to be able to perform PCA for comparison using the Matlab function pca. For the Punkaharju data, trees with a height of 5 meters or less and trees with 10 branches or less were removed.

Table 3.1. *Number of trees in each dataset before and after preprocessing.*

Dataset	before	only 0	$> 10\%$ NaN	after
Wytham	755	0	1	754
Punkaharju	1091	0	0	1091
Bouamir	120	1	0	119

3.4 The method: Decimated Input Ensembles for One Vs All Classification (DIE1VA)

Decimated input ensembles for one vs all classification (DIE1VA) is a filter-wrapper hybrid, implementing some parts and ideas of other hybrid and wrapper methods. The goal is to achieve higher classification accuracy than any of the presented filters could on their own with the expense of some computational time. The method chooses features for a user-defined number of 'One-vs-All' classifiers for all classes, so that each individual classifier is built to differentiate one species from the rest with different feature sets. The idea is that using different feature sets for different classes allows us to use better feature sets for classifying each individual class than using the same features for classifying all classes, thus achieving higher accuracy.

Having multiple classifiers for each class with different feature sets makes the method able to detect each class in more than one way, which could be useful in cases where certain conditions (e.g. growth environment or tree age) might affect which features are the most useful for the classification of a certain class. The final prediction of an observation will be correct even if only one of the correct species' One-vs-All classifiers classifies the observation correctly, as long as none of the other species' One-vs-All classifiers misclassifies it. Moreover, in high-dimensional feature spaces it is often easier to find many good feature sets that are complementary to each other, than to find a single feature set that is significantly better. More often than not, using a combination of multiple good feature sets results in more accurate classification than using one slightly more accurate feature set.

3.4.1 Overview

A flow chart of DIE1VA is presented in Figure 3.3. DIE1VA starts by picking a class and combining all the other classes into a new class called *other*. Features are then ranked based on this One-vs-All class divide using a filter. After ranking the features the HFS method is used to generate a user-defined number of feature sets from consecutive blocks of features in the ranking (the size of one block also being user-defined). The idea is to quickly generate enough decent feature sets that can be further improved upon. Since some blocks might have lots of redundant features — resulting in subpar feature sets being generated — we usually want the number of feature sets generated by HFS to be somewhat larger than the number of final subsets used for the classification. After HFS and any time features are added in later steps, the method checks if removing any features improves accuracy since the added features might be somewhat redundant with already selected ones and removing them might further improve accuracy with the added benefit of simplifying the classifier.

The best feature sets generated by HFS are combined with other HFS-generated feature sets to see if accuracy improves enough. The amount of improvement required for each new feature is set by the parameter `minimpr`. This is done to avoid ending up with feature sets that are too large in the sense that they cannot be much further improved because there is too much redundancy between already selected features and possible additions. Similar or higher accuracies can often be achieved with a fewer number of features if we do not add every feature that marginally improves the accuracy.

In the last phase of FS the method re-ranks the features based on already selected features before adding new features one by one in the order of the new ranking to see if accuracy improves. This step is repeated until no more significant improvement is made as parameters `minimpr` and `maxtime` make sure that we do not spend too much time on minor improvements that may not be worth it. After FS is done for every class, One-vs-All predictions are made using all of the final subsets for each class and these predictions are combined to make a final prediction based on majority voting.

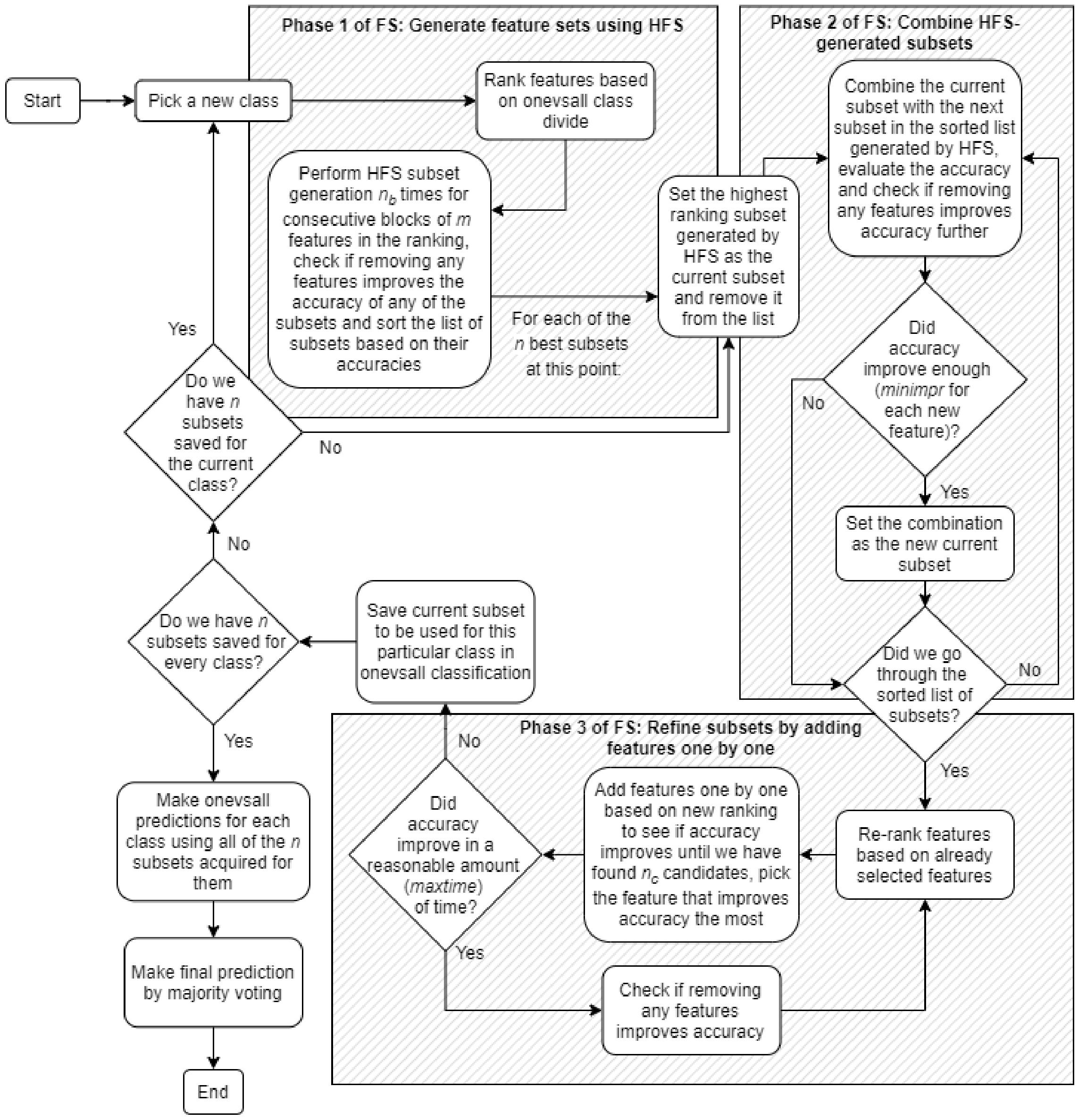

Phase 1 of FS: Generate feature sets using HFS
Phase 2 of FS: Combine HFS-generated subsets
Start
Pick a new class
Rank features based on onevsall class divide
Perform HFS subset generation n_b times for consecutive blocks of m features in the ranking, check if removing any features improves the accuracy of any of the subsets and sort the list of subsets based on their accuracies
For each of the n best subsets at this point:
Set the highest ranking subset generated by HFS as the current subset and remove it from the list
Combine the current subset with the next subset in the sorted list generated by HFS, evaluate the accuracy and check if removing any features improves accuracy further
Did accuracy improve enough (*minimpr* for each new feature)?
Yes
No
Set the combination as the new current subset
Did we go through the sorted list of subsets?
Yes
No
Do we have n subsets saved for the current class?
Yes
No
Do we have n subsets saved for every class?
No
Yes
Save current subset to be used for this particular class in onevsall classification
Make onevsall predictions for each class using all of the n subsets acquired for them
Make final prediction by majority voting
End
Phase 3 of FS: Refine subsets by adding features one by one
Re-rank features based on already selected features
Add features one by one based on new ranking to see if accuracy improves until we have found n_c candidates, pick the feature that improves accuracy the most
Did accuracy improve in a reasonable amount (*maxtime*) of time?
No
Yes
Check if removing any features improves accuracy

Figure 3.3. *Flow chart of the DIE1VA method.*

3.4.2 Details of the method

In this section we will take a closer look at how the method is implemented, what are the parameters and how they affect each other and the performance of the method. The MATLAB code of DIE1VA is available in Appendix A and in GitHub [DOI: 10.5281/zenodo.4244231]. The codes of two functions used in DIE1VA are available in Appendices B and C. The idea of the method is to generate a predetermined number of good feature subsets to distinguish each individual class from the rest, so we go through each class using the same steps.

First we combine the other classes into one and rank the features using the filter of our choosing. If we choose to prioritize the average accuracy of all classes we apply weights to each observation so that each class' contribution to the calculated accuracy will be equal regardless of their share in the sample. Then we can start using HFS to generate feature subsets. After each execution of HFS we check if removing any features from the generated subset improves accuracy by removing each feature one by one, and evaluating the accuracy of the resulting model. HFS is performed for different sets of features based on the ranking and the parameters m_{HFS} and n_b. For example, if $m_{\text{HFS}} = 30$ and $n_b = 5$ we use features ranked 1-30, 31-60, 61-90, 91-120 and 121-150 to generate five subsets using HFS.

The performance (time consumption vs. accuracy) of HFS is significantly affected by the number of top feature sets (n_{HFS}) that are combined with the rest to form new feature sets, because the other features are added based on their performance with the first n_{HFS} features in the block's ranking. If n_{HFS} is small compared to m_{HFS} there is a greater chance that none of the n_{HFS} top features in the block are a good starting point for HFS. The parameter n_b should be large enough so that we get enough good subsets that can be further refined, but not so large that we waste time on generating more subsets than we need. The parameter m_{HFS} should be large enough so there are enough non-redundant features to combine in a block, but also not so large that it would take too much time to test every combination. With larger n_{HFS} values, the number of combinations to test grows more rapidly with m_{HFS}. The optimal values for these parameters depend on how much redundancy there is between features in a block and how much the order in which we combine the features coupled with our parameter choices ends up essentially wasting time by combining redundant features.

We can think of the mutually exclusive features for each subset generated at this point as different perspectives to use in classification. Often they correctly classify different observations so that when they are combined we can correctly classify a larger range of observations from the same species than with only one set of features. This does not guarantee that a single feature could not end up in more than one of the final subsets generated by DIE1VA (which is not a problem necessarily), but usually they do not because the features added later depend on the HFS-generated subsets.

After HFS the method starts to refine the subsets, starting with the most accurate until no more significant improvement can be made. First the method combines the selected subsets with all other HFS-generated subsets to see if accuracy improves enough after also removing any redundant features that lower the accuracy. Although this step does not usually result in improvements for most of the subsets, it is a fairly inexpensive step that can sometimes save time or improve accuracy significantly, if enough of the features in both subsets have positive interactions with each other.

After this, the method re-ranks the features based on the already selected features by using Fisher scores; each feature is combined with the already selected subset to form a reduced data space matrix $\mathbf{Z}$ and the score for that feature is determined by Equation (2.16). Using this new ranking, features are added one by one to the subset of already selected features and the new classification accuracies are evaluated. Each feature that improves accuracy more than the amount specified by `minimpr` is saved as a candidate until the method has found n_c candidates, at which point the feature that resulted in the lowest loss of accuracy is selected.

Re-ranking and adding the features one by one to the subset of already selected features is repeated until no more significant improvement ($>$`minimpr`) to the accuracy is made in a reasonable time ($<$`maxtime`). The parameter n_c should be large enough so that we do not settle for slight improvements when bigger improvements are possible, but not so large that we end up wasting time on trying to find a marginally better new feature. When the accuracy of the subset can no longer be improved by at least `minimpr` in less than `maxtime`, the final subset is saved and the method moves on to refine the next best HFS generated subset (if we have less than n subsets for the current class), start over with a new class (if we don't have subsets for every class yet) or make One-vs-All predictions. These predictions are made by each class' subsets classifying observations as members of that class or the class *other* by using the same One-vs-All class divide for each class as was used in the feature ranking. Final predictions are then made by majority voting; each observation is classified as the most predicted class (excluding *other*) for that observation in the One-vs-All predictions. If an observation has the same number of votes for two or more of the most predicted classes, the final prediction is picked at random from them.

The method has many parameters, some of which are only used in some parts of the method (e.g. n_{HFS} in HFS) and a few that are used for DIE1VA as a whole (e.g. the number n of One-vs-All classifiers for each species). Some of the parameters greatly affect the computational time and for some, the optimal value (in terms of the trade-off between computational time and classification accuracy for example) depends on the data and other parameters. For each parameter, tuning them is at least partly a matter of finding the optimal trade-off between accuracy and time, where larger values usually lead to a higher accuracy, but requiring a longer time.

In addition to parameters we naturally have to choose the filter used to rank the features and the classification method used for the prediction. In the experiments in this thesis I used the methods presented in Chapter 2, first testing all of the presented filters using 1NN and then using Fisher score with 1NN and SVM classifiers on different datasets. Fisher score was used for re-ranking as it was able to score groups of more than one feature. Some other filters could also be defined for ranking groups of features. The method has an option to use balanced class weights instead of default class weights in calculating classification accuracy, essentially maximizing the average producer accuracy instead of the total accuracy. Matlab methods fitcknn and fitcsvm, which are part of the Statistics and Machine Learning toolbox, were used for classification. Table 3.2 contains descriptions of all the parameters and suggested values for our data.

Table 3.2. *Parameters of DIE1VA with suggested values.*

Parameter	Description	Suggested value (explanation)
n	Number of feature subsets for each class's One-vs-All classifier	3-10 (depends on number of classes and computational time constraints)
n_{HFS}	Number of top feature sets that are combined with the rest to form new feature sets in HFS	3-10 (classification accuracy vs. computational time constraints)
m_{HFS}	Size of the block in HFS	30-50 (depends on how redundant features tend to be grouped)
n_b	Number of subsequent blocks HFS is executed for	5-20 ($n_b \geq n$, depends on how many useful features there are)
`minimpr`	Minimum improvement in classification accuracy required per each new feature (after HFS)	0.005-0.05 (depends on how many features we want and what is possible)
n_c	Number of candidate features to look for before choosing	1-10 (classification accuracy vs. computational time constraints)
`maxtime` (seconds)	Time allowed for searching for one candidate feature	30-300 (classification accuracy vs. computational time constraints)

4 RESULTS

In this section the results of the experiments using DIE1VA on the three datasets described in Section 3.2 are presented. Depending on the goal, the performance of a feature set used in classification is evaluated during the execution using either default or balanced class weights in calculating the accuracy, and features are chosen according to that evaluation. The performance of DIE1VA is also evaluated using the same (default or balanced) class weights that are used during its execution. The accuracy that is calculated using balanced class weights is referred to as weighed classification accuracy (while using default class weights, the performance is evaluated using the regular classification accuracy). All evaluations are done using 10-fold cross-validation, i.e. the set of observations is randomly split into 10 parts, each of which is used once as a test set while the other parts are used as the training set. The reported accuracy is the average of the 10 test sets.

In Section 4.1 the effects of the most important parameters and the robustness of the method is studied on the Wytham data. Then the limits of classification accuracy DIE1VA can achieve when given more time are tested, as well as using the combination of SVM and 1NN classifiers. A few feature sets the method selects for classification of each species in the Wytham data are reported. The performance of DIE1VA is also tested on Punkaharju data in Section 4.2 so it can be compared to the results of [1]. Lastly in Section 4.3 the effect of using One-vs-All classifiers with different feature subsets for each species instead of one feature subset for all species is studied on Bouamir data. The FS method used in DIE1VA is also compared to PCA. The experiments were done in Matlab R2020a using an Intel Core i5-9600K (3.7 GHz) processor and 16 GB or RAM.

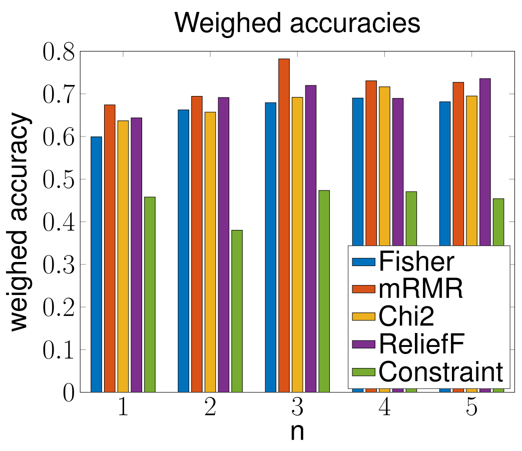
4.1 Wytham

4.1.1 The effect of the number of feature subsets used and the chosen feature filter

The effect of the number of feature subsets (n) used for each class's One-vs-All classifier and the chosen filter to weighed classification accuracy and time consumption on the Wytham dataset was studied by letting n range from 1 to 5 and setting the other parameters as constants ($n_{\text{HFS}} = 3, m_{\text{HFS}} = 30, n_b = 10,\ \texttt{minimpr} = 0.05, n_c = 1,\ \texttt{maxtime} = 60$) with each of the FS methods presented in Section 2.2. 1NN classification was used with balanced class weights. All other parameters used for the classification method were default parameters of the fitcknn function. Figure 4.1 shows the weighed accuracies and time consumptions of each FS method by the number of feature subsets used.

We can see that Constraint score has significantly lower accuracies with DIE1VA than all the other methods, that use class labels directly as supervision information. While mRMR is the most accurate its time consumption is further exacerbated since the ranking is done five times (once for every species). All the other FS methods have weighed accuracies within 5 percentage points of each other and time consumptions within 20 minutes of each other. The slight differences in weighed accuracies could be up to chance as they are within 2 standard deviations of each other according to the experiments on robustness with similar parameters. The weighed accuracy does not seem to consistently improve much after $n = 3$ with the parameters used here.

Since the features are grouped in the blocks in a way that does not take into account their interactions with other features in the same block (except to some extent when using mRMR), the redundancies and positive interactions within that block that affect the performance of any feature subset that can emerge from it are random. Additionally, since the subsets that do emerge are largely dictated by the first n_{HFS} features in any given block, if n_{HFS} is small compared to block size m_{HFS} there is more randomness in the resulting accuracy of those subsets.



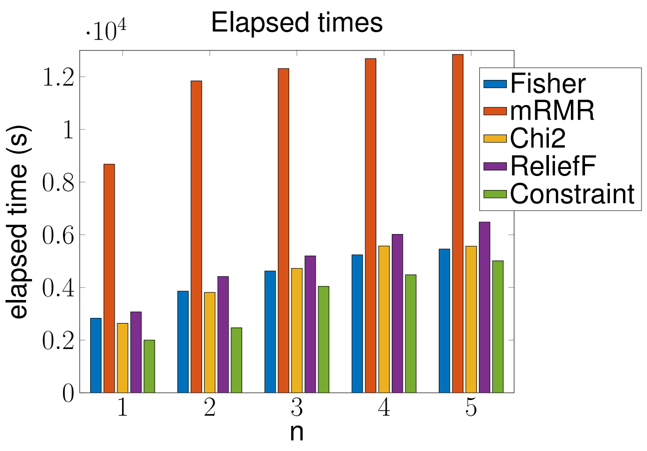


Figure 4.1. *Weighed accuracies (left) and time consumptions of DIE1VA (in seconds, right) with different FS methods and numbers of feature subsets n used for One-vs-All classification of each species.*

4.1.2 Robustness of DIE1VA

To test the robustness of DIE1VA it was executed 10 times with two different sets of parameters (Table 4.1) on the Wytham dataset using 1NN and SVM classifiers with both default and balanced class weights and Fisher score to rank the features. Average accuracies, weighed accuracies, elapsed times and their standard deviations are presented in Table 4.2 and a few confusion matrices from that test in Table 4.3.

Table 4.1. *Parameter sets used in the robustness experiments.*

Set	n	n_{HFS}	m_{HFS}	n_b	`minimpr`	n_c	`maxtime`
1	3	3	30	5	0.050	1	60
2	5	5	40	8	0.005	3	120

Table 4.2. *Average accuracies p, weighed accuracies p_w, elapsed times t and their standard deviations σ_p, σ_{pw} and σ_t over 10 executions of DIE1VA on the Wytham dataset.*

Set	Classifier	p (%)	σ_p (pp)	p_w (%)	σ_{pw} (pp)	t (s)	σ_t (s)
default class weights							
1	1NN	84.88	0.87	63.68	2.18	2359	87.0
1	SVM	78.90	**0.31**	40.65	**0.99**	**1290**	**50.4**
2	1NN	87.68	0.99	71.03	2.13	9020	855.6
2	SVM	79.48	0.40	39.79	1.63	4349	235.5
balanced class weights							
1	1NN	84.12	1.14	62.67	2.47	2303	126.1
1	SVM	70.52	1.03	62.98	1.34	2039	77.7
2	1NN	**87.79**	0.82	**72.11**	2.00	7909	350.3
2	SVM	80.89	0.49	71.18	1.44	7511	387.4

The standard deviations of classification accuracies are smaller across the board than in [2] while using the same classifiers and class weights. Although DIE1VA might choose somewhat different feature sets in different executions of the method, the results are more robust than by using the same set of features for all species repeatedly. Time consumption does not change much between different executions of the method with the same parameters; the standard deviation is always less than 10% of the mean time consumption and in most cases it is closer to 5%.

The parameters affect time consumption significantly. Using higher values for n, n_{HFS}, $m_{\mathsf{HFS}}, n_b, n_c$ and `maxtime` and a lower value for `minimpr` means that the method is more thorough in the search for the best subset, which leads to 2.80–3.67 percentage points higher accuracy and 7.35–9.44 percentage points higher weighed accuracy at the cost of more than triple the computational time, when using 1NN classifiers. Class weights do not seem to have much of an effect on either accuracy or weighed accuracy while using a 1NN classifier; accuracies and weighed accuracies for both parameter sets using default class weights are within one standard deviation from those using balanced class weights. For SVM classifiers on the other hand they do make a difference. With default class weights there is a heavy bias towards the dominant species which only becomes heavier with larger n values (which keeps the overall accuracy from increasing much, while hurting the weighed accuracy), but with balanced class weights the bias flips when n is high enough. The 1NN classifiers have higher accuracy overall, but they tend to be somewhat biased towards the dominant species, even while using balanced class weights. The number of ACERPS predictions is in many cases more than 10% higher than their true

Table 4.3. *Confusion matrices using parameters in Sets 1 and 2 with different class weights and classification methods. Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. Elapsed time (in seconds) is in brackets.*

Set 1, default class weights, 1NN classifiers (2312 s)						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	27	49	1	4	0	33.3
ACERPS	11	533	0	2	0	97.6
QUERRO	1	13	23	0	0	62.2
CORYAV	2	14	0	45	3	70.3
CRATMO	1	6	0	6	13	50.0
PPV (%)	64.3	86.7	95.8	78.9	81.3	85.0

Set 1, default class weights, SVM classifiers (1239 s)						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	5	69	3	1	3	6.2
ACERPS	1	536	4	3	2	98.2
QUERRO	0	29	8	0	0	21.6
CORYAV	2	14	3	44	1	68.8
CRATMO	3	7	3	10	3	11.5
PPV (%)	45.5	81.8	38.1	75.9	33.3	79.0

Set 2, default class weights, SVM classifiers (4384 s)						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	3	72	2	3	1	3.7
ACERPS	2	540	3	1	0	98.9
QUERRO	0	28	9	0	0	24.3
CORYAV	0	16	0	46	2	71.9
CRATMO	0	10	0	16	0	0
PPV (%)	60.0	81.1	64.3	69.7	0	79.3

Set 2, balanced class weights, SVM classifiers (7136 s)						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	48	23	3	7	0	59.3
ACERPS	45	468	17	9	7	85.7
QUERRO	0	6	31	0	0	83.8
CORYAV	7	0	0	57	0	89.1
CRATMO	1	2	0	15	8	30.8
PPV (%)	47.5	93.8	60.8	64.8	53.3	81.2

number. The species that get the most mixed up are FRAXEX and ACERPS, which often grow to the same size and have similar, round crowns. CORYAV and CRATMO are the second most similar pair, both resembling a bush more than a tree and consequently the most mixed up pair without ACERPS.

4.1.3 Combining SVM and 1NN

To see if weighed SVM and 1NN classifiers could be combined to balance out their biases towards the different species and to test the limits of time consumption vs. classification accuracy, DIE1VA was executed using both SVM and 1NN-classifiers with parameters $n = 10, n_{\text{HFS}} = 10, m_{\text{HFS}} = 50, n_b = 20,\ \texttt{minimpr} = 0.005, n_c = 3,\ \texttt{maxtime} = 180,$ balanced class weights and Fisher score to rank the features. This took about 24 hours. Then majority voting was performed using the One-vs-All classifiers with the lowest loss for each species so that the best 1, 3 and 5 1NN One-vs-All predictions for each species were combined with the best 1, 3 and 5 SVM One-vs-All predictions for each species respectively (using a total of 2, 6 and 10 classifiers) and compared to the best 2, 6 and 10 1NN classifiers. The confusion matrices are shown in Tables 4.4 and 4.5. The confusion matrix from using all 10 1NN and SVM classifiers is shown in Table 4.6.

Table 4.4. *Confusion matrices using 2 classifiers for each species, combining 1NN and SVM (top) and using only 1NN (bottom). Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner.*

1 1NN & SVM classifier for each species, weighed accuracy 76.7%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	58	17	1	4	1	71.6
ACERPS	40	495	5	4	2	90.7
QUERRO	0	3	34	0	0	91.9
CORYAV	6	0	0	58	0	90.6
CRATMO	4	4	0	8	10	38.5
PPV (%)	53.7	95.4	85.0	78.4	76.9	86.9

2 1NN classifiers for each species, weighed accuracy 75.9%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	45	30	1	4	1	55.6
ACERPS	10	528	3	4	1	96.7
QUERRO	0	7	30	0	0	81.1
CORYAV	5	3	1	54	1	84.4
CRATMO	1	5	0	4	16	61.5
PPV (%)	73.8	92.1	85.7	81.8	84.2	89.3

The results show that the bias towards the dominant species ACERPS is stronger when

Table 4.5. *Confusion matrices using 6 (top half) and 10 (bottom half) classifiers for each species, combining 1NN and SVM (upper) and using only 1NN (lower). Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner.*

3 1NN & SVM classifiers for each species, weighed accuracy 84.3%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	54	20	2	5	0	66.7
ACERPS	13	524	2	6	1	96.0
QUERRO	0	1	36	0	0	97.3
CORYAV	4	1	0	59	0	92.2
CRATMO	2	2	0	4	18	69.2
PPV (%)	74.0	95.6	90.0	79.7	94.7	91.6

6 1NN classifiers for each species, weighed accuracy 80.5%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	42	37	0	2	0	51.9
ACERPS	3	541	1	1	0	99.1
QUERRO	0	3	34	0	0	91.9
CORYAV	1	5	0	58	0	90.6
CRATMO	0	4	0	4	18	69.2
PPV (%)	91.3	91.7	97.1	89.2	100.0	91.9

5 1NN & SVM classifiers for each species, weighed accuracy 84.4%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	52	24	2	3	0	64.2
ACERPS	14	519	6	6	1	95.1
QUERRO	0	1	36	0	0	97.3
CORYAV	5	0	0	59	0	92.2
CRATMO	1	2	0	4	19	73.1
PPV (%)	72.2	95.1	81.8	81.9	95.0	90.8

10 1NN classifiers for each species, weighed accuracy 78.8%

Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	34	44	1	1	1	42.0
ACERPS	1	541	2	2	0	99.1
QUERRO	0	4	33	0	0	89.2
CORYAV	0	6	0	58	0	90.6
CRATMO	1	4	0	2	19	73.1
PPV (%)	94.4	90.3	91.7	92.1	95.0	90.8

Table 4.6. *Confusion matrix using 10 1NN & SVM classifiers for each species. Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner.*

10 1NN & SVM classifiers for each species, weighed accuracy 85.6%						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	52	23	2	4	0	64.2
ACERPS	9	521	9	6	1	95.4
QUERRO	0	0	37	0	0	100.0
CORYAV	2	1	0	61	0	95.3
CRATMO	0	3	0	4	19	73.1
PPV (%)	82.5	95.1	77.1	81.3	95.0	91.5

using only 1NN classifiers and increasing the number of One-vs-All classifiers for each species exacerbates the bias to the point that between using 6 and 10 1NN classifiers for each species, the total classification accuracy as well as the weighed accuracy decreases. Using one 1NN and SVM classifier for each species the bias is towards the other species with 51 false negatives compared to 24 false positives, but as we increase the number of 1NN and SVM classifiers the difference in the number of false negatives and false positives mostly disappears. The total accuracy is highest using 6 1NN classifiers and second highest using 3 1NN and 3 SVM classifiers, but the weighed accuracy (i.e. the average of the producer accuracies, which we are trying to maximize by using balanced class weights) keeps increasing as we increase the number of SVM and 1NN classifiers, albeit only slightly after using 3 of them for all species. It is not clear how well these results generalise to other data. Other datasets used in the experiments did not have such a strong majority for a single species that it resulted in as heavy a bias in the classifier.

4.1.4 Selected features

To see which features DIE1VA selects for classification of each species in the Wytham dataset, it was executed using 1NN classification and Fisher score to rank the features, and parameters $n = 3, n_{\text{HFS}} = 10, m_{\text{HFS}} = 50, n_b = 20,$ $\texttt{minimpr} = 0.05, n_c = 3,$ $\texttt{maxtime} = 120$. The selected feature sets are presented in Tables 4.7 and 4.8 along with the number of features and classification accuracies for each set. The accuracies are reported in terms of the One-vs-All classification each feature set would perform before the majority voting. They do not necessarily reflect the accuracy of the final classifier for any of the species, as we can see from the resulting confusion matrix in Table 4.9.

Every feature is included only once even though it is theoretically possible to choose the same feature for multiple feature sets after HFS and there is little difference in the accuracies of the One-vs-All classifiers for each species despite them having completely different feature sets. This is most likely due to an abundance of good features and

Table 4.7. *Three feature sets generated by DIE1VA to distinguish FRAXEX and ACERPS from the rest of the species in the Wytham dataset. # is the number of selected features.* p *is the accuracy of the One-vs-All classifier.*

Features	#	p (%)
FRAXEX		
'MaxBranchOrder/CrownRatio', 'VolCylZen_triad_1_1_3_max', 'Rel_branch_Hei_bottom_30%_Vol', 'Rel_branch1_Hei_bottom_50%_Len', 'Rel_branch1_Hei_bottom_45%_Num', 'mean_base_path_length_ord1', '10%_base_path_length_ord3'	7	90.5
'AreCylDia_0.4_0.8/CrownRatio', 'LenCylDia_0.6_1/CrownRatio', 'VolCylZen_triad_1_1_2_max', 'AreCylZen_triad_0_0_1_max', 'Rel_branch1_Hei_bottom_20%_Are', 'Rel_branch1_Hei_bottom_65%_Len', 'Rel_branch1_Hei_bottom_70%_Len', 'Base_path_length_triad_1_2_4_mean', 'NumBranch1Zen_unid_1_4_max', 'branch_or2_angle_mean', 'branch_or2_height_mean/or0_order_mean'	11	92.3
'TrunkLength/CrownLength', 'AreCylHei_0.6_1/TotalVolume', 'AreCylDia_triad_0_0_4_max', 'LenCylZen_normd_1_2_max', '40%_tip_path_length_all', '10%_base_path_length/base_heigth_ord1'	6	90.5
ACERPS		
'VolBranchOrd_2_4/TotalVolume', 'NumBranchOrd_1_2/TreeHeight', 'NumBranchOrd_4_4/CrownVolumeConv', 'Rel_cyl_Zen_bottom_75%_Vol', 'VolCylZen_triad_0_1_1_mean', 'VolCylZen_triad_1_2_2_max', 'AreCylZen_triad_1_2_3_max', 'AreCylZen_normd_3_1_max', 'LenCylZen_triad_1_1_4_mean', 'NumBranchZen_unid_2_4_mean', 'branch_or0_zenith_median', '30%_tip_path_length/tree_height_all', '55%_tip_path_length/tree_height_ord2', '60%_tip_path_length/tree_height_ord3', 'LenCylZen_unid_1_5_max'	15	87.5
'NumBranchOrd_5_5/TotalVolume', 'VolCylDia_0.2_0.4/TotalArea', 'VolCylDia_0.2_0.6/TotalArea', 'NumBranchOrd_2_3/DBHcyl', 'Rel_cyl_Zen_bottom_95%_Vol', 'VolCylZen_triad_1_2_2_mean', 'AreCylZen_triad_2_2_4_max', '20%_tip_path_length/tree_height_all', '45%_tip_path_length/tree_height_ord2', '70%_base_path_length/tree_height_ord1'	10	86.7
'LenBranchOrd_2_3/TotalVolume', 'AreCylDia_0.2_1/TrunkLength', 'LenCylDia_0_0.4/DBHcyl', 'LenBranchOrd_4_4/CrownVolumeConv', 'VolCylDia_triad_2_3_4_mean', 'VolCylHei_normd_1_1_mean', 'AreCylZen_normd_1_1_max', '15%_base_path_length/tree_height_all', '35%_base_path_length/tree_height_ord2', 'min(base_path_length/base_height)_ord3'	10	86.7

the mutually exclusive feature sets generated by HFS having positive interactions with different features because of that exclusivity.

Table 4.8. *Three feature sets generated by DIE1VA to distinguish QUERRO, CORYAV, and CRATMO from the rest of the species in the Wytham dataset. # is the number of selected features. p is the accuracy of the One-vs-All classifier.*

Features	#	p (%)
QUERRO		
’StemBranchLength’, ’VolCylDia_0_0.8/DBHcyl’, ’LenCylDia_unid_1_2_mean’, ’VolBranch1Dia_triad_1_2_4_mean’, ’branch_or3_area_mean’	5	99.2
’VolBranchOrd_3_3/NumberBranches’, ’AreBranchOrd_1_4/NumberBranches’, ’AreCylDia_triad_0_3_4_mean’, ’AreCylDia_triad_1_2_3_mean’, ’AreCylDia_triad_1_4_4_max’, ’AreCylDia_unid_1_4_mean’	6	99.1
’DBH/TreeHeight’, ’VolCylDia_0.2_1/TreeHeight’, ’VolBranchOrd_1_3/TotalLength’, ’VolCylHei_0.2_0.6/TotalArea’, ’VolCylHei_0.2_1/TotalArea’	5	99.1
CORYAV		
’VolCylDia_0.2_0.6/TrunkVolume’, ’Rel_cyl_Zen_bottom_55%_Vol’, ’VolCylZen_triad_0_1_4_mean’, ’VolCylZen_normd_2_1_mean’, ’LenCylZen_triad_1_2_3_mean’, ’LenCylZen_normd_1_1_mean’, ’NumBranchZen_triad_1_2_3_mean’, ’AreCylHei_0.4_0.6/BranchLength’	8	98.0
’TotalArea/CrownAreaConv’, ’NumBranchOrd_2_2/TotalVolume’, ’NumBranchOrd_4_5/TrunkArea’, ’AreCylDia_0.2_0.8/CrownLength’, ’NumBranchOrd_3_3/CrownVolumeAlpha’, ’LenCylDia_unid_3_4_mean’, ’AreCylZen_triad_1_2_2_max’, ’VolBranchHei_triad_3_4_4_mean’, ’LenBranchHei_triad_0_1_3_mean’, ’NumBranchZen_unid_2_3_max’, ’20%_base_path_length/base_heigth_ord1’, ’40%_base_path_length/tree_height_ord2’	12	97.5
’VolBranchOrd_1_3/TrunkVolume’, ’NumBranchOrd_4_5/BranchVolume’, ’LenCylDia_0.4_0.6/CrownLength’, ’LenCylDia_0.2_0.4/CrownRatio’, ’Rel_cyl_Zen_bottom_60%_Vol’, ’Rel_cyl_Zen_bottom_65%_Vol’, ’LenCylZen_triad_0_1_1_mean’, ’Rel_branch_Zen_bottom_30%_Num’, ’NumBranchZen_triad_0_1_1_mean’, ’95%_base_path_length/tree_height_all’	10	97.5
CRATMO		
’BranchLength/TotalArea’, ’VolCylDia_0.2_0.4/TotalVolume’, ’NumBranchOrd_1_5/CrownAreaConv’, ’VolCylDia_triad_1_3_4_max’, ’AreCylDia_normd_3_2_mean’, ’AreCylHei_triad_3_4_4_max’, ’Rel_cyl_Zen_bottom_95%_Len’, ’Base_path_length_triad_1_2_4_mean’	8	98.8
’NumBranchOrd_2_5/CrownAreaAlpha’, ’NumBranch1Hei_triad_3_4_4_max’	2	98.8
’VolBranchOrd_1_3/TotalVolume’, ’LenCylDia_0_0.8/CrownAreaConv’, ’NumBranchOrd_3_5/CrownVolumeAlpha’	3	98.8

Table 4.9. *Confusion matrix using 3 1NN classifiers for each species with the feature sets in Tables 4.7 and 4.8. Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. Elapsed time is in brackets.*

3 1NN classifiers for each species (18635 s)						
Actual\predicted	FRAXEX	ACERPS	QUERRO	CORYAV	CRATMO	p (%)
FRAXEX	42	36	0	3	0	51.9
ACERPS	10	532	1	3	0	97.4
QUERRO	0	5	32	0	0	86.5
CORYAV	2	6	0	56	0	87.5
CRATMO	0	9	0	2	15	57.7
PPV (%)	77.8	90.5	97.0	87.5	100.0	89.8

For FRAXEX, DIE1VA selects many features that are the relative branch height of some percentage of bottom volume, length or area, most of them being for the first order of branches. This might be because the tips of the branches curve upwards, making the height distribution of branches different from the other tall trees with round crowns. There are also some path length based features and distribution comparisons which separate FRAXEX from the more bush-like species in the dataset. For ACERPS there are a lot of path length based features and distribution comparisons, with path lengths commonly being divided by tree height and distributions being functions of the zenith angle or diameter of a cylinder. This might be because ACERPS has the most consistent and thick crown, with longer path lengths than the other trees of similar height.

QUERRO is the most easily separable among the species in this dataset, with the highest accuracies overall among the non-dominant species in the experiments. 5 features is enough for 99% accuracy in the One-vs-All classifiers. The second and third feature sets of QUERRO contain many similar features within the sets. Some of them might very well be close to redundant in the presence of the others, but end up in the same feature set during HFS by slightly improving accuracy. The area and volume distributions as functions of diameter and height being selected could be reflecting the fact that QUERRO has strong, thick first-order branches and there is more of a range in the thickness of branches compared to FRAXEX and ACERPS.

For the bush-like species CORYAV and CRATMO, branch distributions and the numbers of branches are used more commonly than for the other species. It seems that branch distributions separate them from the other species while also containing the most useful information about the structure of the bushes. CRATMO requires only two features to achieve nearly 99% accuracy in the One-vs-All classifiers, but since there are only a few observations of it and the ACERPS classifiers are biased (misclassifying CRATMO as ACERPS, affecting the result of the majority vote), CRATMO's producer accuracy is low.

4.2 Punkaharju

We executed DIE1VA on the Punkaharju dataset to see if the results in [1] could be improved upon and at what cost. The method was executed 10 times with parameters $n_{\text{HFS}} = 3, m_{\text{HFS}} = 30,$ $\texttt{minimpr} = 0, n_c = 10,$ $\texttt{maxtime} = 60$, and n ranging from 1 to 3 while having $n_b = 3n$. Both 1NN and SVM classifiers were used with default class weights and Fisher score to rank the features. Mean accuracies, time consumptions and

Table 4.10. *Average accuracies p, elapsed times t and their standard deviations σ_p and σ_t over 10 executions of DIE1VA on the Punkaharju dataset, using 1NN (left) or SVM (right) classifiers.*

n	p (%)	σ_p (pp)	t (s)	σ_t (s)
1NN				
1	97.13	0.62	**1952**	**425.2**
2	98.37	**0.41**	3578	463.1
3	**98.92**	0.49	7619	826.5

n	p (%)	σ_p (pp)	t (s)	σ_t (s)
SVM				
1	95.35	**0.27**	**1364**	**294.2**
2	96.06	0.34	3482	402.3
3	**96.59**	0.61	5476	622.6

Table 4.11. *Confusion matrices using 1 (top), 2 (middle) and 3 (bottom) 1NN (left) or SVM (right) classifiers for Birch (B), Pine (P) and Spruce (S). Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. Elapsed time is in brackets.*

1 1NN classifier (2064 s)

Act.\pred.	B	P	S	p (%)
Birch	355	2	1	99.2
Pine	12	438	7	95.8
Spruce	3	1	272	98.6
PPV (%)	95.9	99.3	97.1	97.6

1 SVM classifier (2042 s)

Act.\pred.	B	P	S	p (%)
Birch	351	4	3	98.0
Pine	22	430	5	94.1
Spruce	5	10	261	94.6
PPV (%)	92.9	96.8	97.0	95.5

2 1NN classifiers (4164 s)

Act.\pred.	B	P	S	p (%)
Birch	357	0	1	99.7
Pine	6	448	3	98.0
Spruce	4	2	270	97.8
PPV (%)	97.3	99.6	98.5	98.5

2 SVM classifiers (3552 s)

Act.\pred.	B	P	S	p (%)
Birch	351	6	1	98.0
Pine	21	431	5	94.3
Spruce	5	8	263	95.3
PPV (%)	93.1	96.9	97.8	95.8

3 1NN classifiers (7776 s)

Act.\pred.	B	P	S	p (%)
Birch	358	0	0	100.0
Pine	2	455	0	99.6
Spruce	5	0	271	98.2
PPV (%)	98.1	100	100	99.4

3 SVM classifiers (5269 s)

Act.\pred.	B	P	S	p (%)
Birch	357	0	1	99.7
Pine	20	429	8	93.9
Spruce	5	4	267	96.7
PPV (%)	93.5	99.1	96.7	96.5

the corresponding standard deviations are reported in Table 4.10 and confusion matrices from these results in Table 4.11.

In about an hour DIE1VA achieved over 98% accuracy using 2 1NN classifiers for each species, which is one percentage point higher than the highest accuracy in [1]. More than 2 hours was needed to achieve about 99% accuracy with 3 1NN classifiers for each species. Accuracies improved for both kNN and SVM classifiers from those in [1] and kNN is superior while using DIE1VA too. The standard deviation of accuracy is lower than in [1] using the same classifiers. The standard deviation of time consumption is higher as a percentage of the mean than on Wytham data, here more than 10% in each case and more than 20% when using one classifier for each species. Although Pine has the largest population in this dataset, all of the results are slightly biased towards the other species.

4.3 Bouamir

To test the effect of using multiple feature subsets for each species instead of one for all as well as the feature selection method used in DIE1VA, a modified version of DIE1VA was executed, which ranks and selects features based on the original class divide instead of One-vs-All and only uses the subset with the lowest classification loss. Three sets of parameters were used and the original version of DIE1VA was executed with the same parameters for comparison. The modified version of DIE1VA was also compared to using PCA for feature extraction before performing 1NN classification using 22 principal components which explained 80% of the variance. Confusion matrices and parameter sets used are shown in Tables 4.12, 4.13, 4.14 and 4.15.

Table 4.12. *Confusion matrix from using PCA for feature extraction before performing 1NN classification for all species. Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. The amount of explained variance is in brackets.*

1NN classifier for all species based on 22 PCs (~80% explained variance)						
Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	5	2	2	4	2	33.3
Taber	3	8	7	6	6	26.7
Sorin	3	5	1	7	5	4.8
Uapac	3	7	6	9	4	31.0
Strom	7	5	3	2	7	29.2
PPV (%)	23.8	29.6	5.3	32.1	29.2	25.2

Even by using the modified DIE1VA, we get significantly higher accuracies for all species in 10 minutes than by using PCA to extract features. By using One-vs-All classifiers for each species we get another significant increase in accuracy. Nearly all of the producer accuracies as well as the total accuracies are significantly higher when using One-vs-All classifiers for each species instead of one classifier for all species with the same

Table 4.13. *Parameter sets used to test the effect of multiple feature subsets for each species on the Bouamir data.*

Set	n	n_{HFS}	m_{HFS}	n_b	`minimpr`	n_c	`maxtime`
1	1	3	30	2	0.010	3	120
2	3	5	40	5	0.010	3	120
3	5	10	50	10	0.005	5	180

Table 4.14. *Confusion matrices using 1 1NN classifier for all species, with features generated by a modified version of DIE1VA using parameters in Set 1 (top), 2 (middle) and 3 (bottom). Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. Elapsed time is in brackets.*

1 1NN classifier for all species, Set 1 (610 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	7	0	3	2	3	46.7
Taber	3	14	1	6	6	46.7
Sorin	0	4	11	4	2	52.4
Uapac	5	5	3	11	5	37.9
Strom	2	8	2	3	9	37.5
PPV (%)	41.2	45.2	55.0	42.3	36.0	43.7

1 1NN classifier for all species, Set 2 (1457 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	10	1	0	2	2	66.7
Taber	4	16	1	6	3	53.3
Sorin	1	2	5	6	7	23.8
Uapac	5	5	5	9	5	31.0
Strom	0	4	2	4	14	58.3
PPV (%)	50.0	57.1	38.5	33.3	45.2	45.4

1 1NN classifiers for all species, Set 3 (7504 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	6	0	3	4	2	40.0
Taber	2	19	3	4	2	63.3
Sorin	2	3	7	5	4	33.3
Uapac	4	3	3	18	1	62.1
Strom	1	1	3	3	16	66.7
PPV (%)	40.0	73.1	36.8	52.9	64.0	55.5

parameters. The One-vs-All classifiers also have a higher accuracy in a shorter time when using different parameters compared to the one classifier, which would indicate that

Table 4.15. *Confusion matrices using the original DIE1VA with parameters in Set 1 (top), 2 (middle) and 3 (bottom), and 1NN classifiers for each species. Producer accuracies (p) for each species are at the end of each row and positive predictive values (PPV) at the end of each column. Total accuracy is in the bottom right corner. Elapsed time is in brackets.*

1 1NN classifier for each species (1396 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	10	2	0	0	3	66.7
Taber	6	15	5	3	1	50.0
Sorin	3	3	11	2	2	52.4
Uapac	1	5	3	14	6	48.3
Strom	1	3	2	6	12	50.0
PPV (%)	47.6	53.6	52.4	56.0	50.0	52.1

3 1NN classifiers for each species (3662 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	11	0	1	3	0	73.3
Taber	3	26	1	0	0	86.7
Sorin	3	3	10	4	1	47.6
Uapac	5	1	2	20	1	69.0
Strom	5	4	2	2	11	45.8
PPV (%)	40.7	76.5	62.5	69.0	84.6	65.5

5 1NN classifiers for each species (15499 s)

Actual\predicted	Green	Taber	Sorin	Uapac	Strom	p (%)
Green	15	0	0	0	0	100
Taber	0	30	0	0	0	100
Sorin	2	3	14	1	1	66.7
Uapac	0	1	2	24	2	82.8
Strom	0	2	3	1	18	75.0
PPV (%)	88.2	83.3	73.7	92.3	85.7	84.9

at least by using the method of FS used in DIE1VA it is more efficient to select feature sets for each species individually than to look for one feature set to classify all species. In less than 25 minutes, DIE1VA achieved an accuracy of 52.1% using parameters in Set 1 while the modified version achieved an accuracy of 45.4% in about the same amount of time using parameters in Set 2. In just over an hour, DIE1VA achieved an accuracy of 65.5% using parameters in Set 2 while the modified version took more than 2 hours to achieve an accuracy of 55.5% using parameters in Set 3. In about 4 hours and 20 minutes DIE1VA achieved an accuracy of 84.9% using parameters in Set 3.

5 DISCUSSION

In this chapter we will discuss how DIE1VA could be used based on results from the experiments, compare the results to previous research and present ideas for improvement and further research of the method.

5.1 General notes

The results of the experiments indicate that by changing the parameters to make the search for the best feature sets more thorough we are able to achieve higher accuracy on all of the datasets at the cost of additional time consumption. The method is in part a wrapper and features are only added if the model that is built on the new feature set is evaluated to have higher accuracy than the previous one.

When a dataset is randomly partitioned for 10-fold cross-validation during the evaluation there is a range of accuracies the classifier can get because of that randomness. If that range is larger than `minimpr` it is possible for a feature to be added simply because of chance in the way a particular partitioning happened to result in a higher accuracy than we previously had, which has nothing to do with the additional feature actually being useful to the classification task. However, in practice the possible negative effect of this is small compared to the benefits of using a wrapper. Especially since many times the features that improve accuracy the most are not ranked even in the top 5% by any filters while the ones that are, are often redundant to many of the already selected features.

Building the models to evaluate the accuracies is time-consuming, which means that in practical applications an important thing to consider when using DIE1VA is the trade-off between time consumption and classification accuracy. Which of them is prioritized over the other, and by how much, depends on the application and affects the optimal choice of parameters in terms of the trade-off. However, for typical applications in tree species classification the time consumption, if it is not excessive such as over a week, is not an important factor and maximum classification accuracy can be pursued if desired.

Experience with the given data can guide parameter choices. For example, if we are trying to raise the classification accuracy for some data, from 96% to as close to 100% as possible, the value of `minimpr` should be relatively low (maybe even 0), because as we get closer to 100% the possible improvements get smaller. If we know how long it takes to go through a certain percentage of the top ranked features and that useful features would not be found after that point, we can set that time as `maxtime`.

Information of the distribution of accuracies from HFS-generated subsets could be used in deciding the ratio of n/n_b. It is important to note, that a higher value of n does not necessarily lead to higher classification accuracy. A higher value of n further exacerbates the bias of 1NN classifiers with balanced class weights on the Wytham data in Section 4.1.3, resulting in lower accuracy (between using 6 and 10 1NN classifiers for each species, the total classification accuracy as well as the weighed accuracy decreases while the number of ACERPS predictions increases). These biases could be neutralized by combining different classifiers or by adjusting class weights for example, and research could be done regarding what is the most effective way to do it and why.

Even though optimizing the parameters and running the method takes time, once FS is done the classification is fast and the selected feature subsets can be saved for later use. This way we could continually improve the accuracy of our classifier by finding more and more feature sets and keeping the best ones for each species. Feature sets found on different datasets (with the same species growing in different environmental conditions, or among a different set of other species) could also be combined. Research could be done regarding whether or not (or to what extent) this makes the classifier generalise to a larger range of conditions or to a larger set of possible other species.

5.2 Wytham

The results show some improvement over those presented in [2], especially when using a kNN classifier. With default class weights DIE1VA achieved on average 84.88% accuracy in about 40 minutes, almost 5 pp higher and with a significantly lower standard deviation than the corresponding result in [2] (time consumption was not reported). Given enough time, the improvements in total and producer accuracies DIE1VA can achieve over those in [2] are significant across the board. As the most extreme example of this, in 24 hours using 10 weighed 1NN and SVM classifiers, DIE1VA achieved a total accuracy of 91.5% with producer accuracies of at least 64% for each species (three of them having producer accuracies above 95%). Using only one feature subset and classifier for all species resulted in either 80% total accuracy or producer accuracies above 57% for each species in [2]. The producer accuracy of the dominant species ACERPS, while using default class weights, was somewhat lower in many cases compared to that in [2]. This is most likely because the classifier used in [2] (misclassifying 122 other trees as ACERPS while misclassifying ACERPS only 8 times) was even more heavily biased towards ACERPS than the ones in the experiments of this paper.

It seems that when we try to find a single feature set to maximize total accuracy in a dataset where one of the species has such a strong majority, that subset has a tendency to be biased towards classifying the dominant species over others (because it has the largest effect on total accuracy). Consequently, the subset can only be used to correctly classify members of other species if they have enough of a distance to members of the dominant species in the space spanned by that same subset. The subset might be completely useless in distinguishing one of the species from the dominant one though, and if the species are close to each other in the feature space, a majority of the k nearest neighbors is very likely to be of the dominant species due to it having significantly more representation. Using different features for different species could give the less dominant species a better chance to be found by the One-vs-All classifiers, while using more subsets for each species could lower the chance that enough of the dominant species' One-vs-All classifiers are biased to the point that it affects the final outcome of the majority vote.

Only one of the 118 selected features is among the 17 features used in [2] ('StemBranch-Length' for classifying QUERRO). This seems to indicate that many of the thousands of additional features that are left after preprocessing (7000 – 10000 on the datasets used in the experiments) can be as useful in classification as the 17 manually chosen features in [2], at least on this dataset and in combination with other features. Further research could be done on what is the relationship of the selected features (which are more difficult to understand) to the simpler original 17 features, e.g. by studying their correlations.

5.3 Other datasets

On the Punkaharju data there was much less room for improvement in terms of accuracy, but it seems that a few more observations of each species can be found as a result of using different features for different species and a few more by using multiple different feature sets in classifying each species.

The results on Bouamir data show that using multiple One-vs-All classifiers for each species with different features enables us to achieve higher classification accuracy in a shorter time than using only one feature set for all species. The modified version of DIE1VA also achieves higher accuracy than using PCA in combination with the chosen classification method, so at least some of the improvement in terms of accuracy is due to the algorithm used to build and refine the subset for the classifier to use.

5.4 Ideas for further research

There are ways to tweak DIE1VA to achieve a more specific objective than to just maximize overall classification accuracy. The voting method could be modified to introduce some desirable bias; for example, if we want to minimize the number of false positives for class A even at the expense of accuracy, we can change the voting so that any time at least one of the One-vs-All classifiers classifies an observation as a class other than A, the final prediction cannot be A. The voting could also be modified so that each vote is weighed according to the One-vs-All classifier's accuracy for example, which was evaluated during the execution of the method and could be a more reasonable tie-breaker than picking the class at random and also provide some insight into how confident we can be of the classifiers prediction.

Further research could be done to see if the performance can be improved by changing some of the parameter values during the execution. Intuitively it would make sense that we lower the values of `minimpr` and n_c as each One-vs-All classifier gets closer to 100% accuracy and good additional features are increasingly hard to find. The way in which this should be done is far from trivial and likely also depends on the kind of data we are dealing with. Observation weights could also be modified during the execution to make the method pay more attention to observations that have not been correctly classified yet, which could help the final classifier take a wider range of observations into account. Research could also be done on using One-vs-One classifiers that distinguish between a pair of species instead of One-vs-All. The time-based parameter `maxtime` could be replaced by an iteration-based `maxiter` that would not affect the performance of the method depending on the hardware it is executed on.

6 CONCLUSION

Since QSMs enable the computing of thousands of tree features, there is potential to improve tree classification from previous studies which had only a fraction of the feature data that is now available. The abundance of features forces us to focus on feature selection and use a different approach than in previous studies [1, 2], when it was possible to test and optimize classifiers using every possible feature combination. On the other hand, we could use more of the features by using them only in some part of the classification (e.g. for a single species) instead of the classification as a whole (i.e. for all species).

A new method was developed to select features from high-dimensional QSM feature spaces to be used in tree species classification. The method, called DIE1VA, is a filter-wrapper hybrid that selects multiple feature sets for One-vs-All classification of each of the species and combines the votes of the One-vs-All classifiers to make the final prediction. Using multiple feature sets for each species DIE1VA is able to utilize more data in the classification than only using one feature set for all species.

The method was tested on three datasets (QSMs acquired from Wytham (UK), Punkaharju (Finland) and Bouamir (Cameroon)) and experiments showed that, given enough computational time, it achieved higher classification accuracies than previous experiments on the Wytham and Punkaharju data, although the improvement in Punkaharju data was small in part due to the accuracy already being close to 100%. The increase in accuracy is in part due to having thousands of features to choose from instead of a dozen or so, but also due to the method utilizing more of the data by using more features.

On the Wytham dataset the method achieved an accuracy of 84.88% in less than an hour, which is almost 5 percentage points higher than in previous experiments. In 24 hours it reached an accuracy of 91.5%, while having at least 64% accuracy for each class compared to previous studies having either 80% total accuracy or 57% producer accuracies. On the Punkaharju dataset the method achieved up to 98.9% accuracy, 2 percentage points higher than in previous studies. On Bouamir data a modified version of DIE1VA, which only selected one feature set to be used to classify all species, achieved significantly higher accuracies than performing principal component analysis before using one kNN classifier for all species. Using the original version of DIE1VA to select multiple feature sets for each species instead of one for all species resulted in another significant improvement in accuracy.

The presented method has several parameters that greatly affect its performance and experiments were done to study some of their effect, but there is still a need for more research, whether it is optimizing the parameters for different kinds of data or improving the method itself. Even though the method often takes hours to run, once the feature sets for each class are selected they can be saved and the classification is fast. The method could be used as is for more accurate tree species classification than previously, but there are still ways to improve it or tune it for a specific task. Further improvements to the method could make it viable for even more species and applications.

REFERENCES

[1] Åkerblom, M., Raumonen, P., Mäkipää, R. and Kaasalainen, M. Automatic tree species recognition with quantitative structure models. *Remote Sensing of Environment* 191 (2017), 1–12.

[2] Terryn, L., Calders, K., Disney, M., Origo, N., Malhi, Y., Newnham, G., Raumonen, P., Åkerblom, M. and Verbeeck, H. Tree species classification using structural features derived from terrestrial laser scanning. *ISPRS journal of photogrammetry and remote sensing* 168 (2020), 170–181.

[3] Guyon, I. and Elisseeff, A. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research* 3 (2003), 1157–1182.

[4] Seyyedi, S. H. and Minaei-Bidgoli, B. Using learning automata to determine proper subset size in high-dimensional spaces. *Journal of Experimental & Theoretical Artificial Intelligence* 29 (2017), 415–432.

[5] Duda, R. O., Hart, P. E. and Stork, D. G. *Pattern classification*. Wiley, 2001.

[6] Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, 1990.

[7] Jolliffe, I. T. *Principal component analysis*. Springer, 2002.

[8] Love, B. Comparing supervised and unsupervised category learning. *Psychonomic bulletin & review* 9 (2002), 829–835.

[9] Pearson, K. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50 (1900), 157–175.

[10] Ding, C. and Peng, H. Minimum redundancy feature selection from microarray gene expression data. *Computational Systems Bioinformatics* 2 (2003), 523–528.

[11] Robnik-Sikonja, M. and Kononenko, I. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Machine Learning* 53 (2003), 23–69.

[12] Gu, Q., Li, Z. and Han, J. Generalized Fisher Score for Feature Selection. *Uncertainty in Artificial Intelligence* 27 (2011), 266–273.

[13] Zhang, D., Chen, S. and Zhou, Z. Constraint Score: A new filter method for feature selection with pairwise constraints. *Pattern Recognition* 41 (2008), 1440–1451.

[14] Ruiz, R., Riquelme, J. C., Aguilar-Ruiz, J. and García-Torres, M. Fast feature selection aimed at high-dimensional data via hybrid-sequential-ranked searches. *Expert Systems with Applications* 39 (2012), 11094–11102.

[15] Bermejo, P., de la Ossa, L., Gámez, J. A. and Puerta, J. M. Fast wrapper feature subset selection in high-dimensional datasets by means of filter re-ranking. *Knowledge-Based Systems* 25 (2012), 35–44.

[16] Turner, K. and Oza, N. C. Decimated input ensembles for improved generalization. *International Joint Conference on Neural Networks* 5 (1999), 3069–3074.

[17] Cover, T. and Hart, P. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory* 13 (1967), 21–27.

[18] Guenther, N. and Schonlau, M. Support Vector Machines. *The Stata Journal* 16 (2016), 917–937.

[19] Åkerblom, M., Raumonen, P., Kaasalainen, M. and Casella, E. Analysis of Geometric Primitives in Quantitative Structure Models of Tree Stems. *Remote sensing* 7 (2015), 4581–4603.

[20] Raumonen, P., Kaasalainen, M., Åkerblom, M., Kaasalainen, S., Kaartinen, H., Vastaranta, M., Holopainen, M., Disney, M. and Lewis, P. Fast Automatic Precision Tree Models from Terrestrial Laser Scanner Data. *Remote Sensing* 5 (2013), 491–520.

[21] Calders, K., Newnham, G., Burt, A., Murphy, S., Raumonen, P., Herold, M., Culvenor, D., Avitabile, V., Disney, M., Armston, J. and Kaasalainen, M. Non-destructive estimates of above-ground biomass using terrestrial laser scanning. *Methods in Ecology and Evolution* 6 (2015), 198–208.

A MATLAB CODE OF THE DIE1VA METHOD

```
function [pred,preds,acc,elapsedTime,finalsubsets,finallosses] = ...
    DIE1VA(X,Y,n,n_HFS,m,n_b,minimpr,n_c,maxtime,NameValueArgs)

arguments
    % Feature values
    X
    % Class labels
    Y
    % Parameters of DIE1VA
    n
    n_HFS
    m
    n_b
    minimpr
    n_c
    maxtime
    NameValueArgs.FSMethod string = 'fisher'
    NameValueArgs.CMethod string = 'knn'
    NameValueArgs.NumNeighbors uint32 = 1
    NameValueArgs.Distance string = 'euclidean'
    NameValueArgs.Weighed string = 'off'
end

tStart = tic;
unqspecies = unique(Y);
finalsubsets = zeros(length(unqspecies)*n,20);
finallosses = zeros(length(unqspecies)*n,1);
for i=1:length(unqspecies)

    % Combine other species into one class, 'other'
    otheri = 1:length(unqspecies);
    otheri(i) = [];
    other = categories(removecats(unqspecies(otheri)));
    onevsall = mergecats(Y,other,'other');

    % Apply balanced or default class weights
    if strcmp(NameValueArgs.Weighed,'on')
        W = applyWeights(onevsall);
    else
        W = ones(length(Y),1);
    end

    % Rank features using the chosen filter
    if strcmp(NameValueArgs.FSMethod,'fisher')
        scores = fisherScore(X,onevsall,[],0);
        scores = sortrows(scores,2,'descend');
        idx = scores(:,1)';
    elseif strcmp(NameValueArgs.FSMethod,'chi2')
        [idx,~] = fscchi2(X',onevsall);
    elseif strcmp(NameValueArgs.FSMethod,'mrmr')
        [idx,~] = fscmrmr(X',onevsall);
    elseif strcmp(NameValueArgs.FSMethod,'relief')
        [idx,~] = relieff(X',onevsall,1);
    elseif strcmp(NameValueArgs.FSMethod,'constraint')
        scores = constraintScore(X,onevsall,50,10,0.1);
        scores = sortrows(scores,2,'descend');
        idx = scores(:,1)';
    end

    % Generate feature subsets using HFS and save their losses
    subsets = zeros(n_b,20);
    losses = zeros(n_b,1);
    for j=1:n_b
        [subset,loss] = HybridFS(X,onevsall,idx',n_HFS,m,W, ...
            'CMethod',NameValueArgs.CMethod, ...
            'NumNeighbors',NameValueArgs.NumNeighbors, ...
            'Distance',NameValueArgs.Distance);

        % Check if accuracy can be improved by removing features
        while length(subset) > 1
            [subset,betterloss] = removeToImproveAccuracy(X,onevsall, ...
                subset,loss,'CMethod',NameValueArgs.CMethod, ...
                'Weighed',NameValueArgs.Weighed, ...
                'Distance',NameValueArgs.Distance, ...
                'NumNeighbors',NameValueArgs.NumNeighbors);

            % Move on if accuracy did not improve
            if loss == betterloss
                break;
            end

            % Save new loss if accuracy improved
            loss = betterloss;
        end
        subsets(j,1:length(subset)) = subset;
        losses(j) = loss;
        idx(1:m) = [];
    end

    % Sort HFS-generated subsets according to their losses
    subsets = [subsets, losses];
    subsets = sortrows(subsets,size(subsets,2));
    losses = subsets(:,end);
    subsets = subsets(:,1:end-1);

    % Refine HFS-generated subsets starting with the one with the lowest
    % loss
    for x=1:n
        subset = subsets(1,subsets(1,:) > 0);
        prevloss = losses(1);
        subsets(1,:) = [];

        % Combine HFS-generated subsets to see if accuracy improves enough
        for j=1:size(subsets,1)
            newsubset = union(subset,subsets(j,subsets(j,:) > 0));
            if newsubset(1) == 0
                newsubset(1) = [];
            end

            % Build classifier
            if strcmp(NameValueArgs.CMethod,'knn')
                mdl = fitcknn(X(newsubset,:)',onevsall,'KFold',10, ...
                    'Weights',W,'NumNeighbors',NameValueArgs.NumNeighbors,
    ...
                    'Distance',NameValueArgs.Distance);
            elseif strcmp(NameValueArgs.CMethod,'svm')
                mdl = fitcsvm(X(newsubset,:)',onevsall,'Weights',W, ...
                    'KFold',10);
            end

            % Evaluate loss
            loss = kfoldLoss(mdl);

            % Check if accuracy can be improved by removing features
            while length(newsubset) > 1
                [newsubset,betterloss] = removeToImproveAccuracy(X, ...
                    onevsall,newsubset,loss, ...
                    'CMethod',NameValueArgs.CMethod, ...
                    'Weighed',NameValueArgs.Weighed, ...
                    'Distance',NameValueArgs.Distance, ...
                    'NumNeighbors',NameValueArgs.NumNeighbors);

                % Move on if accuracy did not improve
                if loss == betterloss
                    break;
                end

                % Save new loss if accuracy improved
                loss = betterloss;
            end

            % If improvement is not at least minimpr for each new feature,
            % it is not considered enough
            if loss < prevloss - minimpr*(length(newsubset) - ...
                    length(subset))
                subset = newsubset;
                prevloss = loss;
            end
        end

        % Add or remove features one by one until there's little to no improvement
        while length(subset) > 1

            % remove features if it improves performance
            [subset,loss] = removeToImproveAccuracy(X,onevsall,subset, ...
                prevloss,'Weighed',NameValueArgs.Weighed, ...
                'CMethod',NameValueArgs.CMethod, ...
                'NumNeighbors',NameValueArgs.NumNeighbors, ...
                'Distance',NameValueArgs.Distance);

            % If accuracy did not improve by removing features, try adding them
            if loss == prevloss

                % Re-rank features based on their fisher scores with already
                % selected features
                fisherscores = fisherScore(X,onevsall,subset,0);
                fisherscores = sortrows(fisherscores,2,'descend');
                idx = fisherscores(:,1);

                % add a feature if it improves performance
                [subset,loss] = addToImproveAccuracy(X,onevsall, ...
                    subset,prevloss,idx',n_c,minimpr,maxtime, ...
                    'CMethod',NameValueArgs.CMethod, ...
                    'Weighed',NameValueArgs.Weighed, ...
                    'Distance',NameValueArgs.Distance, ...
                    'NumNeighbors',NameValueArgs.NumNeighbors);

                % Move on if accuracy did not improve
                if loss == prevloss
                    break;
                end
            end

            % Save new loss if accuracy improved
            prevloss = loss;
        end

        % Save the final subset and corresponding loss
        finalsubsets((i-1)*n+x,1:length(subset)) = subset;
        finallosses((i-1)*n+x) = loss;
    end
end

% Make One-vs-All predictions for every species using features obtained for
% each species
preds = categorical(zeros(length(unqspecies)*n,length(Y)));
for i=1:length(unqspecies)
    for x=1:n
```

```

        % Combine other species into one class, 'other'
        otheri = 1:length(unqspecies);
        otheri(i) = [];
        other = categories(removecats(unqspecies(otheri)));
        onevsall = mergecats(Y,other,'other');

        % Apply balanced or default class weights
        if strcmp(NameValueArgs.Weighed,'on')
            W = applyWeights(onevsall);
        else
            W = ones(length(Y),1);
        end

        % Build classifier
        if strcmp(NameValueArgs.CMethod,'knn')
            mdl = fitcknn(X(finalsubsets((i-1)*n+x, ...
                finalsubsets((i-1)*n+x,:) > 0),:)', ...
                onevsall,'Weights',W,'KFold',10, ...
                'Distance',NameValueArgs.Distance, ...
                'NumNeighbors',NameValueArgs.NumNeighbors);
        elseif strcmp(NameValueArgs.CMethod,'svm')
            mdl = fitcsvm(X(finalsubsets((i-1)*n+x, ...
                finalsubsets((i-1)*n+x,:) > 0),:)', ...
                onevsall,'Weights',W,'KFold',10);
        end

        % Make One-vs-All predictions
        preds((i-1)*n+x,:) = kfoldPredict(mdl);
    end
end

% Combine One-vs-All predictions to make final prediction based on their
% votes
pred = categorical(zeros(1,length(Y)));
for i=1:size(preds,2)

    % Get all votes that are not 'other' from one observation
    notother = preds(preds(:,i) ~= 'other',i);

    % If only one vote was not 'other', assing final prediction as that class
    if length(notother) == 1
        pred(i) = notother;

    % If all votes were 'other' assign final prediction randomly from the
    % included species
    elseif isempty(notother)
        pred(i) = unqspecies(randi(length(unqspecies)));

    % If there were more than one vote for specific species, assign final
    % prediction as the species with most votes
    else
```

```
        [~,~,C] = mode(notother);

        % If there is a tie, assign final prediction randomly from the
        % species with most votes
        pred(i) = C{1}(randi(length(C)));
    end
end

% If there are extra categories that were not in the predictions, remove them
pred = removecats(pred);

% Calculate accuracy of predictions
acc = nnz(pred == Y)/length(Y);

% Record time consumption
elapsedTime = toc(tStart);
end
```

B MATLAB CODE OF ADDTOIMPROVEACCURACY

```matlab
function [subset,bestloss] = addToImproveAccuracy(FV,species,subset,
    bestloss,featureOrder,ncand,minimpr,maxtime,NameValueArgs)
%ADDTOIMPROVEACCURACY Add a feature to a subset if it improves
%classification accuracy
% Add features to subset one by one according to featureOrder to see if
% classification accuracy improves.  Finds ncand number of candidate
% subsets that have at least minimpr lower loss than bestavgloss and
% returns the one with the lowest loss.  Specify maxtime (in seconds) to
% terminate execution after a certain amount of time has passed after the
% last improvement.
arguments
    % Feature values
    FV
    species
    subset
    bestloss
    % Ranking of features acquired from a filter
    featureOrder
    % Parameters of DIE1VA
    ncand
    minimpr
    maxtime
    NameValueArgs.CMethod string = 'knn'
    NameValueArgs.NumNeighbors uint32 = 1
    NameValueArgs.Distance string = 'euclidean'
    NameValueArgs.Weighed string = 'off'
end

% Temporary subset where we add new features to the already selected feature set
tempsubset = [subset,0];

% Matrix and vector for saving candidate subsets and their losses
candidatesubsets = zeros(ncand,size(tempsubset,2));
betterlosses = ones(ncand,1);

% The index where we save new candidate subsets and their losses
k = 1;

% Apply balanced or default class weights
if strcmp(NameValueArgs.Weighed,'on')
    W = applyWeights(species);
else
```

```
    W = ones(length(species),1);
end

% Start the clock and go through features in the order of their ranking
elapsedTime = 0;
for i=1:size(featureOrder,2)
    tic

    % If feature has not been selected yet, add it to tempsubset
    if ~ismember(featureOrder(i),tempsubset)
        tempsubset(end) = featureOrder(i);

    % If feature has already been selected, move on to the next feature
    else
        continue;
    end

    % Build classifier
    if strcmp(NameValueArgs.CMethod,'knn')
        mdl = fitcknn(FV(tempsubset,:)',species,'KFold',10, ...
            'NumNeighbors',NameValueArgs.NumNeighbors,'Weights',W, ...
            'Distance',NameValueArgs.Distance);
    elseif strcmp(NameValueArgs.CMethod,'svm')
        mdl = fitcsvm(FV(tempsubset,:)',species,'KFold',10,'Weights',W);
    end

    % Evaluate loss
    loss = kfoldLoss(mdl);

    % If improvement in accuracy is more than minimpr, save new candidate subset
    if loss < bestloss - minimpr

        % Reset the clock, since an improvement was found
        elapsedTime = 0;

        % Save new candidate subset and corresponding loss
        betterlosses(k) = loss;
        candidatesubsets(k,:) = tempsubset;

        % If there are ncand candidate subsets, select the one with the lowest
        % loss and stop searching
        if k == ncand
            [bestloss,besti] = min(betterlosses);
            subset = candidatesubsets(besti,:);
            break;
        end

        % Move on to next index
        k = k + 1;
    end

    % Keep track of elapsed time and stop searching if it exceeds maxtime
    elapsedTime = elapsedTime + toc;
    if maxtime < elapsedTime
        break;
    end
end

% If any candidate subsets were found, choose the one with the lowest loss
% (betterloss == 1 if no improvements were found)
[betterloss,besti] = min(betterlosses);
if betterloss ~= 1
    bestloss = betterloss;
    subset = candidatesubsets(besti,:);
end
end
```

C MATLAB CODE OF REMOVETOIMPROVEACCURACY

```
function [subset,loss] = removeToImproveAccuracy(FV,species,subset,loss,
    NameValueArgs)
%REMOVETOIMPROVEACCURACY Remove a feature if it improves accuracy.
% Removes the one feature that improves classification accuracy the most
% (if any).  Rows of FV correspond to features, columns
% to observations.
arguments
    % Feature values
    FV
    species
    subset
    loss
    NameValueArgs.CMethod string = 'knn'
    NameValueArgs.NumNeighbors uint32 = 1
    NameValueArgs.Distance string = 'euclidean'
    NameValueArgs.Weighed string = 'off'
end

len = length(subset);

% Create all possible feature combinations that can be made by removing one
% feature from the subset, initialize vector to save the corresponding losses
featuresubsets = nchoosek(subset,len-1);
iter = size(featuresubsets,1);
losses = zeros(iter,1);

% Loop through every possible combination
for k=1:iter

    inputspace = FV(featuresubsets(k,:),:);

    % Create partition and initialize the number of incorrect predictions
    cv = cvpartition(length(species),'KFold',10);
    wrong = 0;

    % Loop through test sets in evaluating the classifier
    for i=1:cv.NumTestSets

        % Get training and test sets
        trainSet = cv.training(i);
        testSetIdx = find(cv.test(i));

        % Apply balanced or default class weights on the training set
        if strcmp(NameValueArgs.Weighed,'on')
            W = applyWeights(species(trainSet));
        else
            W = ones(nnz(trainSet),1);
        end

        % Build classifier based on training set
        if strcmp(NameValueArgs.CMethod,'knn')
            mdl = fitcknn(inputspace(:,trainSet)',species(trainSet), ...
                'NumNeighbors',NameValueArgs.NumNeighbors,'Weights',W, ...
                'Distance',NameValueArgs.Distance);
        elseif strcmp(NameValueArgs.CMethod,'svm')
            mdl = fitcsvm(inputspace(:,trainSet)',species(trainSet),...
                'Weights',W);
        end

        % Predict the class of each observation in test set
        for j=1:cv.TestSize(i)
            label = predict(mdl,inputspace(:,testSetIdx(j))');

            % If prediction was incorrect, update the number of incorrect
            % predictions
            if label ~= species(testSetIdx(j))
                wrong = wrong + 1;
            end
        end
    end

    % Save the loss of this feature set
    losses(k) = wrong/length(species);
end

% Get the subset with the lowest loss
[newloss,besti] = min(losses);
newsubset = featuresubsets(besti,:);

% If new loss is less than the old one, update subset and loss
if newloss < loss
    subset = newsubset;
    loss = newloss;
end
end
```