KU LEUVEN

FACULTY OF SCIENCES

Random Support Vector Trees

Towards Reconciling Random Forests in Theory and Practice

Vincent Buekers

Promoter: Prof. dr. ir. Johan A.K. Suykens
ESAT-STADIUS

Daily Supervisor: Yingyi Chen
ESAT-STADIUS

Master thesis submitted in fulfillment of the requirements for the degree in Master of Science in Statistics

Academic year 2019-2020

Preface

In this thesis, I present a machine learning model that aims to appeal to theoretical researchers and practitioners alike. It has been written to fulfil the graduation requirements for a master's degree in statistics from KU Leuven. Undoubtedly, writing a thesis is an essential part of education at master level. In line with prior expectations, I have faced a number of difficult challenges along the way. However, one can only grasp the true difficulties during the actual experience. Yet, in the end, it has been one of the most educational events for me to date. I chose this specific topic since it involved the development of an actual model, which is particularly interesting towards future employment. Note that the research topic was issued by the Leuven Statistics Research Centre, in cooperation with ESAT-STADIUS.

I would like to thank my supervisor Prof. dr. ir. Johan A.K. Suykens for the expert knowledge and feedback that he has shared on the topics discussed in this thesis. Furthermore, I want to express my sincere appreciation to doctoral candidate Yingyi Chen for her constant guidance and support throughout the entire research process. Most notably, she has been very helpful towards establishing the mathematical foundations of the proposed model. As a token of my appreciation, I continue speaking in terms of "we" throughout the rest of this thesis. More generally, I would like to take the opportunity to thank the Faculty of Science and the Leuven Statistics Research Centre (LStat) for their excellent organization of the master thesis as well as the clear and concise information that was put forward along the way, even during the challenging times of the COVID-crisis. Furthermore, both my promoter and daily supervisor are part of ESAT-STADIUS, so I would like to thank them as well. The cooperation between the aforementioned departments has been a good demonstration of the interdisciplinary nature of the Master of Statistics program at KU Leuven. Last but not least, I want to express my appreciation for the excellent teaching of the courses that covered the topics discussed in this thesis. In that regard, both the KU Leuven and its lecturers have allowed me to develop the skills and understanding necessary to complete this thesis. In addition, access to online learning platforms such as DataCamp have been very helpful. If not for these resources, the programming aspect of this thesis would likely have been a heavy burden.

- *Vincent Buekers*

Acronyms

COVID COrona VIrus Disease.

CV Cross-Validation.

ESAT Department of Electrical Engineering, KU Leuven.

ETC Extra-Trees Classifier.

GB Gigabyte.

GHz Gigahertz.

KUL Katholieke Universiteit Leuven.

LStat Leuven Statistics Research Centre.

Pegasos Primal Estimated sub-GrAdient SOlver for SVM.

QP Quadratic Programming.

RAM Random-Access Memory.

RBF Radial Basis Function.

RFC Random Forest Classifier.

RHS Right Hand Side.

RSVT Random Support Vector Trees.

SGD Stochastic Gradient Descent.

STADIUS Stadius Centre for Dynamical Systems, Signal Processing and Data Analytics, KU Leuven.

SVM Support Vector Machine.

UCI University of California Irvine.

List of Figures

List of Tables

Contents

Random Support Vector Trees

Vincent Buekers

2019-2020

Random Support Vector Trees

Towards Reconciling Random Forests in Theory and Practice

Vincent Buekers

Abstract

Ever since the inception of random forests, they have received widespread interest due to their universal applicability to a wide range of machine learning problems. From a practical perspective, random forests have a long-standing reputation of delivering state-of-the-art performance while being relatively simple and accessible to researchers and practitioners alike. In contrast, random forests have not been fully understood from a theoretical point of view. Most notably, properties such as the theoretical consistency (which ensures that the accuracy of a classification algorithm converges to the Bayes risk) have still not been established for the original algorithm. As such, the aim of this thesis is to develop a consistent random forest variant that also performs well on real-world machine learning tasks, thereby reconciling theory and practice. On one hand, we completely randomize our tree construction such that the individual trees become less data-dependent, in turn allowing us to derive the consistency results. On the other hand, we replace the traditional leaf nodes with linear support vector machines (SVMs) as to increase the versatility of our model. Straightforwardly, we hence call our method Random Support Vector Trees (RSVT). As the results show, RSVT is able to attain competitive classification accuracy while also having a proved theoretical consistency. Training RSVT comes at a significant computational cost due to the embedded model architecture, yet it remains feasible for data sets of small to moderate size.

Chapter 1

Introduction

Ensemble methods are a popular class of learning algorithms which consist of a set of individual classifiers, referred to as base learners. By virtue of diversification, ensembles offer a way to construct a versatile model with relatively simple building blocks, e.g. decision trees. Bagging (Breiman, 1996) and boosting (Freund and Schapire, 1996) are two of the most popular techniques to build such ensembles. The bagging approach was later consolidated in the seminal paper on random forests (Breiman, 2001), which is now one the most famous machine learning algorithms to date. In practice, random forests have been widely used to solve a variety of machine learning tasks. Most notably, their popularity is due to state-of-the-art performance while also being relatively simple and computationally efficient. Although the method was initially proposed in regard to classification and regression problems, they have since been applied to a wider variety of learning tasks, including density estimation, manifold learning and semi-supervised learning (Criminisi et al., 2011). In the context of this thesis, we exclusively focus on random forests for classification.

Although random forests have demonstrated their appeal in practice, they still lack several essential theoretical properties. Most notably, the consistency of the original algorithm has not yet been established (Denil et al., 2013). For a sequence of classifiers (e.g. random forests) to be consistent, it must be proved that their error converges to the Bayes risk, which is the minimum achievable risk of any classifier for a particular problem (Devroye et al., 1996). The difficulty in deriving the consistency lies in the highly data-dependent nature of the individual trees as well as the bagging operations. In order to make decision trees less data-dependent, researchers have thus far relied on randomizing the tree construction (Wang et al., 2016). As it turns out, this approach has resulted in a number of notable contributions, i.e. consistent random forests variants (Biau et al., 2008; Biau, 2012). Unfortunately, these theoretical advances have substantially diminished the empirical performance and thus also their appeal in practical settings. For that reason, further attempts have been undertaken to close the gap between theory and practice (Denil et al., 2014).

In an attempt to contribute to the consistent random forest variants, this thesis proposes a novel classification forest. Our method deviates from the original random forest in several meaningful ways. In line with previous research endeavours, we randomize the tree construction as to facilitate the theoretical analysis. To that end, we adopt the Extra-Trees algorithm by Geurts et al. (2006). As the name suggests, these extremely randomized trees are highly random by design. In order to completely eliminate the data dependency of the individual

trees, we use the totally randomized variant of Extra-trees. At this level of randomization, the structure of the decision tree is entirely independent of the classification target (Louppe, 2014). At the ensemble level, Extra-trees utilize the entire training sample to train each of the trees. In this regard, Extra-trees also greatly simplify the otherwise complicated bootstrap randomization.

On the other hand, adopting an ensemble of totally randomized trees is not geared towards we replace the traditional leaf node model by linear support vector machines (Vapnik, 1995; Cortes and Vapnik, 1995) as to increase the versatility of the value assignment, i.e. classification. This particular leaf node model has been largely inspired by the promising results of Hang et al. (2019). Conventionally, leaf nodes simply assign a class label based on the empirical distribution of the data contained therein. Instead, we propose an embedded model structure in which support vector machines (SVM) are trained on the data contained within each of the leaf nodes. In comparison to regular leaf nodes, our proposed model structure is rather complex. We therefore strictly adopt linear SVM as to not overly complexify our leaf nodes models. Resorting to linear SVM has the additional benefit that we can rely on highly efficient SVM solvers, e.g. LibLinear (Fan et al., 2008). The reason being is that the proposed embedded model architecture imposes serious computational demands. From a geometrical point of view, the leaf nodes of a decision tree in fact results in a partition of the input space. Moreover, the partition consists of non-overlapping regions by design, resulting in disjoint subsets of the training sample for each of the leaf nodes (Criminisi et al., 2011). Consequently, the tree-structured partition allows to train local SVM in an independent manner. This, in turn, facilitates the use of parallelization across the leaf node training procedures, further reducing the computational demands.

With these design choices in mind, we believe our method is promising from both a theoretical and empirical perspective. Considering we adopt a highly randomized tree construction, combined with using SVM as leaf node models, we hence refer to our method as Random Support Vector Trees (RSVT). The objectives of this model proposal are threefold: (1) establish a theoretical consistency w.r.t our random trees, (2) demonstrate that the embedded model structure is able to attain competitive performance results, (3) keep the computational demands as low as possible. Reaching the aforementioned objectives

The rest of this thesis is structured in the following way. Chapter 2 briefly discusses the literature in function of our RSVT model. More specifically, this entails a short review of decision trees, decision forests and support vector machines. Chapter 3 is devoted to a detailed explanation of our proposed model architecture and the corresponding trade-offs. Chapter 4 provides the consistency results on our random trees (including proofs). Chapter 5 contains the experimental results in regard to the performance and computational efficiency of RSVT. The performance is evaluated on several benchmark data sets (Dua and Graff, 2017). To put our results into perspective, we compare our performance to the original random forest and regular Extra-Trees. On the other hand, the computational complexity is examined by training RSVT on artificially generated data of increasing sample size. Chapter 6 contains an extensive discussion on both the theoretical and experimental results from chapter 4 and 5 respectively. In chapter 7, we finally draw conclusions on the contributions of RSVT.

Chapter 2

Literature Review

2.1 Decision Trees

Consider a general classification problem in which the output space y consists of several classes $\{c_1, c_2, \ldots, c_J\}$. Based on the associated class labels c_j, such a problem inherently defines a partition of the input space X, such that $X = \cup_j X_{c_j}$. A classifier then aims to learn a decision function ϕ in order to produce $X = \cup_j X^{\phi}_{c_j}$ (Louppe, 2014). As such, learning a classifier can alternatively be viewed as learning a partition of X most closely resembling the best possible partition. Binary decision trees, such as obtained by CART (Breiman et al., 1984) or C4.5 (Quinlan, 1993), quite directly apply this principle as they repeatedly split X into two child subsets. To do so, a binary function s_t is applied at each node t, either sending data to the left or right child node. Following the framework of (Criminisi et al., 2011), we consider X_0 to be the entire training sample at the root node, yielding X_1^L and X_1^R for the left and right child nodes respectively. For node t, one then has $X_t = X_t^L \cup X_t^R$. By construction, these subsets are disjoint due to the binary nature of the split. Indeed, $X_t^L \cap X_t^R = \emptyset$. Ultimately, the terminal subsets indeed form a partition of X (Breiman et al., 1984). In that sense, decision trees are as much a partitioning algorithm as they are a classifier. This can easily be illustrated by considering a simple 2-dimensional space that has been partitioned by repeated binary splits along either of the dimensions (Fig. 2.1)

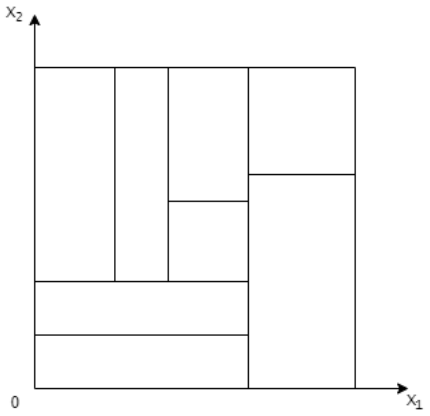


Figure 2.1: Decision tree partition of a 2-dimensional plane

In order to construct such binary decision trees, the binary split function must be determined. Firstly, this involves selecting a particular dimension, i.e. attribute, along which the binary

split should occur. Secondly, a cut-of point needs to be selected along the range of the selected attribute so that the binary nature of the split function can be enforced. As is common for learning algorithms, these parameters are to be optimized according to some training objective function. In the case of decision trees, this is often a deterministic measure such as the information gain or Gini index (Louppe, 2014). By doing so, the decision tree depends to a large extent on random nature of the data. It was eventually recognized that this caused decision trees to exhibit a high variance (Wehenkel, 1997). As such, the generalization capacity of decision trees was often unsatisfactory. This notion has motivated the conception of ensembles such as decision forests.

2.2 Decision Forests

2.2.1 Random Forests

In an attempt to reduce the aforementioned variance, Breiman (1996) proposed to construct an ensemble of randomized trees by means of bagging operations. In order to obtain randomly different trees, bagging simply considers a random bootstrap sample (Efron, 1979), i.e. a random sample with replacement, of the training set to construct each of the component trees. Consequently, a forest of randomly different trees is obtained. In his seminal work on random forests, Breiman (2001) took the randomization a step further by combining the bagging approach with random variable selection (Amit and Geman, 1997). Besides random forests, other notable randomization approaches include the random subspace method (Ho, 1998) and random split selection (Dietterich, 2000). While Breiman's approach randomizes the component trees to a large extent, there remains a data-dependency in determining the optimal cut-of point at each node. Indeed, the cut-of points have been shown to depend heavily on the random nature of the training data (Wehenkel, 1997; Geurts and Wehenkel, 2000). These findings have motivated Geurts et al. (2006) to develop their extremely randomized trees ensemble.

2.2.2 Extremely Randomized Trees

In addition to the random variable selection, extremely randomized trees, also known as Extra-Trees, also select the cut-of point uniformly at random. As such, the tree construction is almost entirely randomized. The only data-dependency remains in selecting the best attribute among the randomly selected ones (Geurts et al., 2006). However, Extra-Trees can be modified such that they become entirely independent of the classification target. Such totally randomized trees simply select a single attribute at random (Louppe, 2014). Unlike random forests, Extra-Trees do not rely on random bootstrap samples to diversify the learning sample for each of the component trees. Instead, the entire training sample is used for every single one of the individual trees (Geurts et al., 2006). Clearly, Extra-Trees are much simpler and, most importantly, less data-dependent than random forests. Evidently, this makes Extra-Trees particularly appealing from a theoretical point of view. On the other hand, the increased randomness produces a lower prediction confidence (Criminisi et al., 2011). Nonetheless, Extra-Trees very often demonstrate competitive performance, making them a rather popular decision forest in practice. An additional benefit is that they are also faster to compute since there is less optimization involved.

2.3 Support Vector Machines

Much like random forests, support vector machines (Vapnik, 1995) are very popular statistical learning algorithms in a wide variety of disciplines. Most notably, they are a well-established method for effectively dealing with classification problems. For separable data structures, the solution takes the form of an optimal separating hyperplane. More specifically, it is optimal in the sense that it is a unique solution that separates the training data with maximal margin (Suykens et al., 2002). The full line depicted in Fig. 2.2 represents such a hyperplane, whereas the margin is represented by the area between the dashed lines. Indeed, the orientation of the hyperplane maximizes the margin, i.e. the distance to the nearest points of both classes.

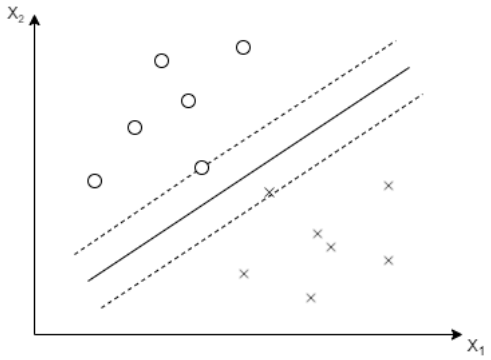


Figure 2.2: Maximum-margin hyperplane for a separable binary classification problem

In what follows, we give a brief overview of the mathematical formulation of linear support vector machines (SVM). In contrast to the maximum-margin version illustrated above, we discuss the more general soft-margin extension (Cortes and Vapnik, 1995) which can be applied to non-separable data.

2.3.1 Mathematical Formulation

Consider a training set $\{(x_i, y_i)\}_{i=1}^{n}$, with the observed attributes $x_i \in \mathbb{R}^d$ and the binary class labels $y_i \in \{\pm 1\}$. We now want to separate the observed data in function of these class labels. To do so, the aforementioned soft-margin classifier can be used, which is defined as (2.1) (Suykens et al., 2002).

$$y(x) = \text{sign}\left[w^T x + b\right] \tag{2.1}$$

where $w^T x + b = 0$ corresponds to the hyperplane, i.e. decision boundary. On the other hand, the margins correspond to $w^T x + b = -1$ and $w^T x + b = +1$. As such, the distance between the margins is $2/\|w\|_2$. Since a maximum margin hyperplane is desired, one must maximize this quantity by minimizing $\|w\|_2$ (Suykens et al., 2002). In order to now obtain the desired hyperplane, one must find the normal vector w. To do so, a constrained optimization problem has to be solved (2.2) (Burges, 1998).

$$\min_{w,b,\xi} \frac{1}{2}\|w\|_2 + C\left(\sum_{i=1}^{n} \xi_i\right)^k \tag{2.2}$$

subject to the constraints

$$y_i \left[w^T x_i + b \right] \geq 1 - \xi_i, \quad i = 1, \ldots, n$$
$$\xi_i \geq 0, \quad i = 1, \ldots, n$$

By adding the slack variables ξ_k, the soft-margin formulation allows for misclassification to occur. In other words, a certain point is allowed to be on the wrong side of the hyperplane. For an error to occur, this means that the corresponding ξ_k is greater than one. In order to regulate the extent to which misclassification is tolerated, the C parameter is introduced. More specifically, C controls the amount of overlap between the two class distributions of a binary classification problem (Hastie et al., 2004). Note furthermore that k specifies the loss function to be used. More specifically $k = 1$ corresponds to an L1-loss whereas $k = 2$ would instead correspond to an L2-loss (Koshiba and Abe, 2003). In order to solve the aforementioned optimization problem (2.2), one has to consider the corresponding Lagrangian, of which the multipliers give rise to the dual of this Quadratic Programming (QP) problem (2.3) (Burges, 1998).

$$\max_{\alpha_i} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.3}$$

subject to the constraints

$$0 \leq \alpha_i \leq C$$
$$\textstyle\sum_i \alpha_i y_i = 0$$

The solution to (2.3) is characterized by the non-zero α_i. The corresponding data points are referred to as the support vector which constitute the sparse solution (Suykens et al., 2002). Note lastly that the explicit dot product $x_i \cdot x_j$ is often replaced by a (non)-linear kernel $K(x_i, x_j)$. For linear SVM, the kernel is simply $K(x_i, x_j) = x_i \cdot x_j$. Instead, non-linear SVM are often obtained by, for instance, an RBF or a polynomial kernel (Menon, 2009). However, we restrict our discussion to that of linear SVM as RSVT does not incorporate non-linear SVM whatsoever.

2.3.2 Efficient Solvers

In his work on dependencies in empirical data, Vapnik (1982) outlined his QP solver for SVM, which solves the problem involving all non-zero Lagrangian multipliers. From a computational point of view, this method is demanding as it only converges in $\mathcal{O}(n^3)$, making it quite infeasible on larger amounts of data. In an attempt to speed up SVM training, many alternative solvers have been proposed. Currently, the availability of efficient SVM solvers is quite extensive; e.g. SMO (Platt, 1998), SVM$^{\text{light}}$ (Joachims, 1999), Pegasos (Shalev-Shwartz et al., 2007), SVM$^{\text{perf}}$ (Joachims, 2006), LibLinear (Fan et al., 2008) and SVM-SGD (Bottou, 2007). For linear SVM in particular, both LibLinear and SVM-SGD are very popular in practice. However, from a theoretical perspective, SVM-SGD is not yet as appealing as LibLinear. As it stands, the mathematical foundations of stochastic gradient descent (SGD) have so far mostly been studied in relation to neural networks (Bottou, 1991, 2010). In relation to SVM, SGD is limited to a software implementation (Bottou, 2007), which has thus far not been accompanied by a formal publication. As such, the convergence of the algorithm is still unknown at this point. In contrast, LibLinear is known to converge in $\mathcal{O}(nd \log(1/\rho))$, where d is the dimension and ρ the optimization tolerance (Menon, 2009).

Chapter 3

Model Architecture

3.1 Algorithm Description

The RSVT classifier is constructed with a rather straightforward rationale and operates as a batch-mode learning algorithm. Firstly, we construct a completely random decision tree following the totally randomized variant of the Extra-Trees algorithm. Recall that this involves selecting one random attribute at every internal node, as well as randomly selecting the cut-of point (Geurts et al., 2006). Once the tree is constructed, its leaf nodes and corresponding data can easily be extracted since the leaf nodes constitute independent subsets of the training sample. Next, we distribute the test sample to the obtained leaf nodes according to the binary decision rules that characterize the tree. Instead of the conventional leaf node predictions, we instead replace the value assignment mechanism with that of a (linear) support vector machine inside each of the leaf nodes of the tree, thus yielding a random support vector tree (Fig. 3.1).

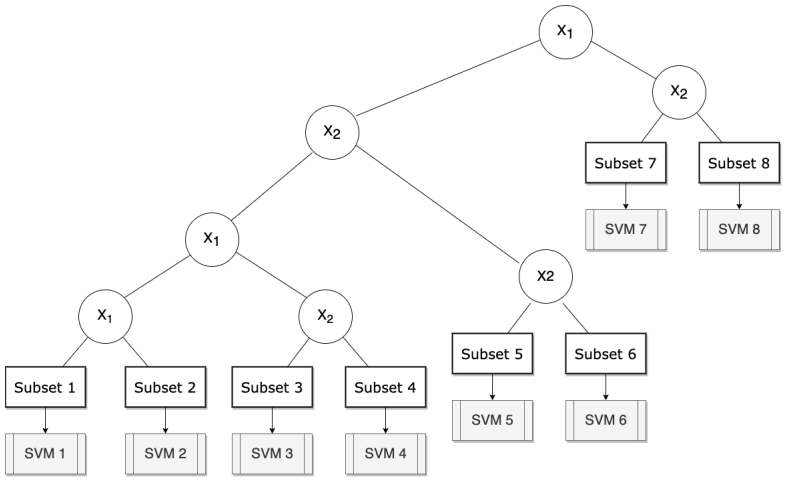


Figure 3.1: A Random Support Vector Tree

In other words, since the leaf nodes of totally random decision tree yield a random partition of the input space, we can proceed by utilizing the leaf node subsets as independent samples for training linear SVM. Considering them as independent samples is possible due to the non-overlapping nature of the partition, ensuring that each instance is exclusively assigned to a single region. Embedding linear SVM within the leaf nodes hence corresponds to constructing soft-margin hyperplanes within each of the regions of the input space. By doing so, we obtain

a collection of local SVM which are all confined to a rectangular subspace of the input space, as is depicted in Fig. 3.2. While each of the leaf node SVM are strictly linear, the overall model is piece-wise linear since the orientation of the soft-margin hyperplanes is different for each region. With this in mind, RSVT should be able to capture non-linear data structures to a certain extent.

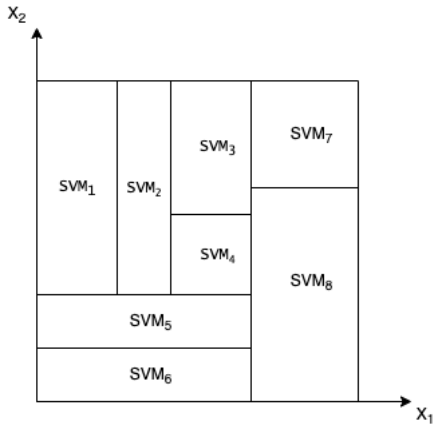


Figure 3.2: Random tree partition with SVM leaf nodes

In order to now construct the RSVT model, we simply extend the above-mentioned method to the ensemble level by constructing multiple of these random support vector trees. By that definition, the RSVT classifier is simply a random decision forest, albeit with a modified leaf node structure. Note that we do not rely on bagging operations to randomize the sample. Instead, we simply use the entire learning sample to construct each tree, just like with regular Extra-Trees. Nonetheless, the trees will be randomly different due to the highly randomized tree construction. Consequently, the resulting partitions are also randomly different from one another. This in turn randomizes the local regions on which the SVM operate.

To finally obtain the predictions of the trees, each of the leaf node SVM predicts the test samples that are sent to its leaf node, i.e. the test samples contained in the corresponding local region. Once more, the non-overlapping nature of the partition guarantees that every test sample will be exclusively predicted by its corresponding SVM. As such, each of the trees results in a vector of unique predictions. At the ensemble level, the test set predictions are aggregated by means of majority vote across the predictions of the individual trees, a common procedure in random forest models.

3.2 Tree Construction

During the construction of the totally randomized tree, each of its nodes is associated with a rectangular cell of the rescaled input space $X := [0, 1]^d$. With X as the root node of the tree, the input space is recursively partitioned into smaller cells along a randomly selected dimension of all d dimensions. This behaviour is achieved by simply setting the Extra-Trees parameter $K = 1$, resulting in a totally randomized tree. As our theoretical results will show, this totally random tree construction facilitates the derivation of its consistency. Note furthermore that the splits are always axis-aligned, i.e. perpendicular to the selected dimension such as depicted

above (Fig. 3.2). Alternatively, more complex splits could be used for $K \geq 2$, e.g. oblique or non-linear splits (Criminisi et al., 2011).

As to manage the continuation of the tree growing procedure, we specify a minimum leaf size, i.e. the minimum number of samples to retain in each of the leaf nodes. We denote this minimum leaf size by ℓ_n. As such, each of the leaf nodes will have at least ℓ_n samples. In our model, ℓ_n depends on the training sample in the following way (3.1).

$$\ell_n \to \infty \quad \text{and} \quad \ell_n/n \to 0 \quad \text{as} \quad n \to \infty \tag{3.1}$$

Most importantly, this relationship is instrumental towards establishing the theoretical consistency of the random tree. Furthermore, each local region is guaranteed to contain a sufficient amount of samples for SVM training, at least asymptotically. Furthermore, we specify the minimum leaf size to be $\ell_n \propto \sqrt{n}$. The reason being is that this satisfies the asymptotic relation between the minimum leaf size and the training sample. Indeed, $\sqrt{n} \to \infty$ and $\sqrt{n}/n \to 0$ as $n \to \infty$. In order to implement $\ell_n \propto \sqrt{n}$ in practice, we introduce a proportionality constant p such that $\ell_n = p\sqrt{n}$ enforces the desired proportionality. By doing so, p becomes a parameter to be optimized in empirical settings. The effective number of samples in each leaf is denoted as n_l, with l the leaf node index.

3.3 Leaf Node SVM

After constructing the above-mentioned decision tree, we easily obtain the leaf nodes and data contained therein, each having at least ℓ_n samples. Throughout its construction, the decision tree also preserves the dimensionality of the input space for each of the leaf nodes. The leaf node subsets are hence characterized as (3.2)

$$\{x_i, y_i\}_{i=1}^{l_n}, x_i \in \mathbb{R}^d, y_i \in \mathbb{R} \tag{3.2}$$

By calling on the independent nature of these training samples, the leaf node SVM can be straightforwardly trained in an isolated manner. As a result, the leaf nodes will constitute a collection of linear classifiers (3.3)

$$y_l(x) = \text{sign}\left[w_l^T x_l + b_l\right], \quad l = 1, \ldots, L \tag{3.3}$$

or alternatively in the dual formulation

$$y_l(x) = \text{sign}\left[\sum_{i=1}^{n_l} \alpha_{i,l} y_{i,l} x_{i,l}^T x_{i,l} + b_l\right], \quad l = 1, \ldots, L \tag{3.4}$$

To now obtain the (3.3) or (3.4), we must solve the related optimization problems, which all correspond to the problem as discussed in the literature (2.2). For reasons of computational efficiency, we rely on LibLinear (Fan et al., 2008) to solve the SVM optimization problem. LibLinear offers two methods for solving the SVM optimization problem. On one hand, the newton method from Lin et al. (2007) solves the primal problem, while the coordinate descent approach (Hsieh et al., 2008) solves the dual problem. Depending on the characteristics

of the data within the leaf nodes, we will either solve the primal or the dual problem. More specifically, we solve the primal problem if $n_l \geq d$, whereas we solve the dual problem otherwise. In other words, the primal is solved whenever the leaf node sample size is greater than the dimensionality of the input space. Note furthermore that the newton method only supports the L2-loss function, whereas coordinate descent supports both L1- and L2-loss functions. We hence opt for an L2-loss since it is available for both the primal and dual problem.

Next, we ideally want to optimize the parameters of the linear SVM on an individual basis as to customize each model to its respective region. In terms of parameters, LibLinear simply involves the regularization parameter C. As such, it is completely analogous to standard SVM in this regard. The authors note that their method is not very sensitive to C, decreasingly so for larger values. Furthermore, they note that the larger C becomes, the higher the computational cost, while model performances remain similar. As their results have shown, the range $\log_2 C \in [-2, 2]$ gives the best cross-validation accuracy (Fan et al., 2008). As such, we limit our search space according to these findings. In order to now determine the appropriate C value, we simply conduct an exhaustive grid search over the search space. Alternatively, it is also possible to use more efficient techniques such as randomized search (Bergstra and Bengio, 2012). However, considering the parameter space is rather small, exhaustive grid search is still feasible. With the generalization of our leaf node SVM in mind, we rely on cross-validation during the above-mentioned optimization procedure.

3.4 RSVT Ensemble

The extension to the ensemble level is rather straightforward. Indeed, it simply involves constructing M random support vector trees. In other words, we obtain M random partitions on which the local SVM are trained. Once more, each of the individual trees considers the entire training sample for its construction. Taking into account our rather complex model structure, the size of the forest should be kept relatively small to achieve the desired performance, at least compared to other decision forest such as random forests or extremely randomized trees. As the empirical results will show, a modest forest of $M \in [1, 10]$ is sufficient.

For M trees, we obtain matrix $\hat{Y}$ with the test predictions of the individual random support vector trees. The prediction for the i-th test observation then becomes the mode, i.e. majority vote, across the columns of the corresponding i-th row. As such, we ultimately end up with a vector $\hat{\mathbf{y}} = (\hat{y_1}, \ldots, \hat{y_n})^T$ containing the ensemble predictions.

3.5 Additional Considerations

3.5.1 Data Transformation

Before training the RSVT classifier, or any SVM-based method for that matter, it is important to rescale the data to an appropriate range as SVMs are not scale-insensitive. Furthermore, the SVM optimization is less prone to numerical instabilities when using rescaled data (Hsu et al., 2003). The most common methods to do so are either standardization or normalization. Whereas standardization results in data with have zero mean and unit variance, normalization rescales the input space to the $[0, 1]^d$ range, with d being the dimension of that space. In

line with our theoretical derivations (section 4.1), we recommend adopting the normalization approach.

3.5.2 Multi-class Settings

With regards to multi-class settings, there is an additional consideration to be made, namely the strategy used to decompose the problem into a set of binary problems. On one hand, one could adopt a one-versus-one approach, which constructs $K(K-1)/2$ classifiers for K amount of classes. On the other hand, the one-versus-all simply trains K classifiers for the same amount of classes (Hsu and Lin, 2002). From a computational point of view, the one-versus-all scheme is most appealing. Note that there also exists a theoretically consistent scheme known as the "Crammer-Singer" approach, which establishes a way to directly treat the multi-class problem rather than to convert it into a series of binary problems (Crammer and Singer, 2002). However, this is known to be very computationally demanding. Upon experimenting with the method, it has not improved the performance of our model. We therefore resort to the one-versus-all scheme.

3.5.3 Class Imbalance

Recall that the decision tree, and hence also the leaf nodes, are entirely random. As such, it is highly unlikely that the class distribution of the root node is preserved in each of leaf nodes. In other words, the resulting partition yields local regions that are prone to issues of class imbalance. Most importantly, we hope to mitigate this problem by means of the ensemble model. However, as to also resolve this problem at the level of the individual trees, we implement a weighted regularization mechanism for each of the leaf node SVM. That is, the regularization constant of each leaf node is adjusted inversely proportional to the class distribution of that particular leaf node. In other words, a class with a higher frequency is less strenuously regularized and vice versa. To do so, we introduce a class-specific weight $\nu_l^{(k)}$ (3.5).

$$\begin{aligned} C_{k,l} &= \nu_l^{(k)} C_l, \quad k = 1, \ldots, K \quad l = 1, \ldots, L \\ &= \frac{n_l}{K \sum_{i=1}^{n_l} \mathbf{1}_{(k)}(y_{i,l})} C_l \end{aligned} \tag{3.5}$$

where the number of classes is denoted as K and $\mathbf{1}_{(k)}(.)$ the indicator function returning 1 for an instance in y_l belonging to class k and 0 otherwise. As such, $C_{k,l}$ is the regularization parameter for the k-th class in the l-th leaf node.

Note finally that, in the extreme case that a leaf node is pure, i.e. contains instances of only a single class, it is not possible to proceed with SVM training. Instead, we rely on the predictions of the totally randomized tree for those particular instances, simply being the unique class label of that pure leaf node. Considering these predictions follow from a totally randomized tree, the ensemble approach is motivated once more as to mitigate the possibly sub-optimal predictions of these pure leaf nodes.

3.5.4 Parallel Computation

With most decision forests, the independent nature of the individual trees allows them to be trained in parallel, dramatically decreasing the runtime as such. Fortunately, the inter-tree independence also applies to RSVT. Consequently, we process the individual trees in parallel. However, RSVT also requires the leaf node SVM to be trained. By calling on the independent nature of the leaf nodes, the embedded SVM can also be processed in parallel, further improving the computational efficiency of our model. As such, the parallelization of our model takes place at the ensemble level as well as the leaf node level.

Chapter 4

Consistency

4.1 Preliminaries

For a sequence of classifiers to be consistent, its error probability must converge to the Bayes risk in probability. With this in mind, let us first define both the error probability of a sequence of classifiers as well as Bayes risk. To do so, we rely on the important work of Devroye et al. (1996). According to this work, a classifier g has error probability L, defined as follows

$$L(g) = \mathbf{P}\{g(X) \neq Y\}$$

where $X \in \mathbb{R}^d$ represents the feature vector and $Y \in \{c_1, \ldots, c_n\}$ the class labels. In other words, it gives us an idea about the probability of error with respect to a pair (X, Y). The best possible classifier g^* is then defined as

$$g^* = \underset{g:\mathbb{R}^d \to \{c_1,\ldots,c_n\}}{\arg\min} \mathbf{P}\{g(X) \neq Y\}$$

, i.e. the classifier that produces the minimal probability of error, also referred to as the Bayes classifier. The risk associated with the Bayes classifier then corresponds to $L^* = L\left(g^*\right)$. Alternatively, Bayes classifier can also be defined as

$$g^*(x) = 1_{\{\eta(x) \geq 1/2\}}$$

where $\eta(x) = \mathbb{P}\{Y = 1 \mid X = x\}$ denotes the a posteriori estimates of correctly predicting the positive class label.

From the important work of Biau et al. (2008), it follows that a sequence of classifiers g_n is consistent if $L\left(g_n\right) \to L^*$ in probability. Equivalently, this means that $\mathrm{E}[L] \to L^*$ as $n \to \infty$ should hold. For this to be the case, it is required that both $\mathrm{diam}(A_n(\boldsymbol{X})) \to 0$ and $N\left(A_n(\boldsymbol{X})\right) \to \infty$ in probability. Note that $\mathrm{diam}(A)$ denotes the diameter of a set A. Furthermore, $A_n(\boldsymbol{X})$ denotes the leaf node that contains X and $N\left(A_n(\boldsymbol{X})\right)$ the number of data points falling into the particular leaf node (Devroye et al., 1996).

With these preliminaries, we can now continue by providing the lemma necessary to establish the consistency of our random forest model. In regard to Lemma 1 and 2, we respectively rely on Biau et al. (2008) and Denil et al. (2013). On the other hand, Lemma 3 is proved in this thesis.

Lemma 1. *Suppose a sequence of classifiers $\{g_n\}$ is consistent for a certain distribution of (X, Y). Then the classifier $g_n^{(M)}$ obtained by taking a majority vote over M randomly different copies of g is also consistent.*

Lemma 2. *Assume we have a posteriori estimates $\eta_n^c(x)$ for each class posterior, and that these are consistent estimates. In that case, the classifier*

$$g_n(x) = \arg\max_{m \in [M]} \{\eta_n^c(x)\}$$

is also consistent in multi-class classification settings.

In order to derive the consistency of a random forest classifier for binary classification, lemma 1 shows that it is sufficient to prove the consistency of the base classifier, i.e. a single classification tree (Biau et al., 2008). Lemma 2 extends this fact towards the multi-class setting, as shown by Denil et al. (2013).

Lemma 3. *For sufficiently large n, every node of the purely random tree will be split infinitely many times in probability.*

Proof. Recall that K is the distance from the root node to the leaf node the tree. For any given K and constant $\delta > 0$, the smallest child node of the root node in the tree has size $\min\{U, 1-U\}$ with a probability of at least $\delta^{1/K} < 1$

$$\begin{aligned} \mathrm{P}\left(\min\{U, 1-U\} \geq \delta^{1/K}\right) &= \mathrm{P}\left(\delta^{1/K} \leq U \leq 1 - \delta^{1/K}\right) \\ &= 1 - 2\delta^{1/K} \end{aligned}$$

Next, the range of every dimension is scaled to [0,1], without loss of generality. Repeating the argument for K times, we have that after K splits, all sides of the child nodes have length of at least δ with probability at least $\left(1 - 2\delta^{1/K}\right)^K$. However, this bound is derived under the assumption that the same dimension is split at each level of the tree. If different dimensions are split however, the probability that all sides have at least length δ is greater than $(1-2\delta^{1/K})^K$. Moreover, it holds that $(1-2\delta^{1/K})^K \to 1$ as $\delta \to 0$. Therefore, for any $\epsilon > 0$, there exists $\delta < \left(\left(1-(1-\epsilon)^{1/K}\right)/2\right)^K$ such that $\left(1-2\delta^{1/K}\right)^K \geq 1-\epsilon$. Consequently, we know that every node at depth K has all sides lengths at least δ with probability $1-\epsilon$. Since the distribution of X has a non-zero density almost everywhere, each of these nodes has a positive measure with respect to P_X. By defining

$$p = \min_{L\text{: a leaf at depth } K} \mathrm{P}_X(L)$$

we know that $p > 0$ since the minimum is over finitely many leaf nodes of the tree, which all contain a positive measure for P_X.

It remains to show that we can choose n large enough so that any set $\mathrm{A} \subset [0,1]^d$ with $\mathrm{P}_X(A) > p$ contains at least ℓ_n sample points. To that end, fix an arbitrary $\subset [0,1]^d$ with $\mathrm{P}_X(A) = p$. In a data set of size n, the number of sample points fallen into A is $N(A) \sim$ Binomial (n, p). By making use of Hoeffding's inequality, we have

$$\begin{aligned}\mathrm{P}\left(N(A)<\ell_n\right)&=\mathrm{P}\left(\frac{N(A)-np}{n}<\frac{\ell_n-np}{n}\right)\leq\exp\left(-\frac{2}{n}\left(\ell_n-np\right)^2\right)\\&=\exp\left(2\left(\ell_n-np\right)\left(p-\ell_n/n\right)\right)\leq\exp\left(2\left(\ell_n-np\right)p\right)\end{aligned}$$

The last inequality holds due to the fact that $\ell_n - np < 0$ when n is large enough. For this probability to be upper bounded by an arbitrary $\varepsilon > 0$, it is sufficient to have

$$\frac{\ell_n}{n}<p-\frac{1}{2np}\log\left(\frac{1}{\varepsilon}\right)$$

.

Since the second term on the RHS goes to zero as $n \to \infty$, the RHS is positive for sufficiently large n. Moreover, since $\ell_n/n \to 0$ as $n \to \infty$, it is always possible to choose a sufficiently large n to satisfy this inequality. To conclude, we have proved that for any $K > 0$, the probability that any leaf at depth K contains at least ℓ_n sample points can arbitrarily large for $n \to \infty$. If the number of samples in the node is larger than n, the top tree will continue to grow. Therefore we have that, by making n large enough, we can make the probability that all branches of the top tree are actually grown to depth K, arbitrarily high. As a result, we have $K \to \infty$ in probability.

□

4.2 Consistency Theorem

Theorem. *suppose that X is supported on $[0,1]^d$ and has non-zero density almost everywhere, then the embedded random forest is consistent provided that $\ell_n \to \infty$ and $\ell_n/n \to 0$.*

Proof. On account that the forest has an embedded structure, we denote $A_n(X)$ the leaf of the purely random tree X falls into. To show the consistency of the base estimators of the embedded forest, it is required that both $N\left(A_n(X)\right) \to \infty$ and $\mathrm{diam}(A_n(X)) \to 0$ as $n \to \infty$ (Devroye et al., 1996).

Firstly, the embedded forest has $N\left(A_n(X)\right) \geq \ell_n$ by construction. Moreover, it holds that $\ell_n \to \infty$ as $n \to \infty$. It is thus easily verified that $N\left(A_n(X)\right) \to \infty$ as $n \to \infty$.

Secondly, in order to prove that $\mathrm{diam}(A_n(X)) \to 0$ as $n \to \infty$, we let $V_i\left(A_n\right)$ denote the side length in the i-th dimension of the rectangular leaf $A_n(X)$. It suffices to show that for all $i \in \{1, 2, ..., d\}$, we have $\mathbb{E}\left(V_i\left(A_n\right)\right) \to 0$. Recall that the construction of the tree follows a purely random splitting criterion. Therefore, given the number of splits coming from the i-th dimension that is used to generate $A_n(X)$, denoted by T, we have

$$\begin{aligned}\mathbb{E}\left(V_i\left(A_n\right)|T\right)\leq\mathbb{E}\Pi_{j=1}^{T}\max\left\{U_j,1-U_j\right\}&=\left(\mathbb{E}_U\max(U,1-U)\right)^T\\&=(3/4)^T\end{aligned}$$

where U is the splitting point in the tree and $U \sim$ Uniform $[0, 1]$.

Assuming that K is the distance from the tree root to the leaf $A_n(X)$, in other words, the number of splits used to generate leaf $A_n(X)$, we have $T \sim$ Binomial $(K, 1/d)$ and

$$\begin{aligned}
\mathbb{E}\left(V_i\left(A_n\right)|K\right) &= \mathbb{E}\left(\mathbb{E}\left(V_i\left(A_n\right)|T\right)|K\right) \\
&= \sum_{t=0}^{K} \mathbb{E}\left(V_i\left(A_n\right)|T=t\right) \mathrm{P}(T=t|K) \\
&\leq \sum_{t=0}^{K} \left(\begin{array}{c} K \\ t \end{array} \right) \left(\frac{3}{4}\right)^t \left(\frac{1}{d}\right)^t \left(1-\frac{1}{d}\right)^{K-t} = \left(1-\frac{1}{4d}\right)^K
\end{aligned}$$

According to the law of total expectation, we then have

$$\begin{aligned}
\mathbb{E}\left(V_i\left(A_n\right)\right) &= \mathbb{E}\left(\mathbb{E}\left(V_i\left(A_n\right)|K\right)\right) \\
&= \mathbb{E}(1-1/(4d))^K
\end{aligned}$$

To finish the proof, it is sufficient for the embedded forest to have $K \to \infty$ in probability, which is shown in Lemma 4.

□

Chapter 5

Experiments

The RSVT classifier has been entirely written in the Python programming language. In doing so, we have mostly relied on the Scikit-learn [1] machine learning library (Version 0.23) and the Jupyter Notebook[2] environment. Furthermore, all experiments were conducted using commodity hardware (2.3 GHz Intel© Quad-core™ i5, 16GB RAM). These basic hardware requirements should allow any machine learning practitioner to verify the obtained results. The software implementation of RSVT can be consulted at the following repository. Besides RSVT, it also includes all the experiments that are conducted throughout this chapter.

https://github.com/VincentBuekers/Master-Thesis

5.1 Performance Benchmarks

5.1.1 Data

We have selected 10 data sets to evaluate the performance of the RSVT classifier, all of which are freely available at the UCI Machine Learning Repository[3] (Dua and Graff, 2017). A summary of the selected data sets can be consulted in Appendix A (Table A.1). The majority of this selection does not have predefined train and test samples. For those data sets, we generate train and test sets with a pre-specified seed as to allow for reproducible results. The corresponding code is reported in Appendix B (listing B.1).

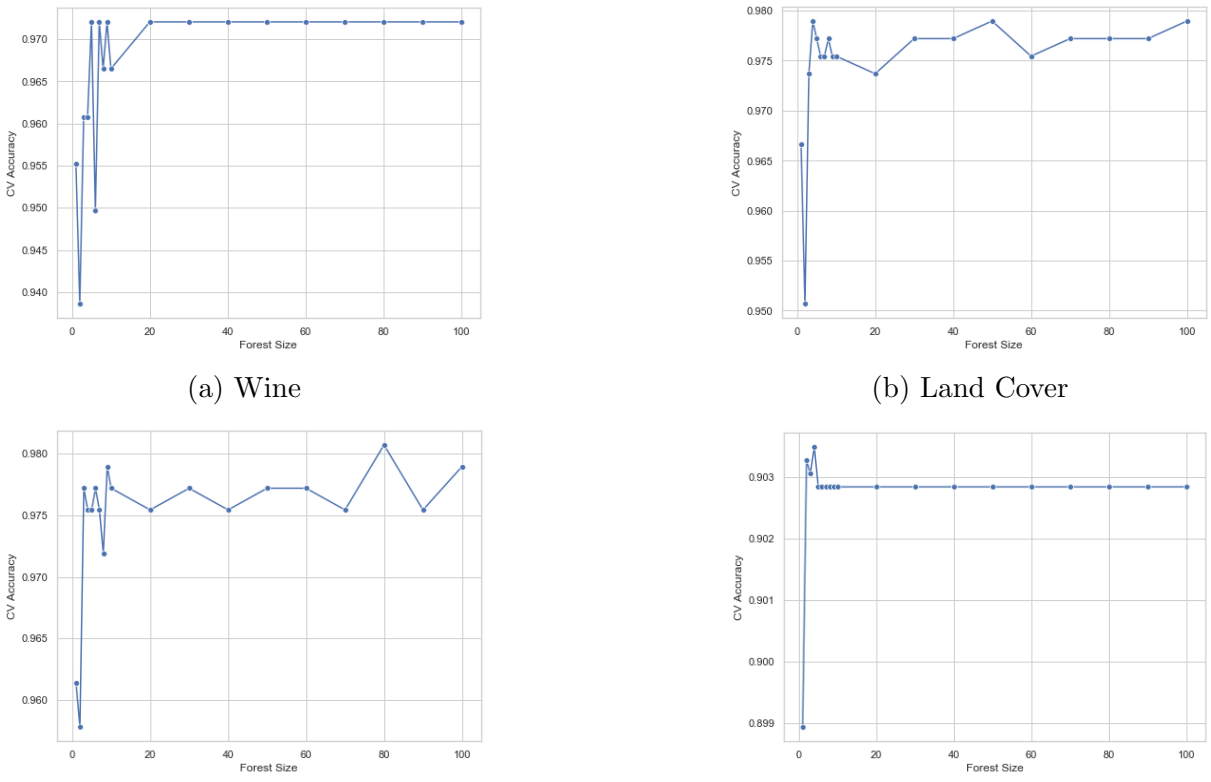
Most importantly, our selection constitutes a variety of classification problems as they substantially vary both in terms of size and dimensionality. Moreover, it includes problems that are both linear and non-linear in nature. Furthermore, they entail both binary and multi-class classification with varying amounts of classes. As such, the selection should be adequately representative to assess the performance of RSVT in different settings. However, we have restricted the problems to those exclusively containing numeric features. Lastly, it should be noted that we always transform the numeric features to the [0,1] range. This a very straightforward operation in Python. For reference, consult Appendix B (listing B.2).

[1] https://scikit-learn.org/stable/
[2] https://jupyter.org/
[3] https://archive.ics.uci.edu/ml/index.php

5.1.2 Experimental Conditions

We compare three different models on each of the selected data sets, namely random forests, regular Extra-Trees and our own RSVT. These are respectively referred to as RFC, ETC and RSVT in the subsequent analysis. As with RSVT, we rely on scikit-learn for the software implementation of RFC and ETC. Note that in order for the results to be reproducible, we fix the randomness of all models. Listing B.3, B.4 and B.5 of Appendix B respectively outline the corresponding programming for RFC, ETC and RSVT.

For the ensemble parameters, we adopt the standard amount of trees for both RFC and ETC, namely 100. In practical settings, the biggest performance gains are achieved within the first 100 trees, as shown by Probst and Boulesteix (2017). In contrast, RSVT only requires a small to moderate forest size to demonstrate the desired performance improvements. To demonstrate this, we test $M \in [1, 100]$ by means of 5-fold cross-validation for 4 example data sets. The code corresponding to this experiment is included in appendix B (listing B.6) As can be seen in Fig. 5.1, the most noticeable improvements occur in the [1,10] range. Past this point, the increase in cross-validation accuracy is practically negligible. Since RSVT exhibited very similar behaviour on the remaining data sets, we simply resort to setting $M = 10$ every data set.



(a) Wine

(b) Land Cover

(c) Image Segmentation

(d) Optical Digit Recognition

Figure 5.1: Number of trees Vs. CV Accuracy

As for the number of attributes to consider, RSVT only considers a single attribute by construction. For RFC and ETC, we instead adopt the default setting, namely $K = \sqrt{n}$. Lastly, we need to consider the minimum amount of leaf nodes to consider. Both Breiman

(2001) and Geurts et al. (2006) use a minimum leaf size of 5 by default. As such, we follow their suggestion in this regard. For RSVT on the other hand, we must abide by $\ell_n \propto \sqrt{n}$ for theoretical reasons. Recall that we introduced a proportionality factor p such that ℓ_n is governed by $p\sqrt{n}$. In our experiments, we evaluate 10 evenly spaced values of p in the interval $[1/3, 3]$. Once again we do so by means of 5-fold cross-validation. Table 5.1 reports the sample variance s^2 of the cross-validated accuracy for the different p. As it turns out, the proportionality factor has virtually no influence on the performance of RSVT. We therefore just adopt $p = 1$ for our benchmark results. As such, we simply have $\ell_n = \sqrt{n}$. Fortunately, this effectively eliminates an otherwise inconvenient parameter. Note that the the code needed to produce these results can be found in Appendix B (listing B.7).

Data set	s^2_{ACC}
Wine	2.37e-06
Breast Cancer (D)	2.77e-06
Urban Land Cover	0.44e-02
Statlog (Vehicle)	3.61e-05
Contraceptive	5.53e-05
Image segmentation	3.81e-05
Madelon	9.57e-06
Spambase	2.15e-08
Optical digit recognition	6.93e-05
Isolet	4.42e-05

Table 5.1: Sample Variance of CV Accuracy for different p

In addition to the ensemble parameters, RSVT also entails the SVM regularization parameter C in each of the leaf nodes of the component trees. As mentioned in section 3.3, we follow the range of C values suggested by Fan et al. (2008), namely $\log_2 C \in [-2, 2]$. More specifically, we evenly space 5 values within the range of this $\log_2$ scale, resulting in C among $\{0.25, 0.5, 1, 2, 4\}$. The best parameter is determined by means of cross-validation on the leaf node sample. However, we choose to restrict this to 3-fold cross-validation so that the leaf node models remains computationally feasible. Each leaf node SVM then automatically adopts the value of C resulting in the cross-validation accuracy. Taking into account the embedded model structure, manually tuning C for every leaf node would be quite troublesome.

5.1.3 Results

Table 5.2 reports the performance benchmarks for RFC, ETC and RSVT in terms of its classification accuracy (%). Based on these results, RSVT has been able to compete with state-of-the-art algorithms such as RFC and ETC. Indeed, RSVT performs best on 4 out of the 10 data sets, which is on par with ETC. Even when RSVT does not perform best, it is still almost as accurate as the competing models. Surprisingly, RFC only performs best on 2 data sets. For RSVT, the most problematic data sets are *Urban Land Cover* and *Madelon*. Note that both data sets have been purposely included as it was expected that RSVT would be prone to the respective data characteristics. For the *Urban Land Cover* data, the performance drop is likely due to the limited number of training samples per class. This is particularly

difficult as it implies that the leaf node SVM will operate on even smaller samples. On the other hand, the *Madelon* data set constitutes an entirely different challenge. The problem here is instead is that the data contain a lot of uninformative attributes while also being highly non-linear. For such problems, the linear SVM might still be inadequate even though they operate on a local, possibly less non-linear region. However, recall that the local regions are formed by means of axis-aligned splits. For highly non-linear problems, alternative splits, e.g. oblique or non-linear, might be more appropriate.

Data set	Acc_{RFC}	Acc_{ETC}	Acc_{RSVT}
Wine	**96.67**	95.00	95.00
Breast Cancer (D)	95.26	95.79	**96.84**
Urban Land Cover	81.07	**82.45**	73.96
Statlog (Vehicle)	72.34	71.99	**74.11**
Contraceptive	51.32	50.10	**52.34**
Image segmentation	96.88	**97.01**	92.08
Madelon	**70.33**	67.33	58.00
Spambase	95.70	**95.76**	93.94
Optical digit recognition	96.66	**98.50**	96.66
Isolet	94.23	95.13	**95.83**

Table 5.2: Accuracy on benchmark data for RFC, ETC and RSVT

5.2 Training Time

As we have devoted considerable attention to the computational efficiency of RSVT, we lastly want to evaluate the training time of our model. To that end, we rely on artificially generated data of varying nature. As it happens, scikit-learn offers a highly customizable data generator for classification problems. These data sets resemble real-world rather than toy problems by following the experimental data design of Guyon (2003). Note that Appendix B (listing B.8) outlines the python code to produce this experiment.

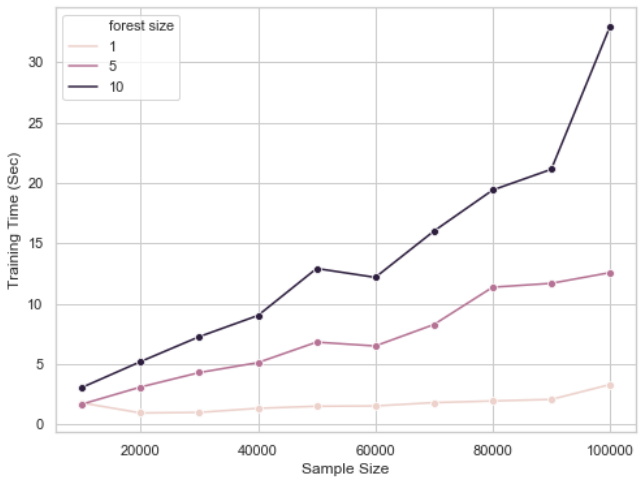
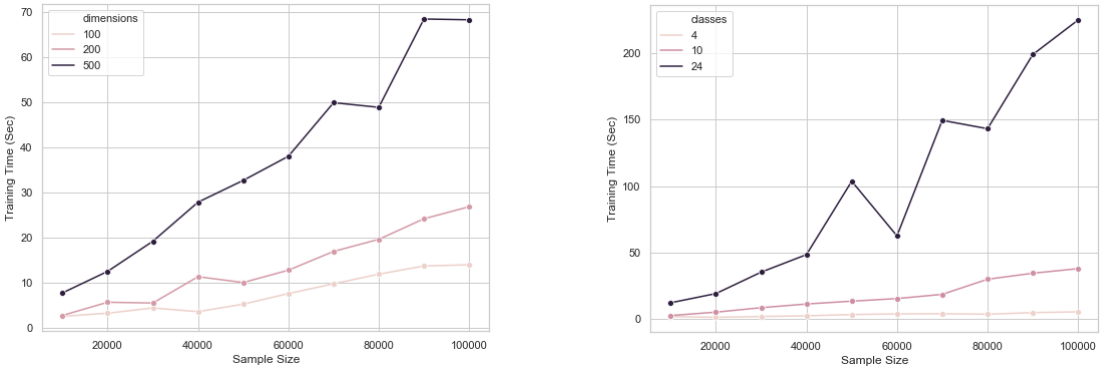


Figure 5.2: RSVT training time (1)

Firstly, we look at the extent to which the sample size affects the training time of RSVT. Moreover, we evaluate our ensemble for different sizes. In line with our performance results, we test modest ensemble sizes, namely M among $\{1, 5, 10\}$. As can be seen in Fig. 5.2, a single decision tree remains quite efficient regardless of the sample size. In contrast, training multiple random support vector trees does impose quite substantial computational requirements. Nonetheless, the training time still remains feasible in absolute terms. Indeed, training 10 trees only takes about 20 seconds for a training sample consisting of 80,000 instances. However, it is important to note that these runtimes follow from a binary classification problem of input dimension $d = 20$

To gain more insight into how the dimensionality affects the training time of RSVT, we test d among $\{100, 200, 500\}$ for increasing sample sizes. As to isolate the effect, we only evaluate a single support vector tree. As it turns out, the dimension of the input space imposes more serious requirements on the runtime of our algorithm. For instance, training a single tree on 100,000 instances almost takes 70 seconds if the input space has $d = 500$ (Fig. 5.3, a). Nonetheless, training times remain manageable for data sets of moderate dimension, even for large sample sizes. Lastly, we have looked at the computational demands of multi-class problems. To that end, we evaluate number of classes among $\{4, 10, 24\}$ for a single support vector tree on increasing sample sizes. As can be seen in Fig 5.3 (a), higher amounts of classes do in fact lead up to quite infeasible training times. For a problem consisting of 24 classes and 90,000 training samples, it takes about 200 seconds to train a single tree. In contrast, moderate amounts of classes such as 4 or 10 pose less of a problem. However, it should be taken into account that these training times follow from training a single tree. With our recommended forest size of 10, large-scale problems are likely not feasible in practice. Among our benchmark data sets, only *Isolet* had both high class count (26) and high dimensionality (617). This indeed lead to a very substantial training time for a forest size of 10.



(a) Varying dimensions

(b) Varying amounts of classes

Figure 5.3: RSVT training time (2)

Chapter 6

Discussion

With both the theoretical and empirical results in hand, we can proceed with discussing RSVT from both perspectives. As for the theoretical results, we have shown that, for sufficiently large sample size, every node of our random tree will be split infinitely many times in probability. Furthermore, the leaf nodes also exhibit the desired asymptotic behaviour, namely that the samples contained therein grow proportionally to the entire sample. As such, we know that each of our trees define asymptotically consistent partitions of the input space. That is, as the data grow infinitely large, the accuracy of the partition converges to the optimum, being the Bayes risk. As it turns out, randomizing the tree construction has thus shown to be instructive towards deriving the consistency results of our trees. This, combined with the required minimum leaf size constitutes a relatively straightforward strategy on the theoretical front.

However, the above-mentioned asymptotic behaviour does not occur in practice. Consequently, our completely random trees would have resulted in sub-optimal classification performance if not for the SVM leaf nodes. Based on the performance benchmarks, the leaf node SVM approach has been quite successful for a variety of classification problems. On the selected data sets, RSVT can often compete with state-of-the-art random forests such as the original random forest (Breiman, 2001) and extremely randomized trees (Geurts et al., 2006). However, the RSVT architecture remains susceptible to certain issues. On one hand, highly imbalanced data sets can cause problems for the leaf node SVM as they will operate on even smaller, and likely even more imbalanced subsamples. On the other hand, RSVT could have trouble modelling highly non-linear problems. Regardless of the piece-wise linearity of RSVT, such problems are likely better dealt with by kernel SVM. However, upon experimenting with kernel SVM, the performance only improved marginally, if at all. Taking into account the increased model complexity and corresponding computational demands, we have chosen to abandon kernel SVM leaf nodes. Alternatively, each leaf node could have evaluated a number of different kernels, among which the optimal one is determined by means of (cross)-validation. However, this would once again impose computational difficulties for the model overall. As such, linear SVM remain the most justifiable approach within the proposed model architecture. Indeed, it does not involve kernel operations or any additional parameters to consider. Moreover, it allows to rely on highly efficient solvers such as LibLinear (Fan et al., 2008). Note that we have also experimented with SVM-SGD (Bottou, 2007) to (approximately) solve the optimization problem. From a computational point of view, the stochastic gradient descent method proved to be as fast as LibLinear, if not faster. Nonetheless, the

theoretical foundations in relation to SVM are still fairly limited. LibLinear therefore better fits the objectives of this thesis. It should furthermore be noted that the leaf node SVM have not yet been integrated into the theoretical results. To do so, one would have to take into account the notions of statistical learning theory (Vapnik, 1995) in addition to the consistency of the forest. Currently, Chang et al. (2010) have made significant progress in deriving the generalization error bounds of a model that is very similar to RSVT, albeit with an entirely different motivation. However, combining decision trees and SVM into a single theoretical framework is an entirely different challenge, far beyond the scope of this thesis. Nonetheless, it could be an interesting topic for future research. In the context of this, the SVM thus simply act as a value assignment mechanism within each of the leaf nodes.

While the SVM leaf nodes have allowed RSVT to achieve competitive performance, they also come with great computational demands. As a countermeasure, we have taken great effort to speed up the run-time of our algorithm. For small to medium sized problems, RSVT is perfectly feasible to train. On the other hand, the training time can quickly escalate in certain situations. Particularly for high-dimensional spaces, as well as multi-class problems with a high amount of classes, one should be aware that training RSVT could take a considerable amount of time, likely infeasible for most machine learning practitioners. Further improving the efficiency of RSVT would likely require a more efficient software implementation. Instead of using a high-level language such as `Python`, a compiled programming languages (e.g. `C++`) would already go a long way.

Chapter 7

Conclusion

In this thesis, we have presented a novel random forest ensemble called Random Support Vector Trees (RSVT). On one hand, we randomize our tree construction to the greatest extent by relying on the totally randomized variant of Extra-Trees. Randomizing the tree construction is a straightforward approach to make the individual decision trees less data-dependent. Indeed, it has in turn allowed us to derive the consistency of our random trees. In contrast, this property has not yet been established for Breiman's original random forest due to the aforementioned data-dependency. Nonetheless, our advances on the theoretical front do not guarantee good performance in practice. As a countermeasure, we have modified the leaf nodes of our trees by using support vector machines as a second stage value assignment mechanism. From our benchmark results, this strategy has proven successful in attaining a more competitive classification accuracy, which has often been a challenge for theoretically consistent random forest variants. Nonetheless, the pursuit of performance has presented considerable computational difficulties. We have taken great effort to reduce these difficulties such that our algorithm retains its practical appeal. In the end, we have been able to diminish these difficulties to a large extent.While RSVT can be efficiently trained on small to medium sized problems, it can still not be considered a viable large-scale machine learning model.

All things considered, the objectives of this thesis have been largely achieved. Most importantly, RSVT possesses the desired theoretical consistency (1). Furthermore, the proposed model also achieves competitive performance results (2), we have reduced the computational difficulties as much as possible (3). In the end, RSVT is by no means a general-purpose classifier. Instead, it constitutes a random forest variant with nice theoretical properties while also attaining competitive performance on most types of classification problems. With this in mind, we hope to have made further progress towards reconciling random forests in theory and practice.

Bibliography

Amit, Y. and Geman, D. (1997). Shape quantization and recognition with randomized trees. *Neural Comput.*, 9(7):1545–1588.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305.

Biau, G. (2012). Analysis of a random forests model. *J. Mach. Learn. Res.*, 13:1063–1095.

Biau, G., Devroye, L., and Lugosi, G. (2008). Consistency of random forests and other averaging classifiers. *J. Mach. Learn. Res.*, 9:2015–2033.

Bottou, L. (1991). Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nîmes 91*, Nimes, France. EC2.

Bottou, L. (2007). Stochastic gradient descent (v.2).

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In Lechevallier, Y. and Saporta, G., editors, *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pages 177–187, Paris, France. Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA.

Burges, C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167.

Chang, F., Guo, C.-Y., Lin, X.-R., and Lu, C.-J. (2010). Tree decomposition for large-scale svm problems. *J. Mach. Learn. Res.*, 11:2935–2972.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Mach. Learn.*, 20(3):273–297.

Crammer, K. and Singer, Y. (2002). On the algorithmic implementation of multiclass kernel-based vector machines. *J. Mach. Learn. Res.*, 2:265–292.

Criminisi, A., Shotton, J., and Konukoglu, E. (2011). Decision forests for classifcation, regression, density estimation, manifold learning and semi-supervised learning. 144.

Denil, M., Matheson, D., and de Freitas, N. (2013). Consistency of online random forests.

Denil, M., Matheson, D., and Freitas, N. D. (2014). Narrowing the gap: Random forests in theory and in practice. 32(1):665–673.

Devroye, L., Györfi, L., and Lugosi, G. (1996). *A Probablistic Theory of Pattern Recognition*, volume 31.

Dietterich, T. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Mach. Learn.*, 40.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *Ann. Statist.*, 7(1):1–26.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, page 148–156, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Mach. Learn.*, 63(1):3–42.

Geurts, P. and Wehenkel, L. (2000). Investigation and reduction of discretization variance in decision tree induction. volume 1810, pages 162–170.

Guyon, I. (2003). Design of experiments for the nips 2003 variable selection benchmark.

Hang, H., Chen, Y., and Suykens, J. A. K. (2019). Two-stage best-scored random forest for large-scale regression.

Hastie, T., Rosset, S., Tibshirani, R., and Zhu, J. (2004). The entire regularization path for the support vector machine. *Journal of Machine Learning Research*, 5:1391–1415.

Ho, T. K. (1998). The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20:832–844.

Hsieh, C.-J., Chang, K.-W., Lin, C.-J., Keerthi, S. S., and Sundararajan, S. (2008). A dual coordinate descent method for large-scale linear svm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 408–415, New York, NY, USA. Association for Computing Machinery.

Hsu, C.-w., Chang, C.-c., and Lin, C.-J. (2003). A practical guide to support vector classification chih-wei hsu, chih-chung chang, and chih-jen lin.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

Joachims, T. (1999). Making large-scale SVM learning practical. In Schölkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods - Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA.

Joachims, T. (2006). Training linear svms in linear time. volume 2006, pages 217–226.

Koshiba, Y. and Abe, S. (2003). Comparison of l1 and l2 support vector machines. volume 3, pages 2054 – 2059 vol.3.

Lin, C.-J., Weng, R., and Keerthi, S. (2007). Trust region newton method for large-scale logistic regression. volume 9, pages 561–568.

Louppe, G. (2014). Understanding random forests: From theory to practice.

Menon, A. K. (2009). Large-scale support vector machines: Algorithms and theory.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14.

Probst, P. and Boulesteix, A.-L. (2017). To tune or not to tune the number of trees in random forest. *J. Mach. Learn. Res.*, 18(1):6673–6690.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Shalev-Shwartz, S., Singer, Y., and Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 807–814, New York, NY, USA. Association for Computing Machinery.

Suykens, J. A., Van Gestel, T., De Brabanter, J., De Moor, B., and Vandewalle, J. (2002). *Least Squares Support Vector Machines*. World Scientific.

Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag, Berlin, Heidelberg.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg.

Wang, Y., Tang, Q., Xia, S.-T., Wu, J., and Zhu, X. (2016). Bernoulli random forests: Closing the gap between theoretical consistency and empirical soundness. pages 2167–2173.

Wehenkel, L. (1997). Discretization of continuous attributes for supervised learning. variance evaluation and variance reduction. *Proceedings of the Seventh International Fuzzy Systems Association World Congress*, 1:381–388.

Appendices

Appendix A

Supplementary data on Experiments

Data set	Size	Dimensionality	Class count
Wine	178	13	3
Breast Cancer (D)	569	32	2
Urban Land Cover*	675	148	9
Statlog (Vehicle)	946	18	4
Contraceptive	1473	9	3
Image segmentation	2310	19	7
Madelon	2600	500	2
Spambase	4601	57	2
Optical digit recognition	5620	64	10
Isolet*	7797	617	26

Table A.1: Overview of benchmark data

Note that the data sets marked with an asterisk (*) have predefined training and testing sets. For the other data sets, we rely on the built-in function `train_test_split()` from the scikit-learn package. More specifically, we use $2/3$ of the sample for training and hence $1/3$ for testing. This is simply achieved by setting `test_size=1/3`. Furthermore, we want our results to be reproducible, meaning we have to fix the randomness involved in splitting the data into training and testing parts. To do so, we set `random_state=754046` (my KUL student number). Furthermore, it is desirable to preserve the class distribution across both the training and testing sample. To that end, we opt for a stratified split, which can simply be implemented by setting `stratify=y`. The corresponding code can be found in Appendix B, listing B.1.

Appendix B

Python Code Snippets

Listing B.1: Splitting data into training and testing sets with scikit-learn

```
from sklearn.preprocessing import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y,
                                                    test_size=1/3,
                                                    stratify=y,
                                                    random_state=754046)
```

Listing B.2: Normalizing data to 0,1 range with scikit-learn

```
from sklearn.preprocessing import MinMaxScaler()

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Listing B.3: Training, testing and evaluating Random Forests in scikit-learn

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

RFC = RandomForestClassifier(random_state=754046)
RFC.fit(X_train, y_train)
preds_RFC = RFC.predict(X_test)
acc_RFC = accuracy_score(y_test, preds_RFC)
```

Listing B.4: Training, testing and evaluating Extremely Randomized Trees in scikit-learn

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score

ETC = ExtraTreesClassifier(random_state=754046)
ETC.fit(X_train, y_train)
preds_ETC = ETC.predict(X_test)
acc_ETC = accuracy_score(y_test, preds_ETC)
```

Listing B.5: Training, testing and evaluating RSVT

```
from sklearn.metrics import accuracy_score

# load RSVT functions
%run RSVT.ipynb

# partition input space
partitions = extra_partition(X_train,X_test, y_train,random_state=754046)

# fit SVMs to leaf nodes
forest = fit_rf_svc(partitions)

# predict test samples
preds = predict_forest(partitions,forest)
acc_RSVT = accuracy_score(y_test, preds)
```

Note that the test sample is also passed to the `extra_partition()` function such that the testing samples are distributed to the leaf nodes according to the tree-structure as obtained through the training samples. This is to say, the function does not utilize the test sample to construct the partition.

Listing B.6: Experiment on forest size

```
import numpy as np
from sklearn import metrics, preprocessing, model_selection

%run "RSVT.ipynb"

n_trees_1 = list(np.linspace(1,10,10))
n_trees_2 = list(np.linspace(20,100,9))
n_trees = n_trees_1+ n_trees_2
n_trees = [int(i) for i in n_trees]

folds = model_selection.StratifiedKFold(n_splits=5)

for m in n_trees:

    acc = []

    for train_index, test_index in folds.split(X, y):

        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        scaler = preprocessing.MinMaxScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        partitions = extra_partition(X_train,X_test, y_train, n_estimators=m)
        forest = fit_rf_svc(partitions)

        preds = predict_forest(partitions,forest)
        acc.append(metrics.accuracy_score(y_test,preds))

    cv_acc = np.mean(acc)
```

Listing B.7: Experiment on proportionality factor p

```
import numpy as np
import statistics
from sklearn import metrics, preprocessing, model_selection

%run "RSVT.ipynb"

sample_factors = np.linspace(1/3,3, 10)

folds = model_selection.StratifiedKFold(n_splits=5)

cv_acc = []

for sample_factor in sample_factors:

    acc = []

    for train_index, test_index in folds.split(X, y):

        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

        scaler = preprocessing.MinMaxScaler()
        X_train = scaler.fit_transform(X_train)
        X_test = scaler.transform(X_test)

        partitions = extra_partition(X_train,X_test, y_train,
                                     min_samples_factor=sample_factor)
        forest = fit_rf_svc(partitions)

        preds = predict_forest(partitions,forest)
        acc.append(metrics.accuracy_score(y_test,preds))

    cv_acc.append(np.mean(acc))

statistics.variance(cv_acc)
```

Listing B.8: Experiment on training time

```
time_forest=[]
time_dim=[]
time_classes=[]
for n in samples:

    n_train = 2/3*n

    for trees in n_trees:

        X, y = datasets.make_classification(n_samples=n)

        X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=1/3)

        t = time.time()

        partitions = extra_partition(X_train,X_test,y_train, n_estimators=trees)
        forest = fit_rf_svc(partitions)

        time_forest.append(time.time() - t)

    for dim in dim_list:

        X, y = datasets.make_classification(n_samples=n, n_features=dim)

        X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=1/3)

        t = time.time()

        partitions = extra_partition(X_train,X_test,y_train, n_estimators=1)
        forest = fit_rf_svc(partitions)

        time_dim.append(time.time() - t)

    for k in n_classes:

        X, y = datasets.make_classification(n_samples=n,
                                            n_classes=k,
                                            n_informative=k,
                                            n_features=2*k)

        X_train,X_test, y_train,y_test = train_test_split(X,y,test_size=1/3)

        t = time.time()

        partition = extra_partition(X_train,X_test, y_train, n_estimators=1)
        svc_tree = fit_rf_svc(partition)

        time_classes.append(time.time() - t)
```

Faculty of Sciences
Kasteelpark Arenberg 11 bus 2100
3001 LEUVEN, BELGIË
tel. + 32 16 32 14 01
fax + 32 16 32 14 01
www.kuleuven.be