MODELING ADDITIVE STRUCTURE AND DETECTING INTERACTIONS WITH GROVES OF TREES

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Daria Sorokina

August 2008

MODELING ADDITIVE STRUCTURE AND DETECTING INTERACTIONS WITH GROVES OF TREES

Daria Sorokina, Ph.D.

Cornell University 2008

Discovery of additive structure is an important step towards understanding a complex multi-dimensional function, because it allows for expressing this function as the sum of lower-dimensional or otherwise simpler components. Modeling additive structure also opens up opportunities for learning better regression models.

The term statistical interaction is used to describe the presence of non-additive effects among two or more variables in a function. When variables interact, their effects must be modeled and interpreted simultaneously. Thus, detecting statistical interactions can be critical for an understanding of processes by domain researchers.

This dissertation analyzes benefits of modelling additive structure for prediction and interaction detection problems. It describes a new learning algorithm called Groves, which is an ensemble of additive regression trees. Groves is based on such existing techniques as bagging and additive models; their combination allows us to use large trees in the ensemble and at the same time model additive structure of the response function. Regression version of the algorithm, Additive Groves, and its classification counterpart, Gradient Groves, yield consistently high performance across a variety of problems, outperforming on average a large number of other algorithms.

Additive nature of Groves makes it particularly useful for interaction detection. This dissertation introduces a new approach to interaction detection: it is based

on comparing the performance of restricted and unrestricted predictive models. Groves of trees allow variable interactions to be carefully controlled and therefore are especially useful for this framework.

The details of proposed practical approach to interaction detection analysis are demonstrated on real data describing the abundance of different species of birds in the prairies east of the southern Rocky Mountains.

BIOGRAPHICAL SKETCH

Daria Sorokina grew up in Moscow, Russia. She graduated from Moscow State 57th High School in 1998. She received Diplom with Honors in Applied Mathematics and Computer Science from Moscow State University in 2003. After that she worked for a year as a junior researcher at Russian Academy of Sciences. Daria spent four years in Cornell graduate school where she did research on statistical interactions and ornithology applications with Rich Caruana and Mirek Riedewald, briefly interrupted by internships in Fraunhofer IPSI and Google Pittsburgh. Daria defines her research interests as a blurry area between data mining and machine learning.

ACKNOWLEDGEMENTS

This work would not be possible without the help and influence of many people I met during my study in Cornell. First of all I would like to thank my advisor, Rich Caruana. His experience and support were invaluable for me. I have learned a lot from him both about machine learning and about how to do research. Many ideas that later formed this thesis emerged in discussions during our meetings.

Mirek Riedewald was my informal second advisor and I really appreciate the influence that he had on my work. He was always available for long discussions of new ideas (often spontaneous and unscheduled) and he was the first to proofread and improve texts of all my papers: his role in developing my technical writing skills is hard to overestimate.

I was also very lucky to have an exciting minor advisor — the role of Giles Hooker went far beyond choosing which classes I should take. He was very involved in my research and his expertize in statistics provided additional dimension to this work.

I would also like to thank Dexter Kozen for serving on my thesis committee and for helpful comments and suggestions.

The motivation and the application of my research came from Avian Knowledge Network project — collaboration with Cornell Laboratory of Ornithology. Steve Kelling and Wes Hochacka are the primary ornithology experts who provided us with data, problems, analysis and interpretations of our results. Daniel Fink introduced me the idea of interaction detection that later developed into my thesis topic. Roger Slothower helped to solve technical issues with the data.

I'd like to mention other students involved in AKN project. Art Munson was my co-author, collaborator, officemate and best friend for the last two years, during which we had many discussions on every possible topic on the project and beyond

it. Tom Finley and Mohamed Elhawary participated in the early stages of data analysis and helped a lot with tedious tasks of data cleaning. Ben Shaby and Dan Sheldon were involved in numerous discussions on our project meetings.

Nicolas Loeff came up with the initial idea for Gradient Groves when I gave a talk on Additive Groves in UIUC. Alex Niculescu-Mizil was of enormous help when I needed to incorporate my algorithms into his model comparison framework.

The AKN project was funded by grants ITR EF-0427914, SEI IIS-0612031, NSF-0412930, IIS-0748626. I was supported by a fellowship from Leon Levy foundation in 2007-2008.

Although my first project in Cornell, plagiarism detection in arXiv, was not related to my thesis topic, it was an interesting and successful project and I had a great experience working on it. I would like to thank all people I worked with at that time: Johannes Gehrke, Simeon Warner, Paul Ginsparg and Patrick Ng.

I also would like to extend special thanks to my Diplom research advisor at Moscow State University, Mikhail Petrovskiy, for introducing me into the world of data mining.

And a lot of thanks go to my family for their indefinite support during my time at graduate school: my parents Viktor and Galina and my husband Alex. Alex also gets credit for technical assistance at deadline times — many figures in this thesis look accurate and interpretable only due to his help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Additive Structure

Regression data sets often describe the continuous response functions with high precision and therefore this function can be quite complex. In most real world functions it is possible to distinguish between two different components of complexity: additivity and non-linear components. Many existing methods are good at modelling one of these components or the other, but not both of them at the same time. It turns out that both components are crucial and ignoring one of them prohibits learning methods to model a complex response function properly. Automatically detecting and modeling additive structure in otherwise non-linear models can help to achieve a very good performance on regression data sets.

1.2 Statistical Interactions

Although achieving best possible performance is critical for many problems in machine learning and data mining, it does not cover all possible goals of the data analysis. Models that act only as black boxes — make good predictions, but do not provide much insight into the decision making process — might be unsatisfactory for domain scientists who also want to answer questions like: Which features are important? What effects do they have on the response variable? Which features are involved in complex effects and must be studied only together with other features? How can we visualize and interpret such complex effects? Separate post-processing techniques are needed to answer these questions.

The term *statistical interaction* is used to describe the presence of non-additive effects among two or more variables in a function. Two variables are said to interact

in a function when the effect of one variable on the response depends on values of the other variable. Precisely, variables x_i and x_j interact in $F(\mathbf{x})$ when partial derivative $\frac{\partial F(\mathbf{X})}{\partial x_i}$ depends on x_j or, in a more general case, when the *"difference in the value of $F(\boldsymbol{x})$ for different values of x_i depends on the value of x_j"* (Friedman & Popescu [20]). This is equivalent to the following definition:

Function $F(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$, shows no interaction between variables x_i and x_j if it can be expressed as the sum of two functions, $f_{\setminus j}$ and $f_{\setminus i}$, where $f_{\setminus j}$ does not depend on x_j and $f_{\setminus i}$ does not depend on x_i:

$$F(\mathbf{x}) \quad = \quad f_{\setminus j}(x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n) + f_{\setminus i}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) \quad (1.1)$$

For example, $F(x_1, x_2, x_3) = sin(x_1 + x_2) + x_1 x_3$ has interactions between x_1 and x_2 and also between x_1 and x_3, but no interaction between x_2 and x_3.

Higher-order interactions between a larger number of variables are defined similarly. There is no K-way interaction between K variables in the function, if it can be represented as a sum of K (or fewer) functions, each of which does not depend on at least one variable in question. If such representation is not possible, we say that there is a K-way interaction. Function $x_1^{x_2+x_3}$ shows a 3-way interaction between x_1, x_2 and x_3, while $x_1x_2 + x_2x_3 + x_1x_3$ has all pairwise interactions, but not a 3-way interaction.

It is important to stress that the concept of statistical interaction is completely unrelated to the dependence and independence of variable distributions. Unfortunately, there is some ambiguity about the use of the term in the literature. Some authors use "interaction" to refer to different types of dependencies between variables, e.g., correlation [29]. In this work we discuss statistical (non-additive) interactions only.

1.3 Practical Importance

Many scientific inquiries seek to identify what variables are important and to describe their effects. Discovery of additive structure is an important step towards understanding a complex multi-dimensional function, because it allows for expressing this function as the sum of lower-dimensional components. When variables interact, their effects cannot be decomposed into independent lower-dimensional contributions and hence must be modeled simultaneously.

Interaction detection has significant practical importance because it provides valuable knowledge about a domain. Some processes can only be described accurately in terms of predictor interactions. For example, in drug design the synergistic effect of antagonistic drug reactions is an interactive effect. Or consider the problem of modeling bird migrations, the temporally coordinated movements of individual birds across a landscape. Migrations are necessarily described in terms of the simultaneous effects of both temporal and spatial predictors, a two-way interaction. In ecology, interactions between multiple environmental stressors can have serious impacts on species health. For example, it is known that decreasing habitat patch size and acidification due to acid rain, each independently have adverse affects on many bird species. Yet Hames et al. [24] found that the combined effects of these two stressors on North American Wood Thrush populations were significantly more severe than would be expected if each acted alone, so these two factors must interact.

1.4 Objectives

The goals of this research are:

1. To develop learning methods capable of modeling complex functions that

contain both additive structure and highly non-linear components

2. To design model postprocessing techniques for direct analysis of inherent additive structure in the data, i.e. detecting and measuring presence and extent of statistical interactions

3. To perform detailed interaction detection analysis on large-scale noisy data sets, identify potential problems that complicate information extraction from such data and develop solutions that are applicable for real world tasks

CHAPTER 2

RELATED WORK

The focus of this thesis is centered around an algorithm based on combining additive models with an ensemble of regression trees and its application to the problem of interaction detection. Accordingly, this chapter reviews three related research areas: ensembles, additive models and approaches to detection of statistical interactions.

2.1 Ensembles

2.1.1 Non-Additive Ensembles

Ensembles are sets of models combined together to produce better results then a single model would be able to do. First we are going to review ensembles that do not try to capture additive structure of the response; they achieve improvement in performance by decreasing variance, gradually expanding model or combining models of different types.

Bagging

Bagging is a well-known procedure for improving model performance by reducing variance [6]. On each iteration of bagging, we draw a bootstrap sample from the training set, and train the full model from that sample. After repeating this procedure a number of times, we end up with an ensemble of models. The final prediction of the ensemble on each test data point is an average of the predictions of all models. In recent work it has been shown that bagged trees are competitive with the best available learning methods [13, 37]

Wagging

Wagging[5] is a variant of bagging, where differences in the training sets between models are achieved not by drawing bootstrap samples, but by creating disturbance in the response: small amount of gaussian noise is added to the data, different values of noise create similar, but slightly different train sets. After that the models are averaged the same way as in bagging.

Bayesian Averaging

Domingos [17] tried to improve bagging by replacing simple average with weighted average, where weights were calculated as posterior probabilities of models given the data. However, this method turned out to be very prone to overfitting and therefore is interesting only from the theoretical point of view.

Random Forest

Random Forests [7] is another ensemble of trees that improves performance of single models by creating many similar models and then averaging their predictions. It randomizes its trees by limiting the number of attributes allowed to use in every node. Each time when the tree-building algorithm wants to split the node, a random set of input variables is chosen and the algorithm is able to choose only one of those variables. The set is defined randomly and is different for different nodes. It is worth noting, that although bagging and Random Forest end up with a sum of models (trees) in the end, they cannot be considered additive models. Their sums come from averaging similar models, each of which is trying to model original data. When we talk about additive structure, we usually mean that different models in the sum can differ from each other significantly, because they model different additive components of the response.

Stacking

Stacking [49] is a technique to combine ready trained model into an ensemble by a separate machine learning algorithm. Zenko et. al. use a meta decision tree [52] as such learning algorithm: instead of providing ready prediction, leaves of the tree show which base algorithm should be used on the current test point.

Now we are going to look at "additive" ensembles, those that model additive structure of the response function as a sum of their components.

2.1.2 Additive Ensembles

Gradient Boosting

Gradient boosting [28] is a general framework for several ensemble algorithms. On the theoretical level the algorithms differ from each other by the loss function that they are optimizing. The training process of each new model in the ensemble depends on predictions of the models that are already in the ensemble: the joint predictions of the ensemble are fit to the training set with respect to a given loss. Here adding a new model to the ensemble corresponds to taking a gradient descent step on the loss opitimization surface, hence the name of the algorithm.

As the ensemble keeps fitting the train set better and better, it inevitably reaches the point where overfitting begins and the performance becomes worse. A regularization parameter, a coefficient $0 < \eta < 1$, controls how long it takes before the model begins to overfit. An improved version of the algorithm, stochastic gradient boosting [18], uses subsampling from the training set to minimize overfitting: each new model is trained on a random subset of the training set.

We are especially interested in the case when loss is calculated as squared error, because in this case the ensemble becomes additive and each new model

is trained on the residuals of all models currently present in the ensemble. In this case gradient boosting acts as an infinite classical additive mode: new models always get added into the ensembles, and old models stay there forever: none gets discarded and replaced. An important fact to note is that such "infinite additive model" can work only when models are simple (e.g., small trees), because complex models have more flexibility, can better and earlier fit the training set, and as a result, overfit too fast.

Gradient boosting is a state of the art ensemble tree method for regression. Chipman et al [14] recently performed an extensive comparison of several algorithms on 42 data sets. In their experiments gradient boosting showed performance similar to or better than Random Forests and a number of other types of models.

Bayesian Ensemble Learning

In Bayesian Ensemble Learning [14] a sum of trees is trained by a backfitting algorithm driven by statistical model.

2.2 Additive Models

2.2.1 Classical Additive Models

A classical additive model is a sum of simpler models, where each simple model depends only on one input variable. The prediction of an additive model is computed as the sum of the predictions of these simpler models:

$$F(\mathbf{x}) = F_1(x_1) + F_2(x_2) + \cdots + F_N(x_N). \tag{2.1}$$

Here each $F_i(x_i)$, $1 \leq i \leq N$, is the prediction made by the i-th model. N is a parameter, the number of simple models.

Algorithm 1 Training additive models by backfitting

function Classical(N,{$\mathbf{x}$,y})
  for $i = 1$ **to** N **do**
    $Model_i^{(N)} = 0$
  end for
  Converge(N,{$\mathbf{x}$,y}, $Model_1^{(N)}, \ldots, Model_N^{(N)}$)
end function

function Converge(N,{$\mathbf{x}$,y},$Model_1^{(N)}, \ldots, Model_N^{(N)}$)
  repeat
    for $i = 1$ **to** N **do**
      newTrainSet = $\{\mathbf{x}, y - \sum_{k \neq i} Model_k^{(N)}(\mathbf{x})\}$
      $Model_i^{(N)}$ = TrainModel(newTrainSet)
    end for
  until (change from the last iteration is small)
end function

In statistics, the basic mechanism for training an additive model with a fixed number of components is the *backfitting algorithm* [25]. We will refer to this as the Classical algorithm for training an additive model.

After initializing all components (simple models), the algorithm cycles through them until the whole additive model converges to some stable state. The first component is trained on the original data set, i.e., a set of training points $\{(\mathbf{x}, y)\}$. Let $\hat{F}_1$ denote the function encoded by this model. Then we train the second component, which encodes $\hat{F}_2$, on the residuals, i.e., on the set $\{(\mathbf{x}, y - \hat{F}_1(\mathbf{x}))\}$. The third component then is trained on the residuals of the first two, i.e., on $\{(\mathbf{x}, y - \hat{F}_1(\mathbf{x}) - \hat{F}_2(\mathbf{x}))\}$, and so on.

After we have trained N models this way, we *discard* the first component and retrain it on the residuals of the other $N - 1$ components, i.e. on the set $\{(\mathbf{x}, y - \hat{F}_2(\mathbf{x}) - \hat{F}_3(\mathbf{x}) - \cdots - \hat{F}_N(\mathbf{x}))\}$. Then we similarly discard and retrain the second model, and so on. We keep cycling through the components in this way until there is no significant improvement in the RMSE on the training set.

Note that the algorithm does not use the fact that every component takes only

one input variable. We can still do the same type of backfitting for more complex additive models, where each simple model can use all input variables.

$$F(\mathbf{x}) = F_1(\mathbf{x}) + F_2(\mathbf{x}) + \cdots + F_N(\mathbf{x}) \tag{2.2}$$

2.2.2 Generalized Additive Models

Generalized additive models, introduced in 1990 by Hastie and Tibshirani [46], allow to model response functions that depend on additive combination of single effects of input variables when this dependency is represented by a non-linear function.

$$G(F(\mathbf{x})) = F_1(\mathbf{x}) + F_2(\mathbf{x}) + \cdots + F_N(\mathbf{x}) \tag{2.3}$$

A useful type of generalized additive models is logistic regression: it allows using additive models techniques on binary classification data.

$$ln(\frac{F(\mathbf{x})}{1 - F(\mathbf{x})}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_N x_N \tag{2.4}$$

Friedman [19] and Collins et. al. [16] demonstrate how logistic regression representation in form of generalized additive model helps to show that logistic regression is actually related to boosting [45]

2.3 Interaction Detection Methods

2.3.1 Early Methods

Early methods for interaction detection were parametric and required explicit modeling of interactions, most often as multiplicative terms. Only limited types of interactions can be detected this way.

Interaction detection is regularly performed as part of many statistical analyses [15]. Most often parametric models are used where the analyst specifies the interaction as a parametric term, or perhaps several terms. In this setting interaction detection is simply a parameter estimation problem. More recently, techniques have been developed to detect interactions within semi-parametric models [22, 50, 44]. Practically, these techniques work best for detection of low-order (currently 2 or 3 dimensional) interactions between specified groups of predictors.

More general approaches to interaction detection were recently introduced [20][27]. These methods are based on building a model and detecting interactions in the function learned by the model.

2.3.2 Partial Dependence Functions

Friedman and Popescu [20] developed tests for interaction detection for a very general class of prediction models, including fully nonparametric models. Their method makes use of the fact that in the absence of an interaction between x_i and x_j the following holds: $\frac{\partial F(\mathbf{X})^2}{\partial x_i \partial x_j} = \frac{\partial F(\mathbf{X})}{\partial x_i} + \frac{\partial F(\mathbf{X})}{\partial x_j}$. They estimate the partial dependence functions[19] of the model and then estimate the strength of an interaction as the difference between the right hand side and the left hand side of the equation above, scaled by variance in the response.

The drawback of that method is that in order to get accurate estimates of the partial dependence function, it relies on predictions for synthetic data points in sparse regions of the input space. As a result, decisions about presence of interactions can be made because of spurious interactions that happen only in those regions[27]. To demonstrate this effect, we generated two simple data sets for the function $F(\mathbf{x}) = x_1^3 + x_2^3$. In the first data set both x_1 and x_2 are distributed uniformly between -10 and 10. For the second data set we took the same points

and removed those where both x_1 and x_2 were positive. Neither of the data sets contains interactions, but the estimates produced by Friedman's approach using RuleFit[21] were 0.0243 for the first and 0.0824 for the second set. The presence of an unpopulated region in the input data increased the estimated strength of the presumed interaction by a factor of three.

In order to deal with this extrapolation problem, Friedman and Popescu [20] suggest comparing the estimated interaction strength produced by the method described above with a similar estimate on the same data, but for a different response function that does not contain any interactions. They refer to the distribution of the latter estimate as a null distribution. Points from the null distribution are generated by means of a bootstrapping technique. Then the hypothesis that the original estimate comes from the null distribution is tested. However, our experiments with RuleFit revealed several examples of unsatisfactory performance of this technique. For instance, we generated 5 data sets with response function $x_1^2 + x_2^2$ without noise and for each of them generated 50 samples from the null distribution. For 3 of those data sets RuleFit produced results that indicated presence of an interaction, i.e., the original estimate was further from the mean of the null distribution than 3 standard deviations.

A similar technique based on marginal integration, was also developed by Linton [33] for kernel methods.

2.3.3 ANOVA Decomposition

Hooker [27, 26] suggests another approach, based on estimating orthogonal components of the ANOVA decomposition. This method has higher computational complexity because it requires generating a full grid of data points with all possible combinations of values for those input variables that are tested for interaction.

Similar to the Friedman and Popescu approach, this method can suffer from extrapolations over unpopulated regions of the input space. To overcome this problem, as well as problems caused by correlations, Hooker suggests imposing low weights for points from low-density regions. Unfortunately, this requires the use of density estimation techniques and further increases complexity of the method.

CHAPTER 3

ADDITIVE GROVES

This chapter presents a new regression algorithm called Additive Groves, an ensemble of additive regression trees. We initialize a Grove with a single small tree. The Grove is then gradually expanded: on every iteration either a new tree is added, or the trees that already are in the Grove are made larger. This process is designed to try to find the simplest model (a Grove with the fewest number of small trees) that captures the underlying additive structure of the target function. As training progesses, this algorithm yields a sequence of Groves of slowly increasing complexity. Eventually, the largest Groves may begin to overfit the training set even as they continue to learn important additive structure. This overfitting is reduced by applying bagging on top of the Grove learning process.

3.1 Algorithm

Bagged Groves of Trees, or bagged Groves for short, is an ensemble of regression trees. Specifically, it is a bagged additive model of regression trees where each individual additive model is trained in an adaptive way by gradually increasing both number of trees and their complexity.

3.1.1 Regression Trees

The unit model in a Grove is a regression tree. Algorithms for training regression trees differ in two major aspects:

1. the criterion for choosing the best split in a node and
2. the way in which tree complexity is controlled.

We use trees that optimize RMSE (root mean squared error) and we control tree complexity (size) by imposing a limit on the size (number of cases) at an internal node. If the fraction of the data points that reach a node is less than a specified threshold α, then the node is declared a leaf and is not split further. Hence the smaller α, $0 \leq \alpha \leq 1$, the larger the tree. (See Figure 3.2.)

Note that because we will later bag the tree models, the specific choice of regression tree is not particularly important. The main requirement is that the complexity of the tree should be controllable.

3.1.2 Additive Models — Classical Algorithm

A Grove of trees is an additive model where each additive term is represented by a regression tree. The prediction of a Grove is computed as the sum of the predictions of these trees: $F(\mathbf{x}) = T_1(\mathbf{x}) + T_2(\mathbf{x}) + \cdots + T_N(\mathbf{x})$. Here each $T_i(\mathbf{x})$, $1 \leq i \leq N$, is the prediction made by the i-th tree in the Grove. The Grove model has two main parameters: N, the number of trees in the Grove, and α, which controls the size of each individual tree. We use the same value of α for all trees in a Grove.

Algorithm 2 shows training of a single Grove by backfitting algorithm described in Section 2.2.1.

Bagging

As with single decision trees, a single Grove tends to overfit to the training set when the trees are large. Such models show a large variance with respect to specific subsamples of the training data and benefit significantly from bagging, a well-known procedure for improving model performance by reducing variance[6]. On each iteration of bagging, we draw a bootstrap sample (bag) from the training

Algorithm 2 Classical additive model training

function Classical(α,N,{$\mathbf{x}$,y})
  for $i = 1$ **to** N **do**
    $\text{Tree}_i^{(\alpha,N)} = 0$
  end for
  Converge(α,N,{$\mathbf{x}$,y}, $\text{Tree}_1^{(\alpha,N)}, \ldots, \text{Tree}_N^{(\alpha,N)}$)
end function

function Converge(α,N,{$\mathbf{x}$,y},$\text{Tree}_1^{(\alpha,N)}, \ldots, \text{Tree}_N^{(\alpha,N)}$)
  repeat
    for $i = 1$ **to** N **do**
      $\text{newTrainSet} = \{\mathbf{x}, y - \sum_{k \neq i} \text{Tree}_k^{(\alpha,N)}(\mathbf{x})\}$
      $\text{Tree}_i^{(\alpha,N)} = \text{TrainTree}(\alpha, \text{newTrainSet})$
    end for
  until (change from the last iteration is small)
end function

set, and train the full model (in our case a Grove of additive trees) from that sample. After repeating this procedure a number of times, we end up with an ensemble of models. The final prediction of the ensemble on each test data point is an average of the predictions of all models.

Example

In this section we illustrate the effects of different methods of training bagged Groves on synthetic data. The synthetic data set was generated by a function of 10 variables that was previously used by Hooker[26].

$$F(x) = \pi^{x_1 x_2}\sqrt{2x_3} - sin^{-1}(x_4) + log(x_3 + x_5) - \frac{x_9}{x_{10}}\sqrt{\frac{x_7}{x_8}} - x_2 x_7 \tag{3.1}$$

Variables x_1, x_2, x_3, x_6, x_7, x_9 are uniformly distributed between 0.0 and 1.0 and variables x_4, x_5, x_8 and x_{10} are uniformly distributed between 0.6 and 1.0. Ranges are selected to avoid extremely large or small function values.

Figure 3.1 shows a contour plot of how model performance depends on both α, the size of tree, and N, the number of trees in a Grove, for 100 bagged Groves

trained with the classical method on 1000 training points from the above data set. The performance is measured as RMSE on an independent test set consisting of 25,000 points. Notice that lower RMSE implies better performance. The bottom-most horizontal line for $N = 1$ corresponds to bagging single trees. The plot clearly indicates that by introducing additive model structure, with $N > 1$, performance improves significantly. We can also see that the best performance is achieved by Groves containing 5-10 relatively small trees (large α), while for larger trees performance deteriorates.

3.1.3 Layered Training

When individual trees in a Grove are large and complex, the Classical additive model training algorithm (Section 3.1.2) can overfit even if bagging is applied. The first several trees might already perfectly model the training data, hence the remaining trees in the Grove are superfluous. (All residuals at this point are zero.) Consider the extreme case $\alpha = 0$, i.e., a Grove of full trees. The first tree will perfectly model the training data, leaving residuals with value 0 for the other trees in the Grove. Hence the intended Grove of several large trees will degenerate to a single tree.

One could address this issue by limiting trees to very small size. However, we still would like to be able to use large trees in a Grove so that we can capture complex and non-linear functions. To prevent the degeneration of the Grove as the trees become larger, we developed a "layered" training approach. In the first round we grow N small trees. Then in later cycles of discarding and re-training the trees in the Grove we gradually increase tree size.

More precisely, no matter what the value of α, we *always* start the training process with small trees, typically using a start value $\alpha_0 = 0.5$. Let α_j denote the

Algorithm 3 Layered training

function Layered(α,N,train)
  $\alpha_0 = 0.5, \alpha_1 = 0.2, \alpha_2 = 0.1, \ldots, \alpha_{\max} = \alpha$
  for $j = 0$ **to** max **do**
    if $j = 0$ **then**
      for $i = 1$ **to** N **do**
        $\text{Tree}_i^{(\alpha_0, N)} = 0$
      end for
    else
      for $i = 1$ **to** N **do**
        $\text{Tree}_i^{(\alpha_j, N)} = \text{Tree}_i^{(\alpha_{j-1}, N)}$
      end for
    end if
    Converge(α_j,N,train,$\text{Tree}_1^{(\alpha_j, N)}, \ldots, \text{Tree}_N^{(\alpha_j, N)}$)
  end for
end function

value of the size parameter after j iterations of the Layered algorithm (Algorithm 3). After reaching convergence for α_{j-1}, we increase tree complexity by setting α_j to approximately half the value of α_{j-1}. We continue to cycle through the trees, re-training all trees in the Grove in the usual way, but now allow them to reach the size correspondent to the new larger α_j, and as before, we proceed until the Grove converges on this layer. We keep gradually increasing tree size until $\alpha_j \approx \alpha$.

For a training set with 1000 data points and $\alpha = 0$, we use the following sequence of values of α_j: $(0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001)$. It is worth noting that while training a Grove of large trees, we automatically obtain all Groves with the same N for all smaller tree sizes in the sequence. Figure 3.2 shows how 100 bagged Groves trained by the layered approach perform on the synthetic data set. Overall performance is much better than for the classical algorithm and bagged Groves of N large trees now perform at least as well as bagged Groves of N smaller trees.

3.1.4 Dynamic Programming Training

Layered training helps prevent overfitting when the trees are large, but the algorithm explores the (α, N) parameter space in a fairly restricted manner. To construct an (α, N) Grove, it goes through a series of Groves with the same N. Intuitively, considering a grid of (α, N) parameter values like in Figure 3.2, the Layered algorithm proceeds left-to-right along the horizontal grid lines, generating the Grove model for a grid point from its left neighbor.

There is no reason to believe that the best (α, N) Grove should always be constructed from a $(\approx 2\alpha, N)$ Grove. In fact, a large number of small trees might overfit the training data and hence limit the benefit of increasing tree size in later iterations. To avoid this problem, we need to give the Grove training algorithm additional flexibility in choosing the right balance between increasing tree size and the number of trees. This is the motivation behind the *Dynamic Programming* Grove training algorithm.

This algorithm can choose to construct a new Grove from an existing one by either adding a new tree (while keeping tree size constant) or by increasing tree size (while keeping the number of trees constant). Considering the parameter grid, the Grove for a grid point (α_j, n) could be constructed either from its left neighbor (α_{j-1}, n) or from its lower neighbor $(\alpha_j, n-1)$. Pseudo-code for this approach is shown in Algorithm 4. We make a choice between the two options for computing each Grove (adding another tree or making the trees larger) in a greedy manner, i.e., the one that results in better performance of the Grove on the validation set. Clearly, for the algorithm to construct an (α, N) Grove this way, it has to generate the whole "grid" of Groves with smaller trees (i.e., larger α) and less than N trees. As the final ensemble will be *Bagged* Groves, each Grove is trained on a bootstrap sample of the data and hence there will be some data points from the training set

that were not used in the current bag. We use the out-of-bag data points[9] as the validation set for choosing which of the two Groves to use at each step.

Figure 3.3 shows how the Dynamic Programming approach improves bagged Groves over the layered training. Figure 3.4 shows the choices that are made during the process: it plots the average difference between RMSE of the Grove created from the lower neighbor (increase n) and performance of the Grove created from the left neighbor (decrease α_j). That is, a negative value means that the former is preferred, while a positive value means that the latter is preferred at that grid point. We can see that for this data set increasing the tree size is the preferred direction, except for cases with many small trees.

This dynamic programming version of the algorithm does not explore all possible sequences of steps to build a Grove of trees, because we require that every grove built in the process should contain trees of equal size. We have tested several other possible approaches that don't have this restriction, but they failed to produce any improvements and were noticeably worse from the running time point of view. For these reasons we prefer the dynamic programming version over other, less restricted options.

3.1.5 Randomized Dynamic Programming Training

Our bagged Grove training algorithms so far performed bagging in the usual way, i.e., create a bag of data, train all Groves for different vallues of (α, N) on that bag, then create the next bag, generate all models on this bag; and so on for 100 different bags. When the Dynamic Programming algorithm generates a Grove using the same bag, i.e., the same train set that was used to generate its left and lower neighbors, complex models might not be very different from their neighbors because those neighbors already might have overfitted and there is not enough training data

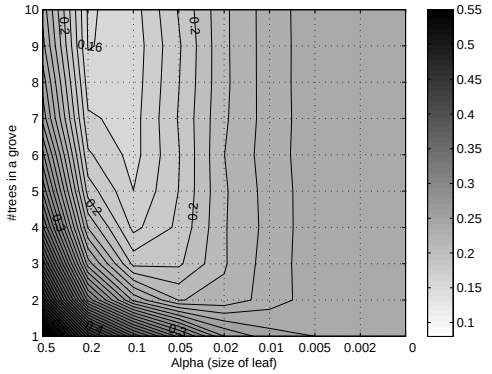

Figure 3.1: RMSE of bagged Grove, Classical algorithm

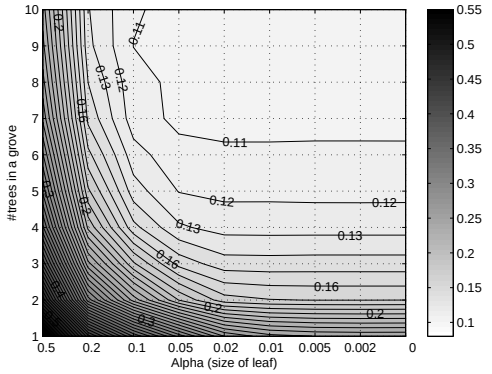

Figure 3.2: RMSE of bagged Grove, Layered algorithm

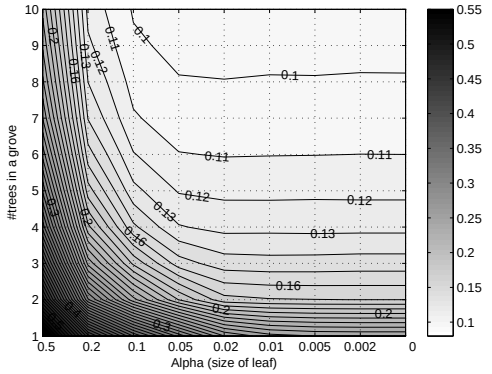

Figure 3.3: RMSE of bagged Grove, Dynamic Programming algorithm

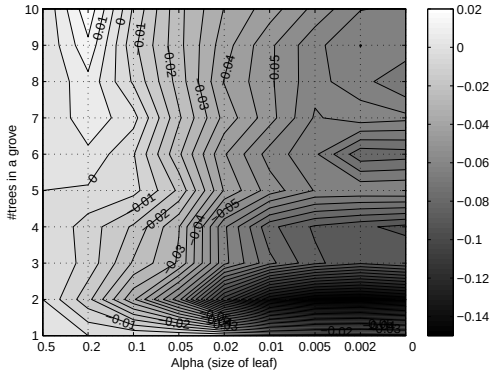

Figure 3.4: Difference in performance between "horizontal" and "vertical" steps

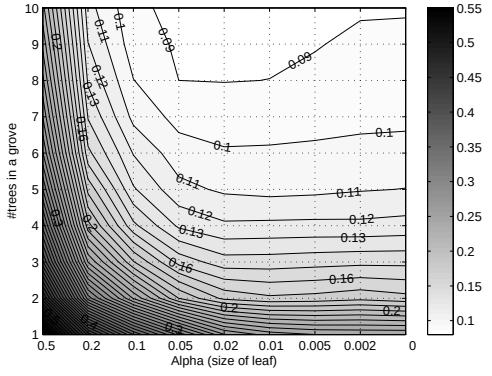

Figure 3.5: RMSE of bagged Grove (100 bags), Randomized Dynamic Programming algorithm

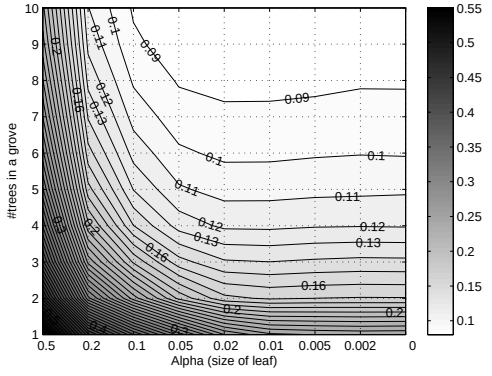

Figure 3.6: RMSE of bagged Grove (500 bags), Randomized Dynamic Programming algorithm

Algorithm 4 Dynamic Programming Training

function DP(α,N,trainSet)
  $\alpha_0 = 0.5, \alpha_1 = 0.2, \alpha_2 = 0.1, \ldots, \alpha_{\max} = \alpha$
  for $j = 0$ **to** max **do**
    for $n = 1$ **to** N **do**

      for $i = 1$ **to** $n - 1$ **do**
        $\text{Tree}_{\text{attempt1},i} = \text{Tree}_i^{(\alpha_j, n-1)}$
      end for
      $\text{Tree}_{\text{attempt1},n} = 0$
      Converge(α_j,n,train,$\text{Tree}_{\text{attempt1},1}, \ldots, \text{Tree}_{\text{attempt1},n}$)

      if $j > 0$ **then**
        for $i = 1$ **to** n **do**
          $\text{Tree}_{\text{attempt2},i} = \text{Tree}_i^{(\alpha_{j-1}, n)}$
        end for
        Converge(α_j,n,train,$\text{Tree}_{\text{attempt2},1}, \ldots, \text{Tree}_{\text{attempt2},n}$)
      end if

      winner = Compare $\sum_i \text{Tree}_{\text{attempt1},i}$ and $\sum_i \text{Tree}_{\text{attempt2},i}$ on validation set
      for $i = 1$ **to** n **do**
        $\text{Tree}_i^{(\alpha_j, n)} = \text{Tree}_{\text{winner},i}$
      end for
    end for
  end for
end function

to learn anything new. We can address this problem by using a different bag of data on every step of the Dynamic Programming algorithm, so that every Grove has some new data to learn from. While performance of a single Grove might become worse, performance of bagged Groves improves due to increased variability in the models. Figure 3.5 shows the improved performance of this final version of our Grove training approach. The most complex Groves are now performing worse than their left neighbors with smaller trees. This happens because those models need more bagging steps to converge to their best quality. Figure 3.6 shows the same plot for bagging with 500 iterations where the property "more complex models are at least as good as their less complex counterparts" is restored.

Table 3.1: Performance of bagged Groves (Randomized Dynamic Programming training) compared to boosting and bagging. RMSE on the test set averaged over 10 runs.

	California Housing	Elevators	Kinematics	Computer Activity	Stock	Synthetic No Noise	Synthetic Noise
Bagged Groves							
RMSE	0.38	0.309	0.364	0.117	0.097	0.087	0.483
StdDev	0.015	0.028	0.013	0.0093	0.029	0.0065	0.012
Boosting							
RMSE	0.403	0.327	0.457	0.121	0.118	0.148	0.495
StdDev	0.014	0.035	0.012	0.01	0.05	0.0072	0.01
Bagged trees							
RMSE	0.422	0.44	0.533	0.136	0.123	0.276	0.514
StdDev	0.013	0.066	0.016	0.012	0.064	0.0059	0.011

3.2 Experiments

We evaluated Additive Groves on 2 synthetic and 5 real-world data sets and compared the performance to two other regression tree ensemble methods that are known to perform well: stochastic gradient boosting and bagged regression trees. Bagged Groves consistently outperform both of them. For real data sets we performed 10 fold cross validation: for each run 8 folds were used as a training set, 1 fold as a validation set for choosing the best set of parameters and the last fold was used as the test set for measuring performance. For the two synthetic data sets we generated 30 blocks of data containing 1000 points each and performed 10 runs using different blocks for training, validation and test sets. We report mean and standard deviation of the RMSE on the test set. Table 3.1 shows the results; for comparability across data sets all numbers are scaled by the standard deviation of the response in the dataset itself.

3.2.1 Parameter Settings

Groves

We trained 100 bagged Groves using the Randomized Dynamic Programming technique for all combinations of parameters N and α with $1 \leq N \leq 15$ and $\alpha \in \{0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005\}$. Notice that with these settings the resulting ensemble can consist of at most 1500 trees. From these models we selected the one that gave the best results on the validation set. The performance of the selected Grove on the test set is reported.

Stochastic Gradient Boosting

The obvious main competitor to bagged Groves is gradient boosting [28][18], a different ensemble of trees also based on additive models. There are two major differences between boosting and Groves. First, boosting never discards trees, i.e., every generated tree stays in the model. Grove iteratively retrains its trees. Second, all trees in a boosting ensemble are always built to a fixed size, while groves of large trees are trained first using groves of smaller trees. We believe that these differences allow Groves to better capture the natural additive structure of the response function.

The general gradient boosting framework supports optimizing for a variety of loss functions. We selected squared-error loss because this is the loss function that our current version of the Groves algorithm optimizes for. However, like gradient boosting, Groves can be modified to optimize for other loss functions.

Friedman[18] recommends boosting small trees with at most 4–10 leaf nodes for best results. However, we discovered for one of our datasets that using larger trees with gradient boosting did significantly better. This is not surprising since some real datasets contain complex interactions, which cannot be accurately mod-

eled by small trees. For fairness we therefore also include larger boosted trees in the comparison than Friedman suggested. More precisely, we tried all $\alpha \in \{1, 0.5, 0.2, 0.1, 0.05\}$. Table 3.2 shows the typical correspondence between α and number of leaf nodes in a tree, which was very similar across the data sets. Preliminary results did not show any improvement for tree size beyond $\alpha = 0.05$.

Table 3.2: Typical number of leaf nodes for different values of α

α	# leaf nodes
1	2 (stump)
0.5	3
0.2	8
0.1	17
0.05	38
0.02	100
0.01	225
0.005	500
0	full tree

Stochastic gradient boosting deals with overfitting by means of two techniques: regularization and subsampling. Both techniques depend on user-set parameters. Based on recommendations in the literature and on our own evaluation we used the following values for the final evaluation: 0.1 and 0.05 for the regularization coefficient and 0.4, 0.6, and 0.8 as the fraction of the subsampling set size from the whole training set. Boosting can also overfit if it is run for too many iterations. We tried up to 1500 iterations to make the maximum number of trees in the ensemble equal for all methods in comparison. The actual number of iterations that performs best was determined based on the validation set, and therefore can be lower than 1500 for the best boosted model.

In summary, to evaluate stochastic gradient boosting, we tried all combinations of the values described above for the 4 parameters: size of trees, number of iterations, regularization coefficient, and subsampling size. As for Groves, we determine the best combination of values for these parameters based on a separate validation set.

Bagging

Bagging single trees is known to provide good performance by significantly decreasing variance of the individual tree models. However, compared with Groves and boosting, which are both based on additive models, bagged trees do not explicitly model the additive structure of the response function. Increasing the number of iterations in bagging does not result in overfitting and bagging of larger trees usually produces better models than bagging smaller trees. Hence we omitted parameter tuning for bagging. Instead we simply report results for a model consisting of 1500 bagged full trees. This number of iterations was more than enough to achieve the best performance for bagging.

3.2.2 Datasets

Synthetic Data without Noise

This is the same data set that we used as a running example in the earlier sections. The response function is generated by Equation 3.1. The function contains 6 additive components, some of them non-linear. The performance of bagged Groves on this dataset is much better than the performance of other methods.

Synthetic Data with Noise

This is the same synthetic dataset, only this time Gaussian noise is added to the response function. The standard deviation σ of the noise distribution is chosen as 1/2 of the standard deviation of the response in the original data set. Bagged Groves still perform clearly better, but the difference is smaller.

We have used 5 regression data sets from the collection of Luís Torgo [47] for the next set of experiments.

Kinematics

The Kinematics family of datasets originates from the Delve repository [42] and describes a simulation of robot arm movement. We used a *kin8nm* version of the dataset: 8192 cases, 8 continuous attributes, high level of non-linearity, low level of noise. Groves show 20% improvement over gradient boosting on this dataset. It is worth noticing that boosting preferred large trees on this dataset; trees with $\alpha = 0.05$ showed clear advantage over smaller trees. However, there was no further improvement for boosting even larger trees. We attribute these effects to high non-linearity of the data.

Computer Activity

Another dataset from the Delve repository, describes the state of multiuser computer systems. 8192 cases, 22 continuous attributes. The variance of performance for all algorithms is low. Groves show (3%) improvement compared to boosting.

California Housing

This is a dataset from the StatLib repository[35] and it describes housing prices in California from the 1990 Census: $20,640$ observations, 9 continuous attributes. Groves show 6% improvement compared to boosting.

Stock

This is a relatively small (960 data points) regression dataset from the StatLib repository. It describes daily stock prices for 10 aerospace companies: the task is to predict the first one from the other 9. Prediction quality from all methods is very high, so we can assume that the level of noise is small. This is another case when Groves give significant improvement (18%) over gradient boosting.

Elevators

This data set is obtained from the task of controlling an aircraft [10]. It seems to be noisy, because the variance of performance is high although the data set is rather large: $16,559$ cases with 18 continuous attributes. Here we see a 6% improvement.

3.2.3 Discussion

Based on the empirical results we conjecture that Bagged Groves outperform the other algorithms most when the datasets are highly non-linear and not very noisy. (Noise can obscure some of the non-linearity in the response function, making the best models that can be learned from the data more linear than they would have been for models trained on the response without noise.) This can be explained as follows. Groves can capture additive structure yet at the same time use large trees. Large trees capture non-linearity and complex interactions well, and this gives Groves an advantage over gradient boosting which relies mostly on additivity. Gradient boosting usually works best with small trees, and fails to make effective use of large trees. At the same time most data sets, even non-linear ones, still have significant additive structure. The ability to detect and model this additivity gives Groves an advantage over bagging, which is effective with large trees, but does not explicitly model additive structure.

Gradient boosting is a state of the art ensemble tree method for regression. Chipman et al[14] recently performed an extensive comparison of several algorithms on 42 data sets. In their experiments gradient boosting showed performance similar to or better than Random Forests and a number of other types of models. Our algorithm shows performance consistently better than gradient boosting and for this reason we do not expect that Random Forests or other methods that are not superior to gradient boosting would outperform our bagged Groves.

In terms of computational cost, bagged Groves and boosting are comparable. In both cases a large number of tree models has to be trained (more for Groves) and there is a variety of parameter combinations that need to be examined (more for boosting).

From the point of view of computational cost, a single ensemble of Groves will take significantly longer to train than boosting. The reason is that Groves need to generate many temporary trees that will not end up in the ensemble, while boosting generates each tree in the ensemble only once. On the other hand boosting has a larger parameter space to explore, and therefore we need to run it many times to find the right combination of parameters. Overall it depends on the specific dataset which of the two algorithms will be computationally cheaper. In our evaluation setting boosting and groves had comparable computational cost.

CHAPTER 4

GRADIENT GROVES

Gradient Groves is a classification version of Additive Groves. It adapts ideas from Gradient boosting framework to extend the original regression algorithm to any given loss, in particular logistic regression log-likelihood loss.

Both gradient boosting and Additive Groves are powerful ensemble methods that are based on additive models. However, they each have different strengths and weaknesses. Gradient boosting is built on a theoretically well-developed framework that allows for optimization for different losses and different problems, including classification. In contrast, the original Additive Groves are limited to squared loss and regression. However, Additive Groves can train ensembles of large trees without overfitting and because of this often outperform gradient boosting on highly non-linear regression data sets that cannot be fit adequately with small trees.

In this chapter we show how the approach of gradient descent in function space can be transferred from gradient boosting's infinite stagewise forward training to Groves' bagged additive models trained by backfitting. Intuitively, we take the internal procedure of building each new tree on pseudo-residuals of gradient descent steps from gradient boosting and combine it with the external loop that controls the size and numer of trees from Additive Groves. This way we are able to combine the strengths of both methods and produce a powerful ensemble of large trees that optimizes to a given loss function. In this paper our goal is to develop a new ensemble method for binary classification, therefore we concentrate on combining Groves with L_2_TreeBoost. Because L_2_TreeBoost implicitly assumes small trees that will never exactly predict class $+1$ or class -1, there was difficulty extending it to deal with large trees. We developed a number of improvements

for this specific type of Gradient Groves. We will often refer to Gradient Groves optimizing binomial log-likelihood as just Gradient Groves.

Boosting algorithms often benefit from calibration: postprocessing that corrects predictions of binary classification models towards actual probabilities[36]. We analyze how calibration affects Gradient Groves and conclude that with the right choice of parameters, Gradient Groves are well calibrated already out-of-the-box. If the parameter choice is suboptimal, however, calibration will improve the probabilities.

We compare our new algorithm with other methods that were studied in a recent empirical comparison[13] of classification algorithms. Our results show that on average Gradient Groves outperforms all other learning methods from that study.

4.1 Background: Gradient Boosting

A short overview of Gradient Boosting was already given in Section 2.1.2; here we describe this algorithm in more details, because some of its ideas will be reused for Gradient Groves.

Gradient boosting was introduced in [28] as a general ensemble framework. Here we describe the version where the models used in the ensemble are small regression trees. Gradient boosting is a stagewise strictly-forward algorithm. Models that are already in the ensemble stay there, are not changed later, and therefore do not depend on models that will be built later. The goal is to minimize $\sum_i L(y_i, F(\mathbf{x}_i))$, where $\{\mathbf{x}, y\}$ is the training set, $L(y, F)$ is a loss function and $F(\mathbf{x})$ is our approximation of the response. On every iteration we represent F as the sum of all trees built so far:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + Tree_m(\mathbf{x}). \tag{4.1}$$

Training new models is viewed as iterations of gradient descent in function space: Each model is a step in the direction of the gradient $\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}$. This is achieved by training the tree on the "pseudo-residuals" — values of the gradient on the training set points:

$$\tilde{y}_i = -\left[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \tag{4.2}$$

After we have found the direction of the m-th step, we need to find the best point (function) in that direction. In the space of functions represented by trees this corresponds to choosing the optimal prediction γ_{jm} in each leaf R_{jm} of the tree. In order to do this we need to solve the equation:

$$\gamma_{jm} = \text{argmin}_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma) \tag{4.3}$$

Algorithm 5 summarizes the technique.

This is a compressed description of gradient boosting. For a more detailed explanation see[28]. One important detail to note: regardless of the loss function, the trees fitting the gradient on pseudo-residuals are regression trees trained to minimize mean squared error. However, the way to calculate γ_{jm} (the actual prediction of a single leaf) from the pseudo-residuals of data points in the leaf depends on the loss that the algorithm is trained to optimize.

Algorithm 5 Gradient Boosting Framework (with trees)

for $m = 1$ **to** M **do**
  $\tilde{y}_i = -[\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}$
  $Tree_m = \text{TrainTree}(\mathbf{x}_i, \tilde{y}_i)$
  for $\forall R_{jm} \in Tree_m$ **do**
    $\gamma_{jm} = \text{argmin}_\gamma \sum_{\mathbf{x}_i \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x}_i) + \gamma)$
  end for
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + Tree_m(\mathbf{x})$
end for

4.1.1 L_2_TreeBoost

Now we describe the algorithm from the gradient boosting family intended for binary classification. L_2_TreeBoost optimizes a loss used in logistic regression — negative binomial log-likelihood.

$$L(y, F(\mathbf{x})) = \log(1 + \exp(-2yF(\mathbf{x}))), y \in \{-1, 1\}, \tag{4.4}$$

where

$$F(\mathbf{x}) = \frac{1}{2}\log\left[\frac{Pr(y=1|\mathbf{x})}{Pr(y=-1|\mathbf{x})}\right]. \tag{4.5}$$

Note that here, as in logistic regression, the additive model will predict log odds instead of the probabilities themselves. To convert $F(\mathbf{x})$ to probabilities we will need to make the following transformation:

$$Pr(y=1|\mathbf{x}) = 1/(1 + e^{-2F(\mathbf{x})}). \tag{4.6}$$

After substituting an abstract loss function with negative binomial loglikelihood loss Equation 4.2 becomes

$$\tilde{y}_i = 2y_i/(1 + \exp(2y_i F_{m-1}(\mathbf{x}_i))) \tag{4.7}$$

and Equation 4.3 that calculates the prediction in a single leaf becomes

$$\gamma_{jm} = \text{argmin}_\gamma \sum_{\mathbf{x}_i \in R_{jm}} log(1 + \exp(-2y_i(F_{m-1}(\mathbf{x}_i) + \gamma))). \tag{4.8}$$

This equation does not have an exact solution and is further approximated by a single Newton-Raphson step.

$$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|) \tag{4.9}$$

Modeling log odds function in L_2_TreeBoost is summarized in Algorithm 6.

Algorithm 6 L_2_TreeBoost

for $m = 1$ **to** M **do**
  $\tilde{y}_i = 2y_i/(1 + \exp(2y_i F_{m-1}(\mathbf{x}_i)))$
  $Tree_m = \text{TrainTree}(\mathbf{x}_i, \tilde{y}_i)$
  for $\forall R_{jm} \in Tree_m$ **do**
    $\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|)$
  end for
  $F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + Tree_m(\mathbf{x})$
end for

4.2 Algorithm

Gradient boosting and Groves share one important aspect of training. Each tree is built on the modification of the original training set where the values of input variables are retained, but the response is modified using the predictions of all trees currently in the ensemble. Additive Groves uses residuals, gradient boosting uses pseudo-residuals calculated from the gradient. This similarity is what allows us to combine the two algorithms. We will use backfitting and the gradual increase of tree size and number of trees from Groves and we will borrow training by gradient descent steps from gradient boosting. When we switch from stagewise forward training to backfitting, we lose the intuitive explanation of approximating the loss by an infinite number of gradient descent steps. However, we can explain the new algorithm as an attempt to choose the best directions for a fixed number of steps: after we have chosen the later steps, we always have the option to revisit and improve the first ones.

Notice that in case of least squares loss the resulting algorithm will be identical to the original regression Groves. In this paper we focus on creating a new classification algorithm, so we give detailed consideration to Gradient Groves trained to optimize the logistic regression loss. In other words, we focus on Groves combined with L_2_TreeBoost. Algorithm 7 shows backfitting merged with L_2_TreeBoost. Gradient Groves uses this algorithm instead of pure backfitting and retains the

outer loop that changes the numbers of trees and their size from original Additive Groves.

Algorithm 7 Backfitting optimizing logistic regression loss (before modifications

function Backfitting(M,TrainSet{$\mathbf{x}$,y})

for $m = 1$ **to** M **do**
$Tree_m = 0$
end for

repeat
for $m = 1$ **to** M **do**
Discard($Tree_j$)
$\tilde{y}_i = 2y_i/(1 + exp(2y_i \sum_{k \neq m} Tree_k(\mathbf{x}_i)))$
$Tree_m = \text{TrainTree}(\mathbf{x}_i, \tilde{y}_i)$
for $\forall R_{jm} \in Tree_m$ **do**
$\gamma_{jm} = \sum_{\mathbf{x}_i \in R_{jm}} \tilde{y}_i / \sum_{\mathbf{x}_i \in R_{jm}} |\tilde{y}_i|(2 - |\tilde{y}_i|)$
end for
end for
until (change from the last iteration is small)

end function

Several issues emerge when we try to combine these algorithms directly. To address them, we need to introduce several changes into the algorithm.

4.2.1 Fitting Large Trees

One of the core differences between Groves and boosting is the way how Groves deal with overfitting. Boosting prevents its model from overfitting by using small size of trees, shrinkage, early stopping or other regularization techniques[28]. In Groves, single models are allowed and even encouraged to overfit: more complex models can learn more detailed information, and excessive variance is later removed by bagging. This allows Groves to use large trees and model more complex structure. We want to retain this property of Groves, so we want to be able to allow large

trees that can fit the training data as closely as possible. Such trees are likely to contain pure leaves that have only positive or negative training data points. Recall that we train the trees to predict log odds of probabilities, therefore, in a pure node where the data is fit perfectly, we would expect to get the prediction $\gamma = \pm inf$. It turns out this is not the case with L_2_TreeBoost. In particular: a pure node means that either all $y_i = 1$ or all $y_i = -1$. For simplicity assume that this is the first iteration ($m = 1$) and that $F_0(\mathbf{x})$ was initialized with zero. Now from Equations 4.7 and 4.9 it follows that $\gamma = \pm 1$: this is very different from the infinity we were supposed to predict in the pure leaf.

We can see that the original L_2_TreeBoost algorithm fails to produce correct predictions in pure leaves. It happens because it approximates the solution of Equation 4.8 by using the Newton-Raphson method. The problem is that Newton-Raphson searches for a *local* minimum – it actually searches for the point where the gradient is zero – and therefore will not necessary produce the correct solution when the minimum is achieved at $\pm\inf$. This is exactly what happens when the node is pure, so the Newton-Raphson approximation cannot be used in this case. This issue was not important for gradient boosting because it uses small trees and small trees almost never have pure leaves. However, in order to use Groves with large trees we need to be able to deal with this special case.

Modification 1. *If all cases in a node belong to the positive (negative) class, simply predict* $+\inf$ *(*$-\inf$*) for log odds instead of using the Newton-Raphson step.*

Of course, predicting infinity might produce other problems. We will discuss these in the next paragraph. Here we want to stress only that the predictions for pure nodes produced by the original algorithm are erroneous and do not fit the intended gradient descent framework. Modification 1 is introduced to fix this issue and to replace incorrect approximation results with exact solutions in special cases.

4.2.2 Constraining the Variance

Bagging is an effective technique for reducing variance, but it has limitations. When the variance of the models becomes too high, bagging fails to eliminate it completely. One reason is that different versions of the training set are drawn from the same data again and again, making the training sets and the models built from them correlated. A second reason is that even if bagging would eventually succeed, for models with very high variance the number of iterations required can be impractically large.

Variance in predicting log odds of probabilities indeed becomes too high. Remember that log odds of probabilities 0 and 1 are equal to $+inf$ and $-inf$, therefore, predictions of the models can be arbitrarily large. One source of extremely large values is Modification 1. And even without that modification, Equation 4.9 can still produce extremely large values of γ_{jm}, prediction in a single leaf, if the values of residuals $\tilde{y}_i$ are close to 2 or 0.

A simple, and effective solution to the problem of excessively high variance is to set thresholds on predictions of each single leaf.

Modification 2. *If a leaf predicts a value larger than a threshold* Γ *(smaller than* $-\Gamma$*), replace its prediction with* Γ *(*$-\Gamma$*).*

Choice of Γ significantly influences the performance of Gradient Groves. Values that are too small decrease the performance, because they push predicted probabilities too far from 0 and 1 towards 0.5, while values that are too large will not be able to sufficiently restrict variance.

Remember that training a single grove begins with training one tree and then the number of trees is gradually increased. If we fix Γ as a maximum prediction of a single tree, then maximum prediction of the whole grove will be also gradually increased during training. To avoid this scenario and to make the threshold affect

groves of different size the same way, we need to set the threshold not as a maximum value for a single leaf, but as a maximum value for the prediction of the whole grove, which is the sum of M trees.

Modification 2a. *If a leaf predicts a value greater than a threshold* $\frac{\Gamma}{M}$ *(less than* $-\frac{\Gamma}{M}$*), replace its prediction with* $\frac{\Gamma}{M}$ *(*$-\frac{\Gamma}{M}$*). Empirically deduced good values for* Γ *depend on the data set but usually are in the range between* 1 *and* 20*.*

4.2.3 External vs. Internal Bagging

The normal sequence of steps during bagging is as follows:

1. Build models
2. Calculate predictions from models
3. Average predictions

If we follow this sequence with Gradient Groves in order to predict probabilities, the algorithm will look like this:

1. Build models (each of them predicts log odds)
2. Calculate predictions from models
    - predict log odds
    - convert to predictions of probabilities
3. Average predictions of probabilities

This algorithm does not work as one would want. Our models have variance and we want to eliminate it by averaging their predictions. However, if we follow the above sequence of steps, we have to apply a *non-linear* transformation to predictions before averaging. Some of variance that was originally present in the

predictions becomes bias during this procedure and can't be further removed by averaging. To prevent this, we need to average predictions before we convert them to probabilities.

Modification 3. *Bagging should be done internally relative to transformation of log odds.*

1. Build models
2. Calculate predictions from models (log odds)
3. Average predictions from the different log-odds models
4. Convert averaged predictions of log odds to probabilities.

4.3 Calibration

Calibration is a postprocessing technique that scales model predictions so that they better fit class probabilities. We experimented with two different calibration methods: Platt scaling [40], and Isotonic regression [51] using the pair adjacent violators (PAV) algorithm [4]. Platt's method fits a sigmoid to the predictions, and then uses this sigmoid to correct for distortion in the predictions. Isotonic regression is a more powerful method than Platt Scaling which can fit any monotonic function (of which the sigmoid is a special case) to the data. Although Isotonic Regression is more flexible, typically more data is required to fit the isotonic function without overfitting; Platt's method is more reliable when data is limited.

From our experiments we observed that Gradient Groves are already well calibrated and post-training calibration with either method does not significantly improve performance when the Γ threshold is chosen properly. However, performance can be significantly improved by calibration when Γ is too small and hence

probabilities are pushed too much towards 0.5. In that case calibration will automatically push the minimum predicted probabilities back to 0 and the maximum predicted probabilities back to 1.

4.4 Empirical Evaluation

In order to guarantee a reliable comparison of Gradient Groves with other classification algorithms, we made use of a recent empirical evaluation of ten machine learning algorithms. [13] evaluated boosted trees (BST-DT), bagged trees (BAG-DT), Random Forests (RF), support vector machines (SVM), neural networks (ANN), nearest neighbor (KNN), boosted stumps (BST-STM), decision trees (DT), logistic regression (LR) and naive Bayes (NB) across 11 binary data sets and 8 performance metrics. We evaluated Gradient Groves using the same data, folds, and metrics so that our results are directly comparable to theirs.

4.4.1 Experiment Settings

We have performed 5-fold cross-validation using exactly the same splits into training, test and validation sets as in [13]. On the training set we trained 440 Gradient Groves with different combinations of parameters α (controls size of tree), M (number of trees in each grove), and Γ (threshold discussed in Section 4.2). α is the maximum size of a leaf node: if the proportion of training set items in the node becomes less than or equal to α during training, such a node becomes a leaf. In our experiments we tried all possible combinations of the following values: $\alpha \in \{0.5, 0.2, 0.1, 0.05, 0.02, 0.01, 0.005, 0.002, 0.001, 0.0005, 0\}$, $1 \leq M \leq 10$, $\Gamma \in \{1, 2, 5, 10\}$. For each combination of parameters we used 100 iterations of bagging.

The metrics used in the evaluation are:

Table 4.1: Empirical comparison of Gradient Groves with 10 other algorithms. Scaled performance measures are averaged over 11 datasets

	C	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	mean
GRGRV	N	.868	**.835**	**.945**	**.972**	**.955**	**.941**	**.886**	**.904**	**.913**
GRGRV	Y	.864	.815	.943	.971	.955	.941	.859	.863	.901
BSTDT	Y	.843	.779	.939	.963	.938	.929	.880	.896	.896
RF	Y	**.872**	.805	.934	.957	.931	.930	.851	.858	.892
BAGDT	N	.846	.781	.938	.962	.937	.918	.845	.872	.887
RF	N	.872	.790	.934	.957	.931	.930	.829	.830	.884
BAGDT	Y	.841	.774	.938	.962	.937	.918	.836	.852	.882
SVM	Y	.824	.760	.895	.938	.898	.913	.831	.836	.862
ANN	N	.803	.762	.910	.936	.892	.899	.811	.821	.854
ANN	Y	.815	.748	.910	.936	.892	.899	.783	.785	.846
BSTDT	N	.834	.816	.939	.963	.938	.929	.598	.605	.828
KNN	Y	.757	.707	.889	.918	.872	.872	.742	.764	.815
KNN	N	.756	.728	.889	.918	.872	.872	.729	.718	.810
BSTST	Y	.724	.651	.876	.908	.853	.845	.716	.754	.791
SVM	N	.817	.804	.895	.938	.899	.913	.514	.467	.781
BSTST	N	.741	.684	.876	.908	.853	.845	.394	.382	.710
DT	Y	.648	.654	.818	.838	.756	.778	.590	.589	.709
DT	N	.647	.639	.824	.843	.762	.777	.562	.607	.708
LR	N	.636	.545	.823	.852	.743	.734	.620	.645	.700
LR	Y	.627	.567	.818	.847	.735	.742	.608	.589	.692
NB	Y	.579	.468	.779	.820	.727	.733	.572	.555	.654
NB	N	.496	.562	.781	.825	.738	.735	.347	-.633	.481

- ACC – accuracy
- FSC – F-score
- LFT – lift
- ROC – area under ROC curve
- APR – average precision
- BEP – precision/recall break even point
- RMS – root mean square error
- MXE – cross-entropy

See[12] and[13] for more details on these metrics.

11 different real data sets were used in the comparison. The selection of data sets is very diverse: they come from different fields, have different structure and favor different types of algorithms[13].

Table 4.2: Empirical comparison of Gradient Groves with 10 other algorithms. Performance averaged over 8 scaled performance measures.

	C	cov	adl	ltr.1	ltr.2	med	slac	hs	mg	cal	cod	bac	mean
GRGRV	N	.891	**.985**	.887	.957	.787	**.920**	.866	**.926**	**.978**	**.926**	.924	**.913**
GRGRV	Y	.884	.970	.876	.950	.756	.900	.869	.904	.969	.922	.916	.901
BSTDT	Y	**.938**	.857	**.959**	**.976**	.700	.869	**.933**	.855	.974	.915	.878	.896
RF	Y	.876	.930	.897	.941	**.810**	.907	.884	.883	.937	.903	.847	.892
BAGDT	N	.878	.944	.883	.911	.762	.898	.856	.898	.948	.856	**.926**	.887
RF	N	.876	.946	.883	.922	.785	.912	.871	.891	.941	.874	.824	.884
BAGDT	Y	.873	.931	.877	.920	.752	.885	.863	.884	.944	.865	.912	.882
SVM	Y	.765	.886	.936	.962	.733	.866	.913	.816	.897	.900	.807	.862
ANN	N	.764	.884	.913	.901	.791	.881	.932	.859	.923	.667	.882	.854
ANN	Y	.766	.872	.898	.894	.775	.871	.929	.846	.919	.665	.871	.846
BSTDT	N	.874	.842	.875	.913	.523	.807	.860	.785	.933	.835	.858	.828
KNN	Y	.819	.785	.920	.937	.626	.777	.803	.844	.827	.774	.855	.815
KNN	N	.807	.780	.912	.936	.598	.800	.801	.853	.827	.748	.852	.810
BSTST	Y	.644	.949	.767	.688	.723	.806	.800	.862	.923	.622	.915	.791
SVM	N	.696	.819	.731	.860	.600	.859	.788	.776	.833	.864	.763	.781
BSTST	N	.605	.865	.540	.615	.624	.779	.683	.799	.817	.581	.906	.710
DT	Y	.671	.869	.729	.760	.424	.777	.622	.815	.832	.415	.884	.709
DT	N	.652	.872	.723	.763	.449	.769	.609	.829	.831	.389	.899	.708
LR	N	.625	.886	.195	.448	.777	.852	.675	.849	.838	.647	.905	.700
LR	Y	.616	.881	.229	.440	.763	.834	.659	.827	.833	.636	.889	.692
NB	Y	.574	.904	.674	.557	.709	.724	.205	.687	.758	.633	.770	.654
NB	N	.552	.843	.534	.556	.011	.714	-.654	.655	.759	.636	.688	.481

For every performance measure we computed the values of all Groves on the validation set, chose the best combination of parameters (using the validation set) and reported the corresponding performance on the test set. Tables 4.1 and 4.2 show the results of our comparison with the other learning methods. The first table shows results for each metric averaged across all test datasets. The second table shows performances on each dataset averaged across all performance metrics. Bold font shows the best performance in each column. Notice that all scores on the various performance metrics are normalized to fall within range $[0.0, 1.0]$ and in such a way that a larger value indicates better performance. This way it is easier to compare performance across metrics such as accuracy (acc), where higher is better, and root mean squared error (rms), where lower is better.

The second column in each table shows if the predictions were calibrated or not (there are two lines in each table for each algorithm, performance with calibration

and without). We tried both of the calibration methods described in Section 4.3, and report only the performance of the better method. We refer the reader to the original comparison paper by [13] for more details about the experimental setup.

4.4.2 Results

On average, Gradient Groves is the best performing algorithm in this comparison. Moreover, it is the best algorithm on every metric but accuracy even without using any post-training calibration: results for Gradient Groves without calibration are in most cases better than results for Gradient Groves with calibration. Gradient Groves is the best algorithm not for all data sets: on some data sets it yields to calibrated boosting and sometimes also to other ensembles of trees, or to neural networks. However, its performance is remarkably consistent and remains among the top performers for most datasets. In particular, on 8 out of the 11 datasets both calibrated and uncalibrated Gradient Groves are among the top-4 algorithms. This stability and excellent performance allows Gradient Groves to outperform boosting on average. When boosting works poorly on a data set, its performance can be very low, but Gradient Groves never exhibits this poor behavior.

4.4.3 Gradient Groves vs. Additive Groves

Table 4.3: Uncalibrated Gradient Groves vs. calibrated Additive Groves. Scaled performance measures are averaged over 11 datasets

	C	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	mean
GRGRV	N	.868	**.835**	**.945**	**.972**	**.955**	**.941**	**.886**	**.904**	**.913**
ADGRV	Y	**.879**	.822	.944	.967	.944	.940	.883	.889	.908

Additive Groves is a regression algorithm, and we cannot compare it directly to Gradient Groves and other classification algorithms. When the datasets have binary (0/1) response, predictions of a regression algorithm can be greater than

Table 4.4: Uncalibrated Gradient Groves vs. calibrated Additive Groves. Performance averaged over 8 scaled performance measures.

	C	cov	adl	ltr.1	ltr.2	med	slac	hs	mg	cal	cod	bac	mean
GRGRV	N	**.891**	**.985**	.887	**.957**	**.787**	**.920**	.866	**.926**	**.978**	.926	**.924**	**.913**
ADGRV	Y	.882	.979	**.920**	.956	.735	.898	**.915**	.910	.903	**.979**	.918	.908

1 or less than 0. Additive Groves indeed provided many of such out-of-bounds predictions and for this reason it was impossible to calculate some of the metrics. However, after calibration predictions returned to $0 - 1$ range and therefore we were able to add calibrated Additive Groves into comparison. Tables 4.3 and 4.4 compare calibrated Additive Groves with uncalibrated Gradient Groves. Additive Groves also achieve high performance, however, Gradient Groves still have better results on average and, as noticed above, the latter algorithm does not require calibration. There is another reason why Gradient Groves might be preferred to Additive Groves for classification: Gradient Groves can be extended to multi-class case similar to how L_2_TreeBoost is extended to L_K_TreeBoost[28]. Additive Groves, a regression algorithm, can be adapted to binary classification, but not to multi-class case.

CHAPTER 5

INTERACTION DETECTION

We introduce a new approach to interaction detection. It is based on comparing the performance of *restricted* and *unrestricted* predictive models. This avoids the drawbacks of previous methods, because it does not require explicit modeling of interacting terms and reports only those interactions that are present in the actual input data. However, the choice of model and the restriction algorithm used are crucial for this framework. We explain why additive models are able to provide the required accurate restrictions and further show that Additive Groves works well in this framework. We also investigate how correlations in the data complicate interaction detection and suggest how this problem can be dealt with via feature selection.

The advantage of our new approach for interaction detection, compared with traditional statistical approaches, is that it is more automatic and does not require limiting the functional form that interactions might take. Statistical methods often represent only multiplicative interactions and thus may miss other forms of interactions. When little is known about the system under study, data-driven scientific discovery requires the data to "speak for themselves" with a minimum of analyst input or assumptions. It is possible to conduct a fully nonparametric analysis with the method we propose in this paper, which is particularly valuable for exploratory analysis.

5.1 Estimating Interactions

Let $F^*(\mathbf{x})$ be an unknown target function and let $F(\mathbf{x})$ be a highly accurate model of F^* that can be learned from a given set of training data. Furthermore, let $R_{ij}(\mathbf{x})$ denote a *restricted* model of F^* that is learned from the same training data. It

is restricted in the sense that it is not allowed to contain an interaction between x_i and x_j, but apart from this limitation should be as accurate a model of F^* as possible.

Our interaction estimation technique is based on the following observation. If x_i and x_j interact, then $F(\mathbf{x})$ should have significantly better predictive performance than $R_{ij}(\mathbf{x})$, because the latter cannot accurately capture the true functional dependency between x_i and x_j. On the other hand, if the two variables do not interact, then the absence of the interaction from the model should not hurt its quality. Hence in the absence of an interaction between x_i and x_j the predictive performance of the restricted and the unrestricted model should be comparable. Note that in order to get an adequate estimate of performance, we must measure it on test data not used for training.

Quantifying interaction strength. We can quantify I_{ij}, the degree of interaction between x_i and x_j, by the difference in performance between $F(\mathbf{x})$ and $R_{ij}(\mathbf{x})$. We measure performance as standardized RMSE: root mean squared error (RMSE) scaled by the standard deviation in the response function. Scaling is done to make the results comparable across different data sets; $\mathrm{StD}(F^*(\mathbf{x}))$ is calculated as standard deviation of the response values in the training data.

$$\mathrm{stRMSE}(F(\mathbf{x})) = \frac{\mathrm{RMSE}(F(\mathbf{x}))}{\mathrm{StD}(F^*(\mathbf{x}))} \tag{5.1}$$

$$I_{ij}(F(\mathbf{x})) = \mathrm{stRMSE}(F(\mathbf{x})) - \mathrm{stRMSE}(R_{ij}(\mathbf{x})) \tag{5.2}$$

Setting the threshold. To distinguish whether a positive value of I_{ij} indicates presence of an interaction or happened due to random variation, we measure whether the performance of $R_{ij}(\mathbf{x})$ is significantly different from the performance of $F(\mathbf{x})$. We follow common practice and define a difference of three standard deviations of the latter from its mean as significant. The distribution of $\mathrm{stRMSE}(F(\mathbf{x}))$

can come either from different random seeds for bagging or from different data samples (e.g., n-fold cross validation). The threshold for significant interactions then becomes:

$$I_{ij}(F(\mathbf{x})) > 3 \cdot \text{StD}(\text{stRMSE}(F(\mathbf{x}))) \tag{5.3}$$

Note that everything above naturally generalizes to higher-order interactions as long as there exists a method to restrict the model on a specific type of interaction.

5.2 Choosing a Prediction Model

To correctly estimate interaction strength with our model comparison technique, we have to make sure that a model has the following key properties:

1. High predictive performance when modeling interactions: if there is an interaction, it should be captured by the unrestricted model.
2. High predictive performance when the model is restricted on non-interacting variables: if there is no interaction, performance of the restricted model should be no worse than the performance of the corresponding unrestricted model.

The first requirement is satisfied by many learning techniques, e.g., bagged decision trees of adequate depth, SVMs, or neural nets. Boosted stumps, on the other hand, do not model interactions. Since they represent functions as the sum of components, each of which depends only on a single variable, boosted 1-level stumps cannot be used in our framework.

While many models satisfy the first requirement, the second requirement — that models perform as well when interaction between non-interacting variables is restricted — is far more challenging. Even when there is a straightforward way of explicitly preventing specific interactions, often the resulting restricted model will

not perform as well as the unrestricted model because the restriction may hamper the search in model space compared to the unrestricted model.

Consider a single decision tree. Variables in the tree can interact only if they are used on the same branch of the tree. So the obvious way to restrict interaction between specific variables is to not use one of them if the other already was used earlier on this branch. Now suppose there is no interaction between variables A and B, but they both are important — if the tree does not use one of them, its performance drops. Assume further that A is more important than B. The tree will tend to choose A earlier than B on all branches (in the worst case it will use A at the root) and will then never be able to choose B. Since B is important, the performance of this restricted tree will drop even though there was no interaction between A and B.

One might be tempted to address this problem with an ensemble method like bagging. Unfortunately the situation will not improve much. In bagging, every tree tries to capture the same function from a different sample of the train set. If A is more important, most trees will choose A before B, use of B will be restricted, and performance will drop as before.

To detect *absence of interactions* between important variables, we need to build a restricted model that uses these variables in different additive components of the function. Additive models based methods naturally fit this requirements. Each component in an additive model is trained on the residuals of predictions of all other previous models in the ensemble. The training set for the new model component is created as the difference between true function values and current predictions of the ensemble. This way, when the function has additive structure, different models (or groups of models) are forced to find and model different components of this structure as opposed to each modeling the whole function.

Not all models that fit residuals are suitable for this framework. Linear models do not model interactions, while generalized linear models disguise additive structure with a non-linear transformation. Neural networks pose problems because they either have additive structure (1 internal layer), or the ability to model complex non-linear functions (several layers), while we need an algorithm that combines both. Restricting interactions in a multi-level network splits it into subnets, ultimately leading to "groves of nets".

In this research, we used layered Additive Groves for the purposes of interaction detection. There exist other methods that might work as well, e.g., gradient boosting trained to minimize least squares loss[28]. However, it is important to understand that the two requirements stated in the beginning of this section are crucial and many (most?) learning algorithms do not satisfy them.

For Additive Groves, although dynamic programming training provides better performance for unrestricted models, we have encountered problems with it when training restricted models. Therefore we prefer layered Additive Groves for interaction detection. Note that we need to use layered training even for the unrestricted model in order for the performances to be comparable.

5.2.1 Parameter Space

Here we illustrate how the performance of a Grove depends on its two parameters—the number of trees and their size. Performance of a model depends on its complexity, which for trees roughly corresponds to the number of partitions the model creates. Complexity of a Grove can be controlled in two ways: through the size of the individual trees and the number of trees. To some extent, one would expect that a reduction in the size of individual trees can be compensated by a larger number of trees in the Grove. Our empirical observations confirm this intuition.

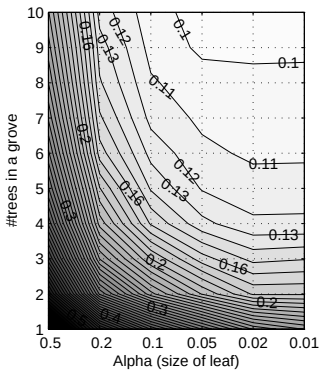

Figure 5.1: Performance (RMSE) of the unrestricted Grove on the synthetic dataset.

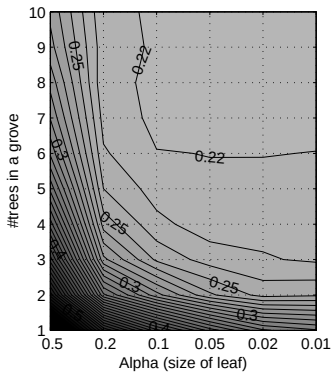

Figure 5.2: RMSE of the restricted Grove; restriction on interacting variables x_1, x_2.

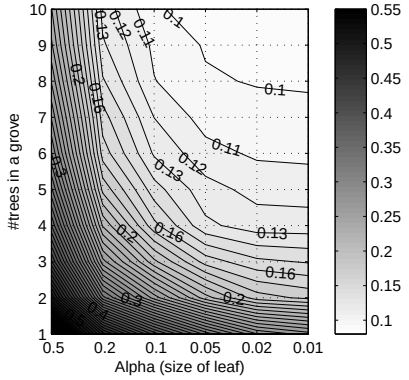

Figure 5.3: RMSE of the restricted Grove; restriction on non-interacting variables x_3, x_4

The Grove model has two main parameters: N, the number of trees in the Grove, and α, which controls the size of each tree (only nodes with more than a fraction of α of the training cases are split when growing the tree). Figure 5.1 shows a contour plot of how model performance depends on both parameters for a synthetic dataset without noise described in Section 3.1.2. The performance is measured as root mean squared error (RMSE) on the test set. Notice that lower RMSE implies better performance.

Figures 5.2 and 5.3 show an analogous performance grid on the same dataset for restricted Groves. In Figure 5.2 the Grove is restricted on *interacting* variables x_1 and x_2, in Figure 5.3 the Grove is restricted on *non-interacting* variables x_3 and x_4. We can see that the performance in Figures 5.1 and 5.3 is almost identical for every parameter combination. The performance of the model where interacting variables are restricted is noticeably worse. We observed similar results for other datasets.

It is worth noticing that Figures 5.1 and 5.2 are similar in the lower left corner. This happens because for these parameter values even the unrestricted model is not

complex enough to capture the interaction between x_1 and x_2. This highlights the fact that in order to detect interactions, we have to use models that are complex enough (i.e, have many large trees) to achieve the best possible performance.

5.3 Feature Selection

Correlations among features are common and complicate the task of detecting interactions. Suppose there exists an interaction between variables x_i and x_j. At the same time, a third variable, x_k, is present in the data. Assume it is highly correlated with x_j, to such an extent that the model can freely use either x_k or x_j with similar results. In this case we will not be able to detect the interaction between x_i and x_j. When we restrict the model to prevent a tree from using x_j, it can use x_k instead and performance will not drop. The same will happen when we try to detect an interaction between x_i and x_k.

Correlation among features is an intrinsic problem of high dimensional data that confronts all methods for interaction detection. For example, methods based on partial dependence functions[20] suffer from a similar problem. The unrestricted prediction model might sometimes use x_j and sometimes x_k. As a result it will find only weak interaction between x_i and x_j and also between x_i and x_k, even though the true interactions are much stronger. If there are more than two correlated variables (again, this is common in high-dimensional datasets), the interaction can be spread out in tiny portions over all of them, making it virtually impossible to detect.

As a consequence, before attempting to detect interactions, we must eliminate correlations. This can be achieved by a feature selection process, which removes some of the variables. The final set of variables should be a compromise between two goals: (1) The performance of the unrestricted model should still be good,

ideally at least as good as before feature selection. (2) Each variable should be important, i.e., if we remove it from the set of features, the performance of the unrestricted model should drop significantly. The second criterion also gives us an estimate of the maximum strength of interactions that we can detect: if the performance of the unrestricted model drops by δ when we remove x_i, then we cannot expect the performance of the best model restricted on x_i and x_j to drop by more than δ. The intuition here is that removing an important variable is a stronger restriction than prohibiting its interactions.

We use a variant of backward elimination[23] for the feature selection process. The main idea is to greedily eliminate all features (variables) whose removal either improves performance or reduces performance by at most Δ compared to performance on the full-feature data set. In our experiments we estimated $d = \text{StD}(\text{RMSE}(F(\mathbf{x})))$, where $F(\mathbf{x})$ is the unrestricted model, before running feature selection and used $\Delta = 3d$.

The feature selection procedure is not stable—it depends on the order in which we test each feature. For example, if we consider two completely correlated variables x_j and x_k, we can remove x_j and leave x_k in the set of the features. Or we can do exactly the reverse, depending on which variable we tried to remove first during feature selection. If there is a strong notion of which features should stay in the data set after feature selection, i.e., if we want to test certain features for interactions, the feature selection process should be modified so that features of interest are not removed.

5.4 Complexity Issues

One concern about interaction detection is the need to conduct a separate test for each interaction. If we want to test for all possible interactions, in theory we

need $O(n^k)$ tests, where n is the number of variables and k is the order of the interaction. However, such complexity is unlikely to be required in practice. First, the feature selection process usually leaves a relatively small set of features that makes it feasible to test all pairs for possible interactions. Second, as noted by[26], interactions possess an important monotonicity property. A k-way interaction can only exist if all its corresponding $(k-1)$-interactions exist. This fact is a straightforward consequence from the definition of a k-way interaction. Hence after we have detected all 2-way interactions, we need to test for 3-way interactions only for those triples of variables that have all 3 pairwise interactions present, and so on. As complex interactions are rare in real datasets, in practice we usually need only few tests for higher-order interactions. Some domains do pose an exception, for example, see our experiments on the $kin8nm$ dataset.

5.5 Experiments

We have applied our approach to both synthetic and real data sets. We can evaluate the performance of our algorithm on synthetic data because we know the true interactions; for real data we try to explain the detected interactions based on the data set description.

In all our experiments we used 100 iterations of bagging. Apart from that, Additive Groves requires two parameters to be set: N (number of trees in a single Grove) and α (fraction of train set cases in the leaf, controls size of a single tree). We determined the best values of α and N on a validation set and reported the performance of Additive Groves with these parameters on a test set. We ran each experiment for the unrestricted model 10 times, using different random seeds and therefore different bootstrap samples for bagging. From these results we estimated the distribution of performance and then calculated the interaction threshold us-

ing Equation 5.3. After that we ran the experiment for each unrestricted model only once. If the resulting estimate of the interaction was above the threshold, we considered it to be evidence of an interaction. Otherwise it was considered insignificantly different from zero, indicating absence of an interaction. Notice that due to variance, in the latter case the estimate could be even negative, but should always be close to zero.

5.5.1 Synthetic Data

This data set was generated by a function that was previously used in [26].

$$\begin{aligned} F(x) &= \pi^{x_1 x_2}\sqrt{2x_3} - sin^{-1}(x_4) + \\ &\quad log(x_3 + x_5) - \frac{x_9}{x_{10}}\sqrt{\frac{x_7}{x_8}} - x_2 x_7 \end{aligned} \tag{5.4}$$

Variables x_1, x_2, x_3, x_6, x_7, x_9 are uniformly distributed between 0.0 and 1.0 and variables x_4, x_5, x_8 and x_{10} are uniformly distributed between 0.6 and 1.0. Training, validation and test set contain 1000 points each. Best parameters were detected as $\alpha = 0.02$ and $N = 8$. Feature selection eliminated variables x_6 (not present in the function) and x_8 (virtually no influence on the response). For each of the 28 pairs of remaining variables we constructed a restricted model and compared it to the unrestricted model. Figure 5.4 shows the interaction value for each variable pair as computed by Equation 5.1. The dashed line shows the threshold. We can see a group of strong interactions high above the threshold — pairs (x_1, x_2), (x_1, x_3), (x_2, x_3), (x_2, x_7), (x_7, x_9). All cases without interactions fall below the threshold. There are also several weak interactions in the data set: our estimate for (x_9, x_{10}) is barely above the threshold and we failed to detect interactions (x_3, x_5) and (x_7, x_{10}). By construction, x_5 and x_{10} have a small range and their interactions are not significant. There is only one triple of variables with 3 pairwise

interactions detected: (x_1, x_2, x_3). A separate test correctly reveals that there is a 3-way interaction between them. Note that this is the only higher-order interaction that we need to test to conclude the full analysis. The original formula has another 4-way interaction, (x_7, x_8, x_9, x_{10}), but interactions of x_8 and x_{10} turned out to be very weak in the data, so the model did not pick them up.

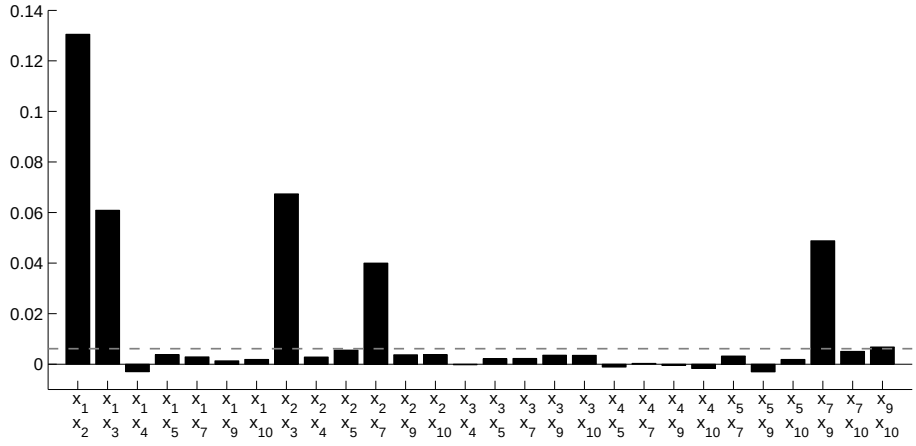

Figure 5.4: Interaction estimates on synthetic data

For more realistic results, we generated a version of the same data set with a $2:1$ signal-to-noise ratio. Now feature selection left only 5 variables: x_1, x_2, x_3, x_5, x_7, and results of interaction detection between those variables were qualitatively the same as the correspondent results for the data set without noise.

5.5.2 Real Data Sets

We have run experiments on 5 real data sets, 4 of them are regression data sets from Luís Torgo's collection[47], and the last one is a bird abundance data set from the Cornell Lab of Ornithology[11]. We used 4/5 of the data for training, 1/10 for validation and 1/10 for testing.

California Housing

California Housing is a regression data set introduced in [38]. It describes how housing prices depend on different census data variables. Parameters used: $\alpha = 0.0005$, $N = 6$. Feature selection identified six variables as important: longitude, latitude, housingMedianAge, totalRooms, population and medianIncome. [27] describes the joint effect of latitude and longitude on the response function. Our results confirm that there is a clear strong interaction between these two variables — the location effect on prices cannot be split into the sum of latitude and longitude effects. We have also found an evidence of interaction between population and totalRooms (Figure 5.5).

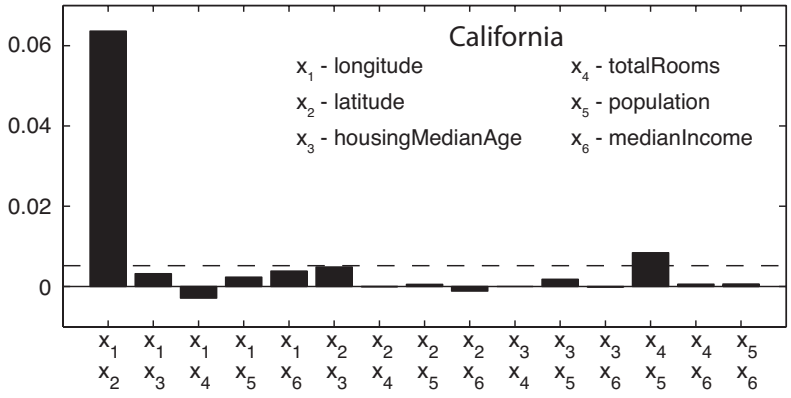

Figure 5.5: Interaction estimates for California Housing.

Elevators

This data set originates from an aircraft control task [10]. Parameters used: $\alpha = 0.02$ and $N = 18$. Feature selection left six variables: *climbRate*, p, q, *absRoll*, *diffRollRate*, *Sa*. We detected strong pairwise interactions in the triple (*absRoll*, *diffRollRate*, *Sa*) and a separate test confirmed that this is indeed a strong 3-way interaction (Figure 5.6). No other interactions were found.

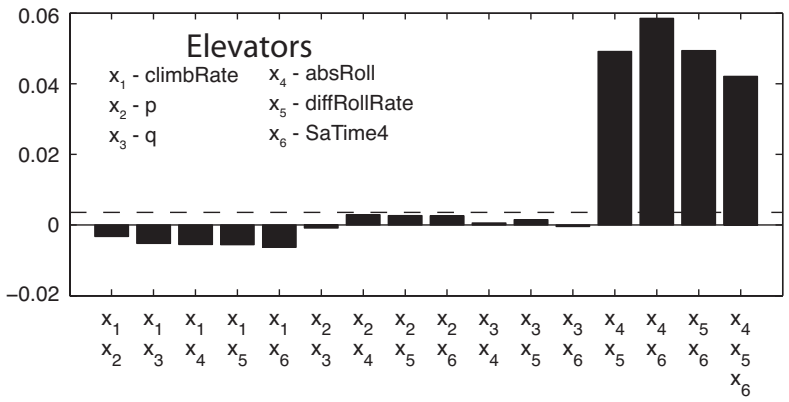

Figure 5.6: Interaction estimates for Elevators data.

Kinematics (kin8nm)

The *kin8nm* dataset from the Delve repository [42] describes a simulation of an 8-link robot arm movement. Its input variables correspond to the angular positions of the joints and it is classified as highly non-linear by its creators. Parameters used: $\alpha = 0.005$ and $N = 17$. Our analysis produced symmetrical results that reveal the simulation nature of the dataset: all 8 features turn out to be important, 2 of them do not interact with any other features and the other 6 are connected into a 6-way interaction (Figure 5.7). For brevity we show only results of tests for 2-way interactions and the final 6-way interaction, but we have also conducted tests for 20 3-way, 15 4-way and 6 5-way interactions between those 6 variables following the procedure described in Section 5.4. All tests confirmed the presence of interactions. *kin8nm* is the only data set where we had to test for many higher-order interactions. This is a property of the domain: the formula describing the end position of the arm based on joints angles results from interaction between most of the variables.

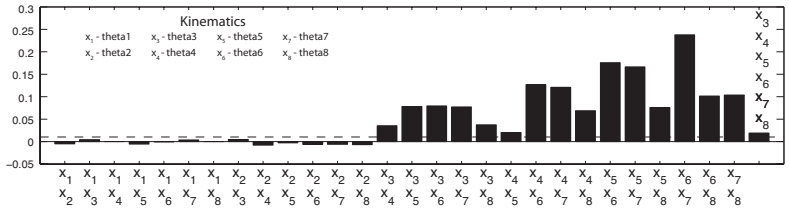

Figure 5.7: Interaction estimates for Kinematics (kin8nm) data.

CompAct

Another dataset from the Delve repository, it describes the level of CPU activity in multiuser computer systems. Parameters used: $\alpha = 0.05$ and $N = 18$. Feature selection left 9 variables: *lread*, *scall*, *sread*, *exec*, *wchar*, *pgout*, *ppgin*, *vflt*, *freeswap*. This data set turns out to be very additive. Although there are many 2-way interactions, they all are relatively small (Figure 5.8). The largest interactions are (*freeswap*, *wchar*), describing the joint effect of the number of blocks available for swapping and system write call speed, and (*freeswap*, *vflt*), describing an interaction between the same available blocks variable and the number of page faults.

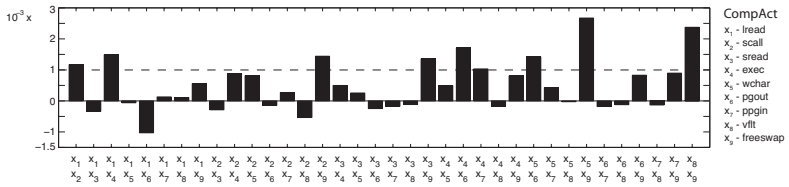

Figure 5.8: Interaction estimates for CPU Activity (CompAct) data set.

House Finch Abundance Data

We tested our approach on a dataset with sightings of House Finches in the North-Eastern US as introduced in[11]. The strongest interactions that we detected are between the following variables: (*latitude*, *longitude*, *elevation*) and (*year*, *latitude*,

longitude). The first 3-way interaction describes the effect of geographical position which is expected to be non-additive. But the interactions between year and location is less trivial. Normally one would not expect that the effect of latitude or longitude on bird abundance would be very different in different years. However, it turns out that during the decade covered by the data set, the population of House Finches was suffering from an eye-disease that was spreading slowly and was responsible for changing the effect of geographical location on bird abundance over time. Our results show that interesting domain information like this can be discovered with the help of interaction detection analysis.

5.6 Comparison of Models: Statistical Testing

In this section we discuss possible methods to test how significant the difference between unrestricted and restricted model is. For all experiments in this thesis the first of the described methods was used, primarily because of running time concerns, however, it has some drawbacks and other tests could be possible.

5.6.1 One-Sample Z-Test

In the method we currently use we estimate whether the difference between the models is significant in the following way: for a single test we generate a single restricted model and many (we use 10) unrestricted models. Randomization in unrestricted models comes from using different bootstraps for bagging. We further evaluate the performance of all models on the test set, estimate mean ($\hat{\mu}$) and standard deviation ($\hat{\sigma}$) of the distribution of performances for unrestricted model, and declare that the restricted model is significantly different from the unrestricted if its performance is different from $\hat{\mu}$ by more than $3\hat{\sigma}$.

This procedure can be explained as following in statistical terms. First, we make assumptions about the distributions of performance values.

1. Performance estimates of models of the same type (i.e. restricted or unrestricted) are distributed normally.
2. When there is no interaction, performance estimates of restricted and unrestricted models come from the same distribution, i.e. normal distribution with the same mean and variance.

Then, we form the null hypothesis. It states that the interaction is absent and therefore, following the second assumption, the single estimate of a restricted model comes from the same distribution as all estimates of unrestricted models.

After that we assume that $\hat{\mu}$ and $\hat{\sigma}$ are accurate estimates of true parameters of the normal distribution and perform a z-test to test how likely performance of restricted model comes from normal distribution with mean $\hat{\mu}$ and standard deviation $\hat{\sigma}$. We use z-score of 3 as a threshold, it corresponds to a confidence level of 0.9987 for a one-sided test.

Although this approach works quite well in practice, it has several drawbacks from the theoretical point of view. First, z-test assumes that parameters of normal distribution are known. As in our case they are estimated, t-test probably would have been more appropriate. Second, and this is a more serious issue, the second assumption might be too strict. Namely, the part that claims that distributions of results for restricted and unrestricted models have the same variance when there is no interaction might be wrong in some cases. Even if the models are equally good (i.e., have the same performance on average), these are different types of models, therefore variance in their performance values can be different. If the variance for restricted models is higher than for unrestricted, then we risk rejecting the true null hypothesis with higher probability then we expect.

In the rest of the section we discuss other possible approaches that don't have the problem described above.

5.6.2 Two-Sample T-Test

The most statistically sound approach would be to collect many samples of performance estimates for both restricted and unrestricted models, estimate their distributions, and test whether their means are the same. One possible test is a two-sample t-test for different variance. The assumptions are now simplified:

1. Performance estimates of models of the same type (i.e. restricted or unrestricted) are distributed normally.
2. When there is no interaction, performance estimates of restricted and unrestricted models come from distributions with the same mean.

In this test, t-value is calculated by the following formula:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{5.5}$$

where s is an unbiased estimator of variance. Distribution of this statistics is approximated with Student's t distribution with the number of degrees of freedom calculated as

$$df = \frac{(s_1^2/n_1 + s_2^2/n_2)^2}{(s_1^2/n_2)^2/(n_1 - 1) + (s_2^2/n_2)^2/(n_2 - 1)} \tag{5.6}$$

Although this approach is more theoretically sound, its requires creating large number of models for each test and for this reason is impractical.

5.6.3 Resampling Errors

There exist another approach that requires building a collection of many unrestricted models and only one restricted, and at the same time it does not require

equal variance assumptions. Here is the outline.

1. Build a single restricted model.
2. Calculate errors (predictions minus true values) on the train set.
3. Shuffle error values.
4. Create new labels for the train set by adding (shuffled) error values to predictions of the restricted model.
5. Build an unrestricted model on this new train set.
6. Estimate the performance of this model on the test set.
7. Repeat steps 3–6 several times.
8. Build and evaluate an unrestricted model on the original data.
9. Test whether the performance estimate received on the last step comes from the same distribution as numbers acquired on steps 3–6.

The main underlying assumption in this approach is that we will be able to approximate the true function by the restricted function well enough when interaction is absent, and that datasets with reshuffled errors will essentially represent the same function as the original dataset. More formally, there are assumptions concerning Bias/Variance/Noise decomposition of errors of the restricted model and they are the following:

1. Noise term is normally distributed with zero mean and fixed variance with respect to different points in the data.
2. Bias and Variance terms are negligible.

First assumption is quite standard and realistic for most data sets, however, the second assumption might pose problems. The variance term might be relatively

large for some datasets, therefore, the new data sets might be significantly different from the original.

Another problem with this approach is still the complexity. At first it seems that the number of generated models is the same as in the first approach: a single test requires one restricted model and a set of unrestricted models. However, there is a big difference when we compare the number of models required to conduct all interaction detection tests for one data set. In the first approach we need to generate a set of unrestricted models only once and then it can be reused. Each new test requires building only one new (restricted) model. In this approach, however, we would need to build the whole set of models for every test, none of the models can be reused.

5.6.4 Estimating Variance of Bagged Models

A possible way to reduce the complexity of all methods described above is to estimate the distribution of performance values using a single model only. This might be possible if we remember that an Additive Groves model is an fact an ensemble of single Grove models. More, it is an ensemble produced by bagging: every prediction is an average of single predictions and therefore distributions of performance estimates of single Grove models and the correspondent distribution of performance of Additive Groves are related. Here is an initial analysis of how we could estimate the latter through the former.

Squared errors of a single model (MSE) are often decomposed as a sum of squared bias and variance (noise is considered a part of bias in this approach).

- Bias is an error due to the type of model — an average absolute error that all models of this type will make.
- Variance term is variance of error due to differences in the models. Errors are

assumed to be normally distributed with zero mean and standard deviation σ across different models. Therefore variance term itself is distributed as χ^2_1 streched by a factor of σ^2 and has mean σ^2 and variance $2\sigma^2$.

Squared bias stays constant for different models (we define this constant as C). Therefore MSE of a model is distributed as χ^2_1 streched by a factor of σ^2 and shifted by C. Mean of this distribution is equal to $C + \sigma^2$ and variance equals to $2\sigma^2$.

Suppose we do bagging and average N bagged models. Bias stays constant, while variance is decreased by factor of N (standard deviation is decreased by factor of $\sqrt{N}$). Therefore MSE of bagged models will be distributed as stretched and shifted χ^2_1 with mean of $C + \frac{\sigma^2}{N}$ and variance of $\frac{2\sigma^2}{N}$.

When we build a single ensemble of bagged models (e.g. single Additive Groves model), we build N single models (e.g. single Groves) and therefore can directly estimate mean and variance of their MSE values, calculate estimates of C and σ and then predict what the distribution of MSE of bagged models will look like even without training several of those models. We did not extend this line of reasoning to produce the strict mathematical form of distribution of $stRMSE$, the measure that we actually use, but it seems to be possible at least on an approximate level.

A possible caveat in this approach is in that bagging does not always decrease the variance by a factor of N. It happens only in an ideal theoretical case when the data set is infinite. Otherwise bagged models are not exactly independent because they are trained on bootstraps drawn from the same data set. In some cases variance of bagged models might end up being visibly higher then estimated, especially when the data set is relatively small and noisy (see, for example, our ornithology data sets in Section 7).

5.7 Discussion

We presented a novel technique for detecting statistical interactions in complex data sets. The main idea is to compare the predictive performance of unrestricted models to restricted models, which do not contain the to-be-tested interaction. Although this idea is quite intuitive, there are significant practical challenges and few algorithms will work in this framework. We demonstrated that layered Additive Groves can be used in this approach due to its high predictive performance for both restricted and unrestricted models. Results on synthetic and real data indicate that we can reliably identify interactions.

CHAPTER 6

PRACTICAL ISSUES

In this section we describe a variety of issues that appear when working with large noisy real-world data sets. We discuss many subtleties and problems one needs to be aware of in order to perform interaction detection analysis successfully when working with such data sets. In particular:

1. Correlations in the data are a big problem. A restricted model can use a correlated variable to effectively bypass the restriction and achieve good performance even if there is an interaction. Therefore, a thorough feature selection process is required before we can test for any interactions.

2. The second issue is the appropriate choice of parameters for the learner (Additive Groves). Parameters resulting in the best possible predictive performance will not necessarily result in the most reliable model for interaction detection.

3. After detecting an interaction, it is important to visualize and interpret it. During this process we have to be aware that interactions in the model and interactions in the data sometimes are not the same. For example, because the distribution of the interacting variables' values is not independent, some parts of visualisation plots might not be supported by any real data and thus should be ignored when interpreting the results.

6.1 Overfitting Issues

Experiments with Groves on synthetic and standard real data sets from repositories might lead to a belief that complex models with larger trees always demonstrate performance at least as good as their simpler counterparts as long as enough bag-

ging iterations are performed. Such belief might be based on the fact that this is how the ensembles of bagged trees usually behave. However, when working with real noisy data sets, we discovered that this claim is not always true neither for Groves nor for bagged trees themselves.

6.1.1 Overfitting in Bagged Trees

Bagging[6] is a well-known ensemble method that creates variation in a set of models by sampling from the training set, and then decreases variance by averaging the predictions of these models. Large decision trees are low-bias, high variance models that benefit significantly from bagging, and often bagging works best with larger trees. However, on noisy ornithology data, large trees perform much worse than small trees, even after a large number of bagging iterations. Figure 6.1 shows the performance of 100 bagged trees of different sizes on the commonly used California Housing[35] data set, and for the Horned Lark, one of the species in the RMBO data set (described in more details in Section 7). The difference in performance of bagging for large and small trees on the two data sets is striking.

This poor performance of bagging large trees sometimes happens on large noisy data sets that have few useful attributes. Bagging can never remove variance completely, because it draws versions of the data again and again from the original training data set. The different training samples inevitably overlap and produce partially the same results. On cleaner data sets with many attributes this effect is not very visible, but on real data that do not possess these qualities, the situation can be as bad as illustrated in Figure 6.1.

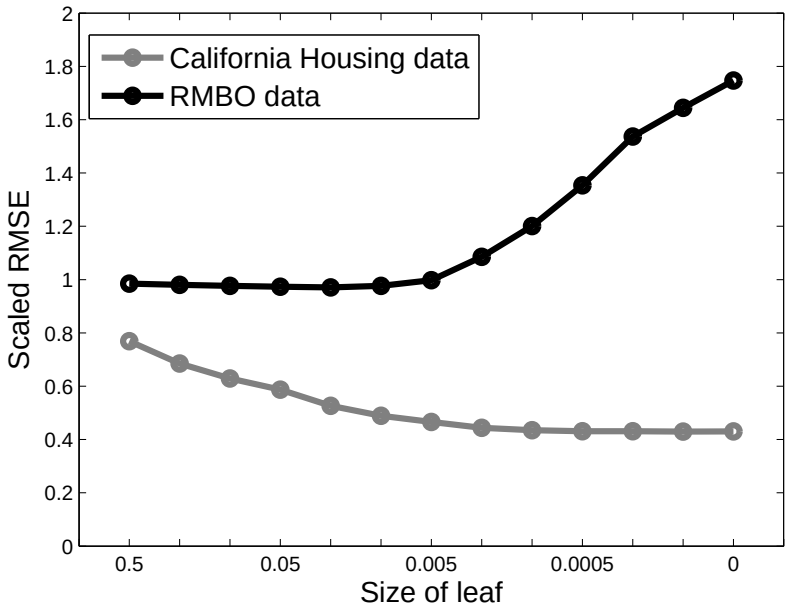

Figure 6.1: Performance of 100 bagged trees on "standard" California Housing data set vs. noisy RMBO data. Small RMSE means better performance. Even with bagging large trees (right) overfit much more then small trees (left) on RMBO data.

6.1.2 Overfitting in Additive Groves

Experiments in Section 3.1 and Section 5.2.1 suggest that Additive Groves are robust to overfitting as long as they are bagged sufficiently many iterations. This is the case as long as the bagging process succeeds in removing most variance. Unfortunately, similar to the observation above about bagging individual trees, there are some data sets where this is not achieved. Figure 6.2 shows a contour plot of how performance of Additive Groves depends on values of α and N on one of ornithological data sets. Performance is measured using weighted root mean squared error, therefore smaller numbers correspond to better performance. We can see that the best performance is reached for comparably small models, and then rapidly decreases when the models become more complex. This property of the data makes the interaction detection process more complicated.

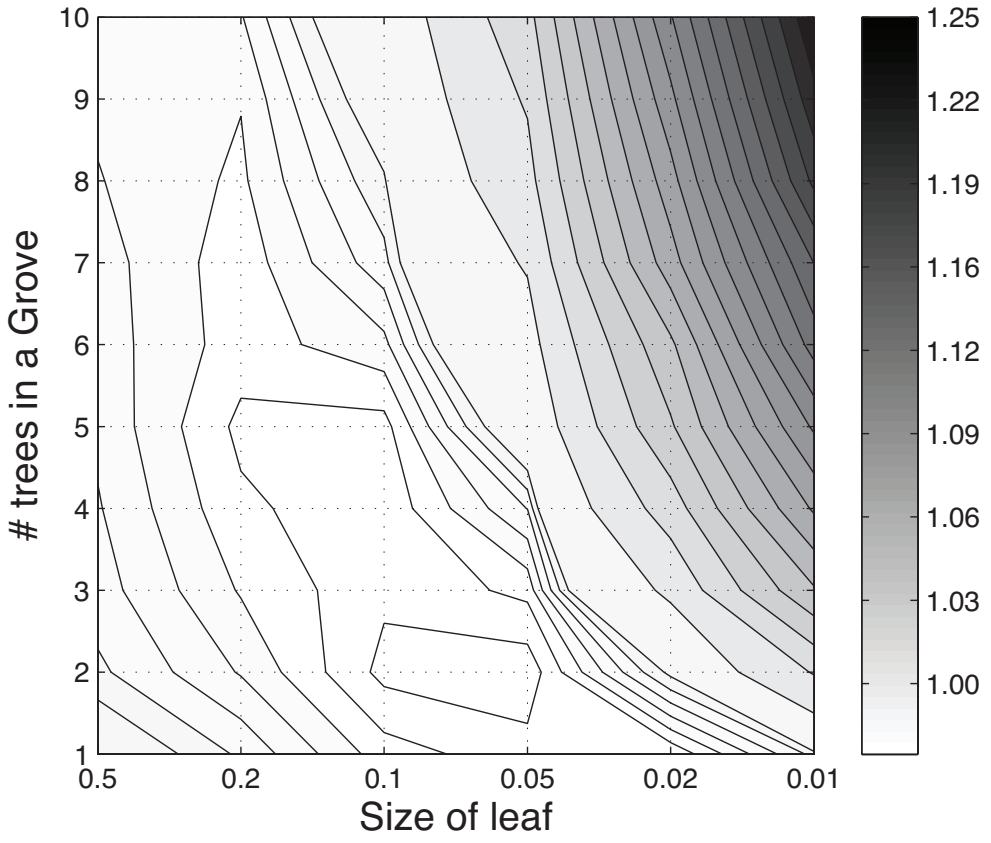

Figure 6.2: Weighted RMSE of 100 bagged Additive Groves on RMBO data for Horned Lark abundance

6.2 Feature Selection

As already mentioned in Section 5.3, correlations between variables pose a problem for any kind of interaction detection algorithm. For our approach based on model comparison, they can effectively "hide" existing interactions. Suppose we want to test for an interaction between x_i and x_j, and there is another variable x_k that is almost identical to x_j. When we restrict a model on interactions between x_i and x_j, it can use x_k instead of x_j and thus bypass the restriction. Hence even if x_i and x_j interact, we can not discover this unless we remove x_k from the data.

In general, for detecting an interaction involving a variable x_i, removing x_i from

the data set should result in a significant drop in performance. In fact, removing x_i is a stronger limitation for the model than restricting it on an interaction with x_i. If performance does not drop when we completely remove x_i, we cannot expect it to drop when restricting on an interaction with x_i. More precisely, if performance after removing x_i drops by Δ, then performance of a model restricting on an interaction involving x_i theoretically can not drop by more than Δ. For these reasons we have to eliminate all variables (features) from the data until we are left with a set of variables such that removing any of them would significantly decrease model performance. We discuss how to do this in the remainder of this section.

When working with large high-dimensional data sets, thorough feature selection based on generating different models for different combinations of features is practically infeasible due to the large number of possible feature subsets. We therefore adopt a two-step approach. In the first step we perform fast but rather crude elimination of the least important features. In the second step we perform a more careful elimination.

6.2.1 Fast Feature Evaluation

In this section we discuss possible approaches to fast and crude feature selection. We first describe existing technique — sensitivity analysis, then suggest faster white-box tree-based methods and then empirically show that some of these latter methods produce similar results to sensitivity analysis and thus can be reliably used for the first phase of feature selection.

Black Box Approach: Sensitivity Analysis

Sensitivity analysis assesses the importance of a feature by comparing the performance of the model on a real test set against performance on a perturbed test set.

To measure the importance of feature A, the perturbed test data set is generated by adding noise to the A-values. More precisely, all A-values are shuffled, essentially permuting the original vector of A-values (when viewing the data set as a matrix whose rows are the different observation records and columns correspond to the different features). If the attribute is important, performance should drop on the perturbed test data set compared to the real one, because the model relies on the spoiled values when making predictions.

There are many different measures of model performance. For our sensitivity analysis, we selected a diverse set of commonly used measures to avoid measure-related bias. In particular, we based the importance rankings on three different metrics: accuracy (ACC), root mean squared error (RMS), and ROC area[41] (ROC). The performance was measured on a separate test set, i.e., none of the test records was used for training the bagged trees.

Table 6.1 shows the feature sensitivity results for the House Finch in BCR 30, sorted by RMS. An entry in the table reports the relative loss in performance between real and perturbed test data set, computed based on the corresponding measure. For example, assume the accuracy on the real test data is x. Then, after permuting the latitude values, the accuracy changes to y. The corresponding relative loss for latitude then is computed as $(x-y)/x$. Since RMS is the only one of the selected measures for which lower values indicate better performance, we report $(y-x)/x$ for it. Breiman[7] used a similar technique to measure variable importance; this method is related to randomization and permutation tests used in statistics[32].

Sensitivity analysis is a relatively fast method for estimating variable importance. Once the model is trained, we only need to evaluate its performance for different perturbed test data sets, one for each attribute. This is much faster than

Table 6.1: Top-20 attributes for sensitivity analysis, sorted by RMS

attribute	ACC	RMS	ROC
latitude	0.079	0.15	0.070
longitude	0.0056	0.045	0.014
numfeeders_hanging	0.012	0.034	0.013
halfdays	0.013	0.034	0.015
yearseason	0.012	0.032	0.014
dayselapsed	0.016	0.030	0.014
numfeeders_thistle	0.0098	0.022	0.0095
ave_fam_sz	0.0016	0.011	0.0040
effort_hrs_atleast	0.0030	0.010	0.0045
asian	0.0014	0.0091	0.0030
elev_ned	0.00023	0.0067	0.0024
evgr_trees_atleast	0.00072	0.0050	0.0017
numfeeders_suet	0.00045	0.0048	0.0016
gcsnow2912	8.5E-05	0.0045	0.0015
pop00_sqmi	8.5E-05	0.0041	0.0012
vacant	0.00048	0.0037	0.0011
count_area_size	0.00016	0.0037	0.0012
other	-0.00063	0.0035	0.0011
elev_gt30	0.00012	0.0032	0.0010
ave_hh_sz	-0.00093	0.0030	9.8E-4

the costly approach of re-training models for different sets of attributes, which takes between 1 and 2 hours on a single CPU modern PC and is necessary for feature selection methods[30, 31, 23]. Nevertheless, for large high-dimensional data sets like PFW, even sensitivity analysis requires considerable resources: evaluating the sensitivity of a *single* feature using the 32K test cases for BCR 30 takes about 4-5 minutes. Using this approach for all 197 features of interest (or even pairs or larger sets of features) and for all 1000 interesting BCR-species combinations requires access to expensive high-performance computing resources. In the following section we propose efficient heuristics to address this issue.

White Box Approach: Looking at Trees

The methods discussed in this section take advantage of the fact that we are using ensembles of decision trees. Trees enable us to look inside the model to see what attributes have been selected. Selected attributes clearly are important

predictors for observation probability because they separate positive and negative observations. If attributes are important for many of the trees in the ensemble, then this provides strong evidence of their overall importance. The main challenge is to define a good measure for quantifying the importance of an attribute in a tree and in an ensemble of bagged trees.

We have implemented a range of different possible ranking methods that use only the information about the tree structure and how a training set is partitioned by the different trees. This information is available once the ensemble is built, so there is no need to generate new models or new predictions in order to calculate these rankings. This is a clear advantage over black box methods like sensitivity analysis or feature selection. Our white box approach speeds analysis up by a factor of more than 500! On some of the datasets we can compute the complete ranking of *all* features in less than 2 minutes (no matter which of the methods introduced below we are using), compared to 4-5min *per feature* for sensitivity analysis.

The importance score of an attribute for the tree ensemble is computed by summing the importance scores on the individual trees. Notice that the scores computed by different methods are not normalized in any way, and hence are not comparable. All we can derive from the different methods are attribute rankings. To illustrate the differences between the methods, we will use the simple tree shown in Figure 6.3. It splits on three attributes A, B, and C. The training set has 100 records; numbers in parentheses indicate the number of records affected by the corresponding split (i.e., the number of records in the corresponding subtree).

We consider the following methods for computing attribute importance scores on a single tree.

Number of trees (#trees). The first simple measure that we tried defines

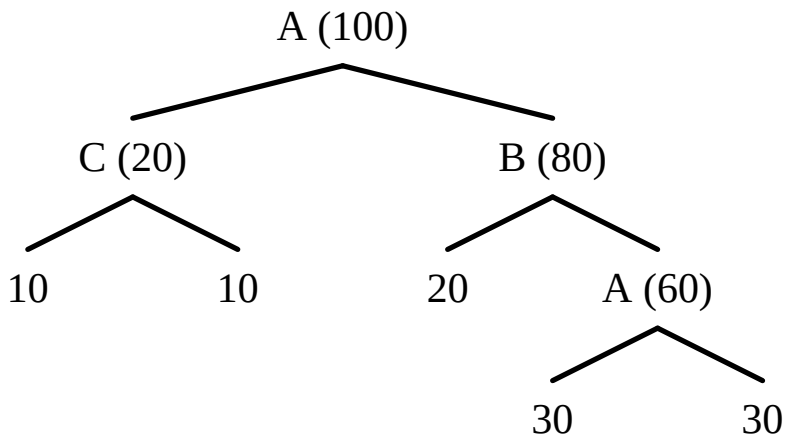

Figure 6.3: Sample decision tree

importance of the attribute as the number of trees in the ensemble that are using it. It turned out that this measure does not work — more than half of all attributes were used by all trees and therefore their importance could not be compared using this method.

Number of nodes (#nodes). This is a very simple measure. The score of an attribute is computed as the number of nodes in the tree that selected this attribute for the split. In our example attribute A gets importance score 2, while B and C receive importance scores of 1 each. This method will give too much weight to continuous attributes, because the tree can split on them more often. The following methods address this issue.

Weighting by height (height). Most greedy tree growing tends to choose the most important attributes early, so they appear higher in the tree structure. This method weights each node inversely proportionally to the length of the path from it to the root. The root itself is considered to have importance 1, so in the example attribute A receives importance $1+1/3$ (importance of root + importance of the rightmost subtree split), attributes B and C each have importance $1/2$.

The example in Figure 6.3 illustrates a problem with the height-based weights.

Attributes B and C receive the same weight, whereas splitting on them affects different numbers of cases in the data set. To correct for this, the following methods take into consideration the number of training cases affected by the split.

Weighting by size of training set — multiple counting (multiple). This method weights a node by the number of training cases in its subtree, i.e., the cases affected by the split at this node. In the example, attributes A, B, and C receive scores of 160, 80 and 20, respectively.

Weighting by size of training set — single counting (single). As with weighting by the number of nodes, there is a risk that continuous attributes will get over-weighted when using the multiple counting of training points. In the example, the 60 records in the lower-right subtree with root A are counted twice towards A's score. To fix this problem, the single counting method assigns weight zero to all nodes that have an ancestor with the same split attribute. In the example A receives an importance score of 100 instead of 160, while the scores for B and C do not change.

Weighting by size of training set — giving weight to the path (path). This method is between the extremes of single- and multiple counting. Intuitively training records from every leaf are distributed evenly between the splits on the path from the root to the leaf. Each split is still counted, even if there is another split on the same attribute in an ancestor node. In our example, the 30 records from the rightmost node are distributed between the two splits on A and the one split on B, i.e., 20 points go to A and 10 to B. Similarly, the 10 points from the leftmost leaf are given to A and C, in this case 5 points to each. Overall A receives an importance score of (counting inputs of all leaves from left to right) $5+5+10+20+20 = 60$, B gets $0+0+10+10+10 = 30$ and C gets $5+5+0+0+0 = 10$. It is worth mentioning that importance scores for all attributes sum to the size

of the training set over each tree in this method. A similar method was used by Friedman[20] for estimating attribute importance in an ensemble of rules.

Comparison of Rankings

For the comparison we used one of real-world ornithological data sets describing the abundance of House Finches the U.S. Atlantic coastal plain region from southern-most Maine to northern-most Virginia. This data set has $92,514$ observation records, reporting the House Finch to be present $55,860$ times, and absent $36,654$ times. 197 attributes were available for analysis.

Comparison and analysis of different rankings revealed the following results. All three measures based on size of training set in splitting nodes seem to be very similar (Figure 6.4). This result is surprising, because different ways of estimating continuous attributes could in theory have significant influence on resulting rankings. In practice we observed only minor differences.

#nodes and *height* produced rankings that are very similar to each other, but differ from the previous group. Figure 6.5 shows that *height* and *#nodes* almost always agree, but are very different from the diagonal where they would be if they were correlated with *single*. Subsequent tests showed that results of these methods are less reliable than those of *single*, *multiple* and *path* (see next subsection).

One of the sensitivity analysis rankings — *sensitivity-rms* — shows a lot of similarity with the three most reliable methods from the "white-box" group (Figure 6.4). *sensitivity-acc* tends to agree with them only for the top ranked features and then shows a significant amount of discrepancy (Figure 6.5). Accuracy is known to be a measure with higher variance, but RMS is very stable. Hence we have more confidence in the results of *sensitivity-rms*. *sensitivity-roc* produced results similar to *sensitivity-acc* and therefore is omitted from the plot.

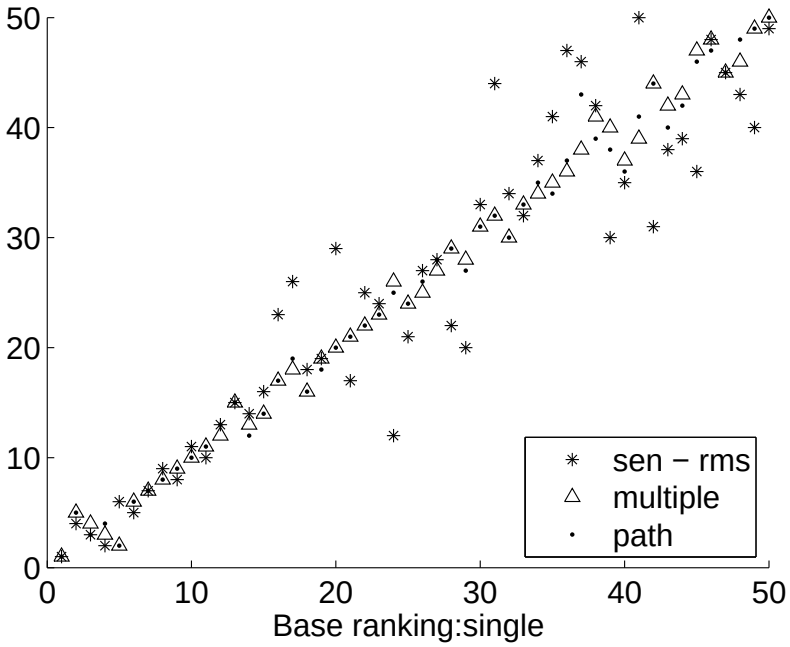

Figure 6.4: Comparison of different rankings (first 50 features shown). X-axis represents attributes in the order induced by ranking *single*, y-axis measures their position in other rankings.

6.2.2 Sanity Check

There is no guarantee that taking the top-ranked features from any of these importance measures will yield an ensemble with good predictive power. While prediction accuracy is not the only goal of this study, it is a necessary precondition. Clearly we cannot hope to learn something about this domain by studying inaccurate models. Also, ecologists are interested in comparing the important attributes of a species occurrence in different BCRs. This can be achieved by comparing rankings, but only after checking that some minimum predictive performance is met in all analyses to be compared.

As a sanity check, we compared the performance of bagged trees trained using all features with bagged trees trained using only the top 20 features from the different importance rankings. With all features, the bagged trees achieve a RMS

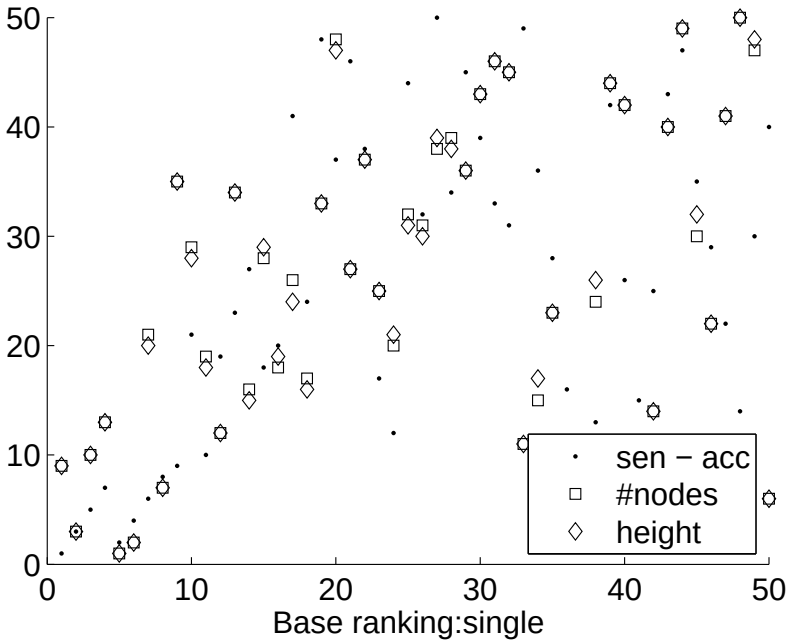

Figure 6.5: Rankings that do not agree well with *single*. The farther from the diagonal each point is, the larger is the disagreement.

of 0.3469, accuracy of 0.8336, and area under the ROC curve of 0.9012.

Figure 6.6 plots the ensemble's RMS performance when only the top N features from each ranking are used, for different values of N. Because the rankings differ from each other, different features are included at each point for the different lines (Table 6.2 and the RMS column from Table 6.1). The overall pattern is similar for accuracy and ROC area, so we omit those graphs.

We make several observations from Figure 6.6. First, the ensembles built using only 20 features perform quite well, although not quite as well as ensembles using all the features. The top 20 features do seem to catch most of the predictive power found in the full feature set. This gives us some confidence in relying on these measures as indicators of which features are important for modeling the PFW domain.

Second, while the rankings from single counting, path counting, and sensitivity-

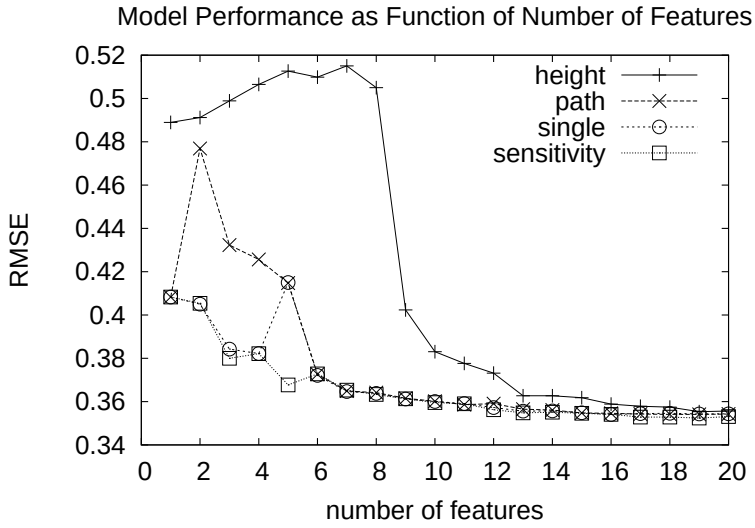

Figure 6.6: Performance as a function of the number of features used for training. Each line represents a different method for ordering features by importance—yielding slightly different sets of features.

rms analysis show similar behavior, the height-based ranking behaves very differently. This agrees with the finding above that the height importance measure is not as highly correlated with the other measures.

One surprising aspect of this graph is that all the lines go up at least once: *path* at feature 2, *single* at feature 5, *sensitivity* at feature 6, and *height* for the first half of the graph. This phenomenon is partly caused by the feature *dayselapsed*; whenever it is added, performance gets worse in this graph. Given that all the measures rank this feature highly, and the ecologists believe it to be an important predictor, this is rather surprising.

By 16 features, however, all the rankings have very similar performance. For the most part performance asymptotes around 16 features, therefore these features can be used for the next steps of our analysis.

Table 6.2: Top-20 attribute rankings; 'numfeeders_' is abbreviated as 'nf_'.

	height	**path**	**single**
1	dayselapsed	latitude	latitude
2	yearseason	dayselapsed	halfdays
3	halfdays	nf_hanging	nf_hanging
4	temp_lo_atleast	longitude	longitude
5	temp_hi_atleast	halfdays	dayselapsed
6	precip_len_atleast	yearseason	yearseason
7	effort_hrs_atleast	nf_thistle	nf_thistle
8	snow_dep_atleast	effort_hrs_atleast	effort_hrs_atleast
9	latitude	ave_fam_sz	ave_fam_sz
10	nf_hanging	elev_ned	elev_ned
11	nf_ground	asian	asian
12	nf_suet	pop00_sqmi	nf_suet
13	longitude	nf_suet	count_area_size
14	snow_cov_atleast	vacant	pop00_sqmi
15	nf_platfrm	count_area_size	vacant
16	pop00_sqmi	elev_gt30	black
17	elev_gt30	black	age_65_up
18	nf_water	ave_hh_sz	elev_gt30
19	asian	age_65_up	ave_hh_sz
20	black	houden	houden

6.2.3 Backwards Elimination

To make the first step of feature selection fast enough, we used only bagged trees. Now, for the more finegrained second step, we want to use the more expensive Groves. This is important because Groves will be used for interaction detection, hence we have to be sure that the remaining features are important from the perspective of the Grove models. At this step we do not know anything about how to set the Grove parameters α and N. We therefore build Grove models for the data set with its remaining set of features selected at a previous step with a variety of parameter combinations. From this "grid" of models (grid points are defined by combinations of parameter values), we select values for N and α that resulted in the best performance. These values are used for all models that are built in the next series of feature elimination steps. For some exceptionally noisy data sets it is possible that the best performance is produced by small models consisting of few small trees or even single trees ($N = 1$). In the latter case Additive Groves

become equivalent to traditional bagging of decision trees.

Recall that in order to be able to run effective interaction detection, we need to be left with a small set of important features. Important here means the following property: If we remove this feature, the performance drops by more than Δ. Δ needs to be defined to indicate a significant difference. We estimate the distribution of Grove performances on the data with all features currently in the set by creating the model several times with different random seeds. We then define $\Delta = 3 * \sigma$, where σ is the standard deviation of the estimated distribution. Then we perform standard backwards elimination. We begin with a model for the data set with all current features. Then we try to remove features one-by-one. If the performance does not drop by at least Δ, the feature is removed permanently. If it does, the feature is considered important and left in the data. Removing features can change the distribution of performances so this distribution needs to be recalculated occasionally. We recalculate the distribution when selection can't remove any more features with the current estimates of the distribution.

This algorithm was described in Section 5.3. It implicitly assumes that removing a feature will decrease the performance. However, this is not always the case for noisy data sets and an extension of the algorithm is required. Trees can mistakenly use "bad" features and benefit when those features are removed and we have seen cases of significant improvement in performance during the second step of feature selection. To handle this case, we extended the algorithm as follows: If performance is *better* than the original estimate by Δ, the algorithm must recalculate the estimate of the performance distribution.

6.3 Choosing Parameters for Interaction Detection

After we are left with only few important features, we need to choose the right Grove model (right values of parameters) for interaction detection. At this stage interaction detection should be easy for "standard" data sets, where complex Groves perform at least as well more simple models: We find the parameters for the best performing Grove model and then compare performance of restricted and unrestricted models using these parameters. With the challenging noisy data sets this is not always possible. Our model should meet (or be a compromise between) the following requirements:

1. It should be as complex as possible to catch all interactions.
2. It should have sufficient additive structure to allow for restrictions.

Ideally, to meet the first requirement, we need to choose the parameters that give the best performance. To meet the second requirement, we need to choose large N. From our experience, $N = 8$ usually is a safe value, $N = 6$ will work for most data sets, but smaller values usually hurt the performance of the restricted models. As mentioned above, it is easy to meet this requirement when bagging successfully removes most overfitting. One can increase complexity of a model without harming performance, as long as the model is bagged sufficiently. However, for noisy data we might observe that the best performance is achieved by a rather small model, while for more complex models the performance rapidly decreases below baseline. Figure 6.2 shows the performance of the model for Horned Lark after the feature selection. The best performance is achieved for $N = 2$ and trees of moderate size, increasing complexity on any of these parameters decreases performance.

Selecting the final parameters for interaction detection is different for different data sets, and occasionally requires multiple trials with a human in the loop.

Fortunately, one needs to do this only once for each response function, regardless of the number of features and interactions and the selected Grove parameters remain the same for the rest of the interaction detection process. Our experience can be summarized as follows:

- Since interaction detection uses the same basic model for the restricted and unrestricted case, the process is fairly robust with respect to choosing Grove parameters. Even with Grove parameter values that result in suboptimal performance, we can still successfully discover interactions. In most cases we can lose $\approx 8 * \Delta$ of predictive performance without hurting final interaction results.
- It is safer to choose a parameter combination for which Groves slightly underfit (simpler than the best model), rather than overfit because variance will be higher with the overfit models making the results less reliable.
- Even if there is no clearly optimal point with large N on the grid, we can try points with small N and set the threshold for interaction presence higher then usual when estimating the performance difference.

For example, we selected $N = 6$ and $\alpha = 0.2$ for Horned Lark abundance data based on the countour plot in Figure 6.2 and the rules described above.

Note that if different Grove parameters are selected than those used during backward elimination, it is necessary to run another round of backward elimination to make sure that each feature is still important for the new Grove configuration. With the new parameters, removing the feature should result in performance drop of at least Δ. If this is not the case, sometimes several more features will have to be eliminated to make the features that remain important once more. Interaction detection can only proceed for attributes that are important in the Grove model.

Similarly to how we define whether an attribute is important, interaction is considered significant if the difference between performance of the unrestricted and restricted models is more than Δ. Notice that values of Δ are different for different data sets and model parameters and often indicate the amount of variance in the model.

6.4 Visualization

After we detected a presence of an interaction between two variables x_i and x_j, we want to see what their joint effect on the response function looks like. In other words, we need to represent the response as a function of x_i and x_j only. After that we can plot the joint effect of two variables as several one-dimensional plots, each of which shows the dependence of the response value on x_i for a fixed value of x_j. Different lines on the plot correspond to different values of x_j. For example, Figure 7.2 shows the joint effect of elevation and edge density of shrub patches on the abundance of Lark Buntings. Each line correspond to an effect of shrubs at some fixed level of elevation. Non-parallel regions of the lines correspond to interactions and can provide us with insight into its nature. In this example we can see that presence of shrubs shows a positive effect on abundance of Lark Buntings on the lowest elevation, but on the higher elevations larger amounts of shrubs patches on the contrary, discourage this birds. (See Section 7.4 for more details on this interaction.)

A technique to create such two-dimensional models, partial dependence plots, was introduced by Friedman[28] as a tool to visualize the effects of a fixed number of variables averaged over the values of all other variables. The general "black-box" method for generating partial dependence plots is rather complicated and requires generating many synthetic data points. Fortunately, there exists an easier method

for regression tree models that takes advantage of the ability of trees to deal with missing values. When a case is missing the value of an attribute tested at a node, it descends to the child nodes with weights corresponding to proportions of training cases that went to each child. Cases apportioned this way reach multiple leaves with different weights, and the predictions of each leaf are combined using these weights. Given this approach to handling missing values in regression trees, a plot of the effect of two specific variables can be created that averages over the values of all other variables by setting the values of all other attributes to missing and plotting the performance obtained by sweeping the values of the two variables in question.

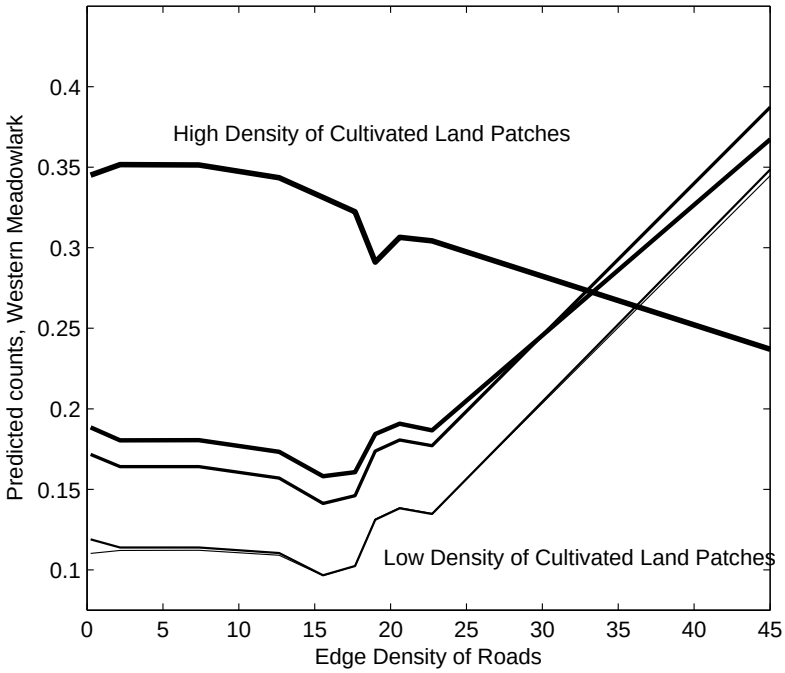

Figure 6.7: Western Meadowlark. Partial dependence plot produced with an unrestricted model shows a spurious interaction between density of roads and density of patches of cultivated crops

It is very important to notice that partial dependence plots by themselves are unreliable for interaction detection, because they depict interactions in the model instead of the data. Hooker[27] demonstrated that potential spurious interactions

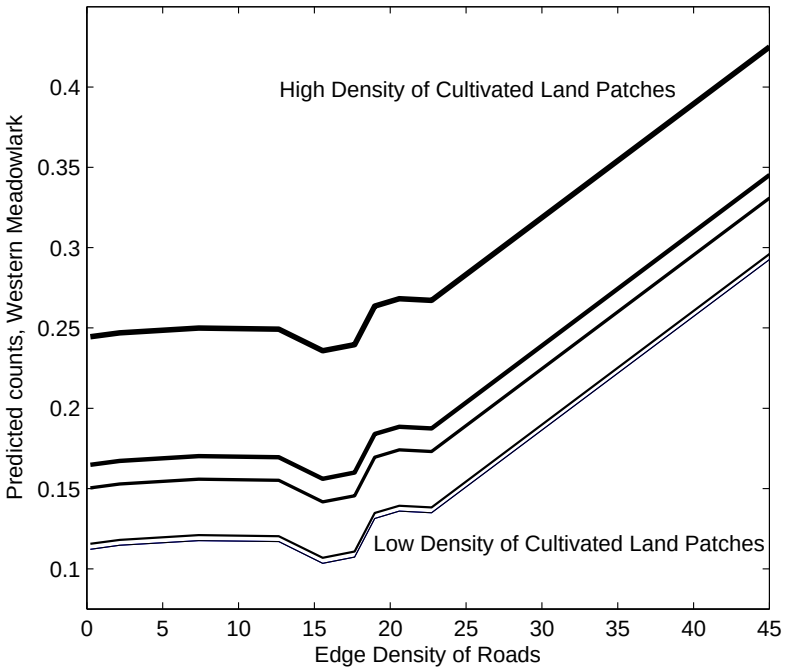

Figure 6.8: Western Meadowlark. Partial dependence plot produced with a restricted model without an interaction between density of roads and density of patches of cultivated crops

of arbitrary strength can appear in a partial dependence plot. This happens when some parts of prediction model are unsupported by the data and only emerge because of a presence of a few outliers.

Here is a stark example that emerged during our analysis of RMBO data: Figure 6.7 pictures a partial dependence plot for joint effect of presence of roads and cultivated crops areas on Western Meadowlark abundance generated by an unrestricted model. The plot clearly shows a strong interaction similar to the one we have just seen on Figure 7.2. However, there is no such interaction in the data! The restricted model that does not have this interaction has the same predictive performance: our performance comparison method estimated the size of interaction as -0.00009 and the significance threshold as 0.0005, which clearly indicates absence of interaction. [1] Figure 6.8 shows a similar plot produced by a

[1] When estimating a size of a non-existing interaction, negative numbers insignificantly differ-

restricted model. We can see that the effect of roads corresponding to the highest level of density of cultivated land patches is now very different from the previous picture. However, the performance of the model is the same. The explanation is that there are very few points with this level of cultivated land density in the data, clearly not enough to estimate a real effect. The interaction that we could see on Figure 6.7 is a mere random fluctuation. This example illustrates that partial dependence plots should be used for visualization only, when we already have confirmed the presence of interaction in the data by comparing restricted and unrestricted models.

ent from 0, of course, can happen as often as positive numbers. Negative number *significantly* different from 0 would indicate some problem, most probably bad choice of Groves parameters.

CHAPTER 7

EXPERIMENTAL VALIDATION ON OBSERVATIONAL ECOLOGY DATA

In this section we demonstrate our approach to interaction detection on real data describing the abundance of different species of birds in the prairies east of the southern Rocky Mountains. This data is very noisy. In fact, predictive models built from this data perform only slightly better than baseline. We show, however, that even when the data is this noisy, it is still possible to detect interactions between important features and the response function. We demonstrate and interpret the results of our analysis for several bird species.

7.1 Rocky Mountain Bird Observatory Data

The data used in our analyses come from the data warehouse of the Avian Knowledge Network (AKN)[2], an international collaboration of government and non-government institutions focused on understanding the patterns of distribution and dynamics of bird populations across the Western Hemisphere. This collaboration is creating the framework for gathering and storing existing and new bird-monitoring data from all available sources. It organizes these resources in such a way as to enhance application development, archiving, visualization and exploration, and makes these data generally available. The AKN also creates information products that use its data resources to produce visualizations such as maps, graphs, and tables, as well as scientific and technical analyses.

We selected data from one bird-monitoring program run by the Rocky Mountain Bird Observatory (RMBO)[3] for this analysis. The monitoring program, called the Section Survey, has collected counts of birds of different species observed at over 10,000 sites across a large part of the region known as the shortgrass prairie

(Figure 7.1). This is an arid zone in the rain shadow of the Rocky Mountains,

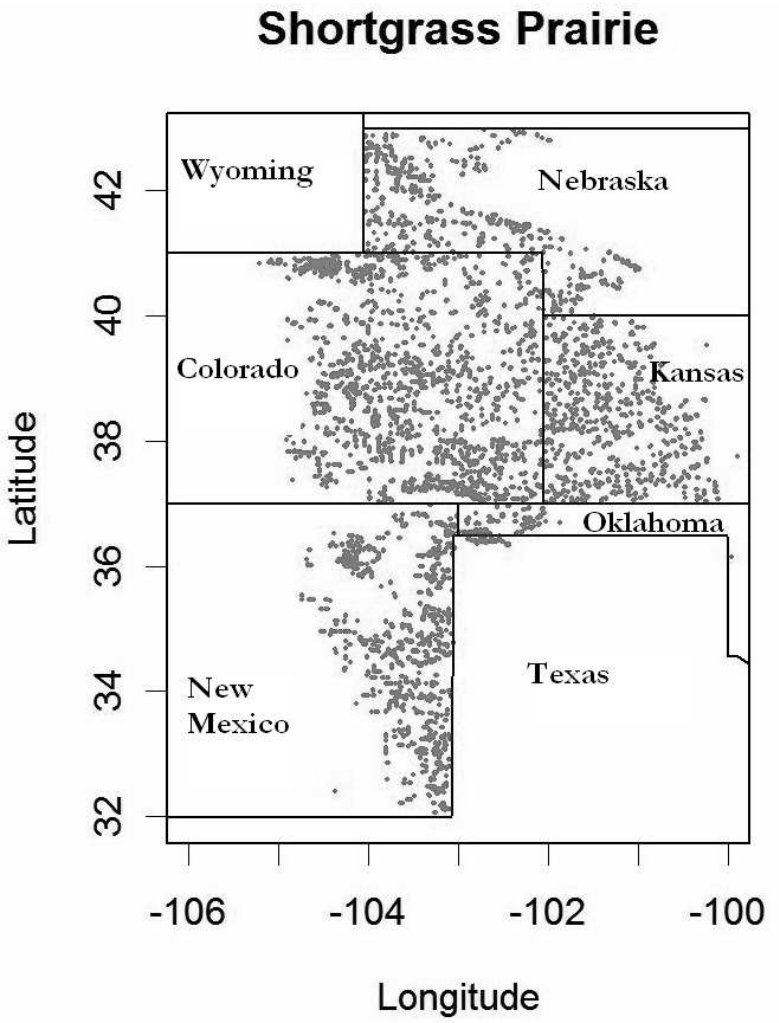

Figure 7.1: Observation sites.

characterized by short and sparse vegetation. Bird species specialized to grassland habitats, including those living in the shortgrass prairies, are some of the fastest and most consistently declining bird species in North America[39]. The Section Survey monitoring scheme is one effort to understand the causes and identify management actions that would reverse these declines. The Section Survey collects data on both abundances of birds (using a distance-sampling protocol[8]), as well as local vegetation at the survey sites. The goal is to identify associations between bird abundance and local vegetation, and the objective of identifying management

actions (such as livestock grazing regimes) that would make habitat more suitable for grassland bird species. For our analyses, we used the numbers of detected birds within 100 meters of the observer in a 3-minute period as the response variable, with a different response for each species identified on the survey.

When choosing where to live, birds consider not just local habitat characteristics — such as those measured by the Section Survey protocol — but also habitat configuration over larger regions[43, 48]. Therefore a more complete understanding of management actions requires not just an understanding of the effects of local habitat features, but also needs to place their effects within the context of larger-scale habitat configuration. Stated differently, we need to identify whether local habitat manipulation will always be effective at increasing bird populations, or whether local management will only be effective within specific larger-scale configurations of habitat. We include the larger-scale habitat configuration using interpreted satellite imagery from the 2001 U.S. National Land Cover Data[1], which classify habitat acoss the United States into 21 classes. The finest resolution of this data is a 30m by 30m pixel, and we have aggregated the NLCD data into a series of concentric squares (See Appendix, Table A.1), each centered on the pixel containing a bird census location. Various measures of habitat configuration (See Appendix, Table A.2) were calculated from these aggregations using the program FRAGSTATS[34]. These habitat configuration metrics, combined with the observed bird count response variable, are the data we analyse. The resulting data sets contain 700 features and 20000 observations for each bird.

7.2 Choice of Loss Function

Our technique for finding variable interactions is based on the comparison of the performance of models. To test for an interaction between variables x_i and x_j, we

train two models. The *restricted* model is not allowed to model the interaction between the variables. For the unrestricted model, there is no limitation in terms of which interactions can be modeled or not. If appropriate models are used, then the difference in performance between restricted and unrestricted model indicates the strength of the interaction between x_i and x_j.

The first fundamental challenge is to select the appropriate performance measure, or *loss function*. A common choice for general regression problems is root mean squared error (RMSE). However, this metric is less appropriate for bird observation data, which are counts. RMSE penalizes *absolute* deviation from the true response value. For example, predicting 25 birds instead of 20 will be penalized as heavily as predicting 5 birds when there were none. This is not desirable because the estimation error for the smaller response value is much more serious. For this reason analysis of point counts is often conducted using the logarithm of the original response function.

Unfortunately, working with log-transformed response values has an undesirable side-effect on the interaction detection task. Instead of discovering additive structure in the original function $F(\mathbf{x})$, we would now search for additive structure in the different function $\log(F(\mathbf{x}))$. Since $\log(f_1) + \log(f_2) = \log(f_1 \cdot f_2)$ for any response values f_1, f_2, we would in fact model multiplicative structure, instead of additive, in the original function F. Modelling multiplicative interactions might be of interest as well, but if we want to understand simpler additive interactions, working with log counts is not appropriate.

So what loss function should be used to penalize errors for low counts more? Instead of changing the response function, we change the loss that our models are trying to minimize. In order to still obtain a simple additive loss and at the same time achieve approximately the same effect as log-transforming the counts, we use

the first 3 terms of the Taylor expansion of the squared error of log counts. Since the first 2 terms of this particular expansion are equal to 0, this is equivalent to only using the third term:

$$(log(y+1) - log(F+1))^2 \approx \left(\frac{1}{y+1}(y-F)\right)^2. \tag{7.1}$$

Here y corresponds to the original response, F corresponds to the predicted value. A constant value (usually 1) is added to the counts before taking the logarithm in order to be able to allow zero counts. To derive this approximation, we view the loss function as a function of F with y fixed and take the Taylor expansion at the point $F = y$.

We substitute squared error in RMSE with the obtained weighted squared error $\left(\frac{y-F}{y+1}\right)^2$ and refer to the new loss as weighted RMSE. More, to make the results comparable across data sets, we use a standardized version of this metric: we divide it by similarly weighted standard deviation of response in the data set. The baseline performance by such standardized metric is equal to 1 on every data set and smaller numbers indicate better performance.

We would like to point out how challenging predictive modeling of RMBO data is. The improvement over baseline typically is only 2%-5%. For example, for Horned Lark, the bird we could extract most information about, the best value of performance we could achieve is 0.974 (measured by loss discussed above with baseline 1.0).

7.3 Feature Selection Details

At the first step we selected 50 useful features for each species. To do this, we used one of the "white-box" feature evaluation techniques from Section 6.2.1. In particular, we used the "multiple counts" method. This technique ranks attributes

based on how often trees in the ensemble use them in their nodes. The larger the subset of the train set in the node, the larger the score of the splitting attribute in that node. Experiments in Section 6.2.1 showed that multiple counts, the simplest and fastest of these metrics, produces results of similar quality compared to more expensive methods.

As we mentioned above, using large trees hurts performance for the noisy RMBO data. Hence for each species we generated several ensembles of 100 trees of different sizes, tested their performance on the test set and then chose the best performing one to use for determining feature importance. In most cases these were ensembles consisting of relatively small trees, up to ≈ 10 or 20 nodes.

After we have chosen 50 most important features for each of the species, we ran more detailed backward elimination analysis. Given the weak predictive performance of models trained on the RMBO data, we were not surprised that feature selection left few important features for most bird species. In the best case (Horned Lark) we had 8 features left, in the worst cases, only 1 or 2.

7.4 Results of Interaction Detection Analysis on RMBO Data

In this section we present and explain a few of the interactions we found in the RMBO data using the interaction detection method described earlier. This interactions provided new findings about both collected data and biological relationships that were previously unknown before, and yet are consistent with the general body of ecological knowledge.

The most complex, allbeit small, interaction that we identified was for Lark Buntings (*Calamospiza melanocorys*), with elevation and density of scrub/shrub

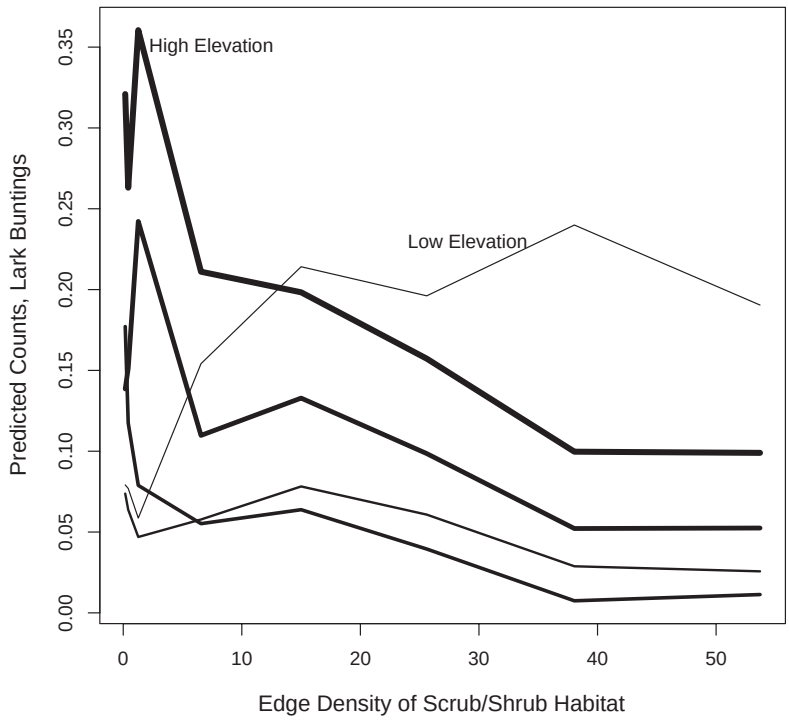

Figure 7.2: Lark Bunting. Interaction between elevation and density of edges of scrub/shrub vegetation patches

edges simultaneously affecting bunting abundance (Figure 7.2). Size of interaction is estimated as 0.00037, significance threshold as 0.00032. A higher density of scrub/shrub edges denotes an area in which a moderate proportion of the habitat contains short woody vegetation patchily inter-mixed with other habitats. At the lowest elevation sites, farthest from the base of the Rocky Mountains, Lark Buntings were more abundant in areas with a higher amount of patchily-distributed scrub/shrub vegetation. However, closer to the Rocky Mountains, the presence of scrub/shrub habitat inhibited Lark Buntings from settling. We believe that this result indicates that the habitat classified as "scrub/shrub" represents very different things in different parts of the study region, and that at higher elevations "scrub/shrub" contains plant species or habitat configurations that are unsuitable for Lark Buntings. In other words, we believe that this unexpected finding tells us something about our predictors rather than about the birds we are studying,

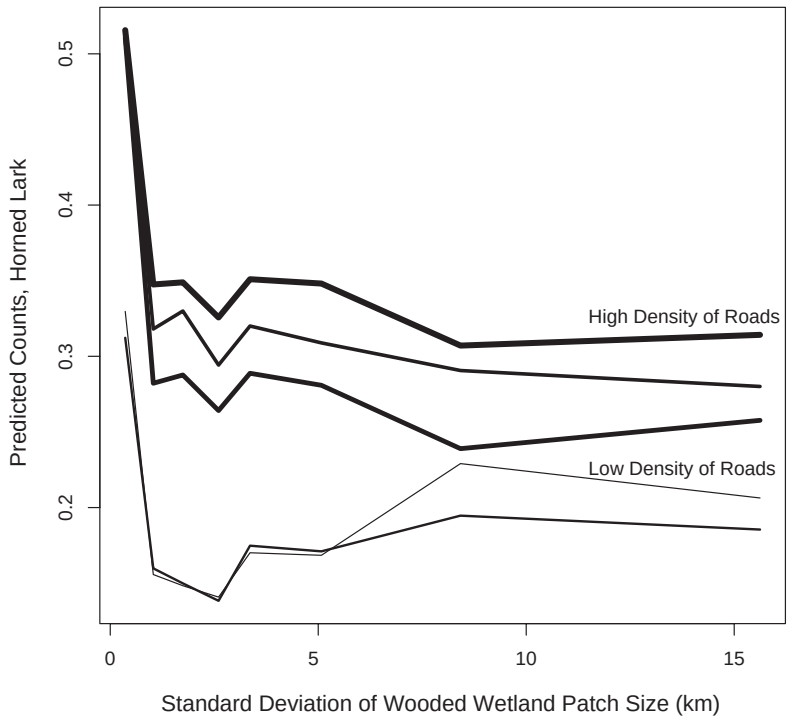

Figure 7.3: Horned Lark. Interaction between standard deviation in patch sizes of wooded wetlands and density of roads, which are entirely edges at the pixel resolution of our habitat data

specifically that the scrub/shrub habitat class is more heterogeneous than the classification would at first lead us to suspect.

The Horned Lark (*Eremophila alpestris*; known as the Shore Lark in Europe) is a species widely distributed across the Northern Hemisphere. It preferentially lives in barren habitat with short and patchy vegetation. The most unexpected interaction that we found was related to this preference for barren habitat: abundance of Horned Larks differed across our study area as a function of both the density of roads and the variation in sizes of patches of wooded wetland. Size of interaction is estimated as 0.00163, significance threshold as 0.00085. In the shortgrass prairie region "wooded wedland" effectively means wooded areas along rivers and these are essentially the only large areas of taller vegetation in the entire region. We suspect that the variation in sizes of wooded patches per se is not important. Ex-

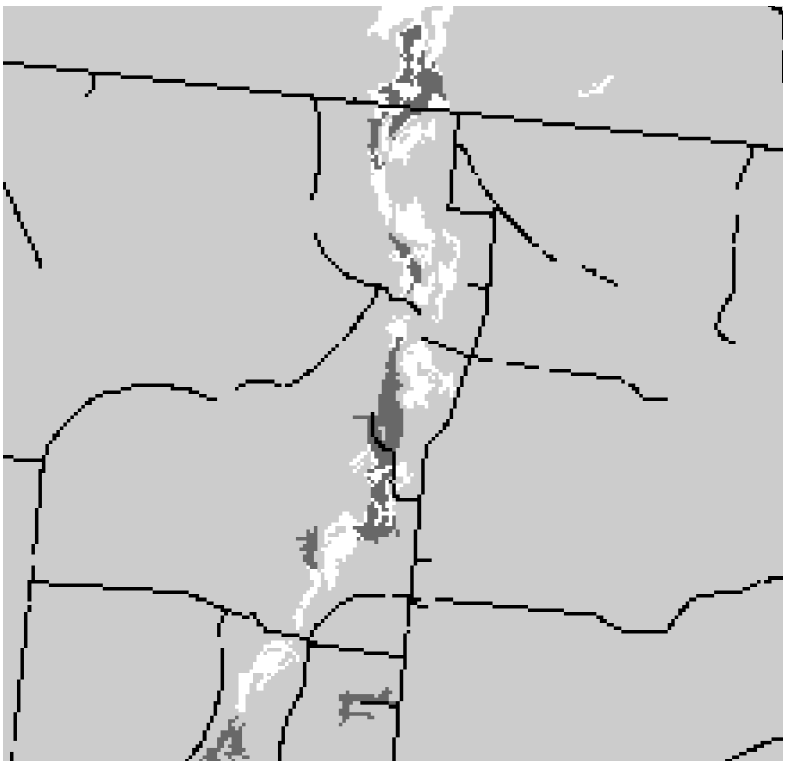

Figure 7.4: Habitat of Horned Larks. Color coded NLCD layers: black — developed, open space (roads); dark grey — wooded wetlands; light grey — grassland; white — water.

amination of the full set of features indicates that increased total area in a habitat is associated with increased variation in patch sizes: A large range of variation in patch sizes can only exist when at least some large patches are present. Thus, greater variation in size of wooded patches is related to a broader distribution of trees in the overall region and a greater fragmentation of the open habitat that the larks prefer. Figure 7.3 shows that there is a sharp drop in abundance of Horned Larks as soon as there is any substantial amount of wooded wetland habitat. Horned Larks do not like wooded habitat. However, the effect of woodland was ameliorated by the presence of roads, with more Horned Larks present, even in areas with higher amounts of forest, when these regions had a higher density of roads. Effectively, the roads create open areas of habitat preferred by Horned Larks. Figure 7.4 shows a representative example of the distribution of habitat

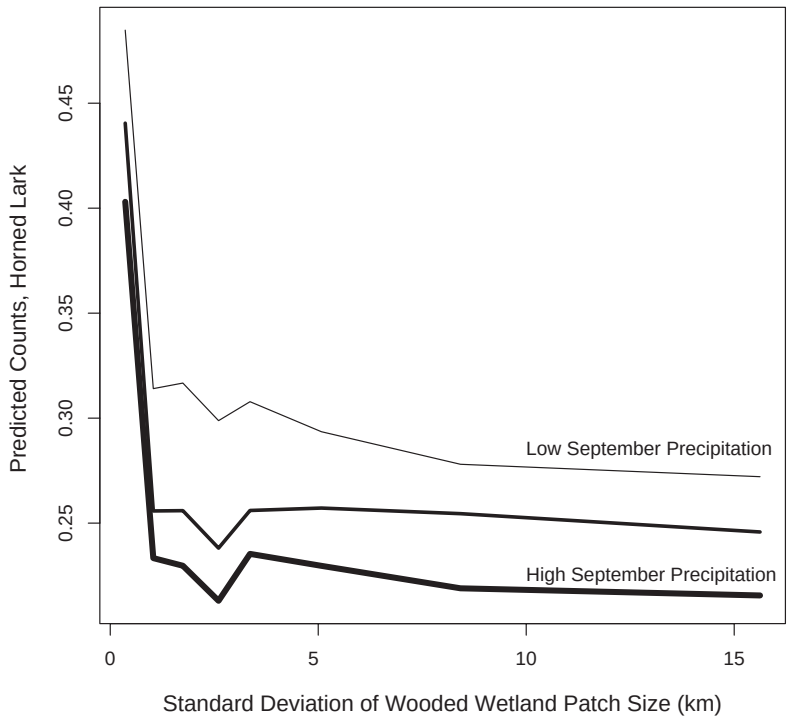

Figure 7.5: Horned Lark. Interaction between variation in sizes of patches of wooded wetlands and precipitation in September

types in an area of lark habitat (grassland) in which wooded wetlands and roads are also present in relatively high densities. Detecting this interaction has helped us identify an unexpected impact of human modification of landscape.

Another interaction identified affecting the abundances of Horned Larks was between variation in size of wooded wetlands (again, likely indicating the prevalence of forest in general) and multi-year average precipitation within the same region (Figure 7.5). Size of interaction is estimated as 0.00091, significance threshold as 0.00085 September precipitation amounts were identified as having high predictive importance in our analyses. Again, Horned Larks are less abundant in areas containing any substantial amount of forest habitat. However, this effect was lessened in parts of the shortgrass prairie that received less rainfall in September (and also throughout the rest of the year). This also is consistent with our knowledge of the species, which prefers more arid areas with very short vegetation, and lower

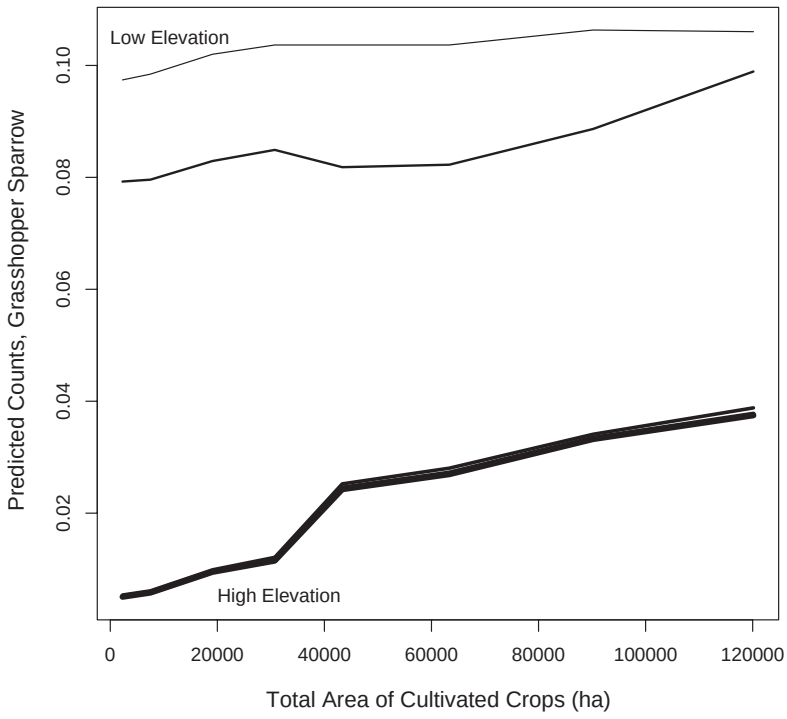

Figure 7.6: Grasshopper Sparrow. Interaction between elevation and total area of cultivated crops

amounts of precipitation would allow such habitats to exist even relatively close to the forested habitats along watercourses.

Grasshopper Sparrows (*Ammodramus savannarum*) are a species that lives in moderately lush grassland habitat (by the standards of the shortgrass prairie region), an effect that we believe is indicated by the sharp drop in abundance of this sparrow at higher elevations: Drier sites are closer to the rain shadow of the Rocky Mountains. Figure 7.6 shows a threshold-like effect; note that three separate partial-dependence prediction lines are essentially overlapping at higher elevations. However, the elevation effect was eased by the presence of cultivated crops at higher-elevation sites within the grasslands. Size of this interaction was estimated as 0.00223, significance threshold as 0.00093. We suspect that this unanticipated finding results from the presence of artificial water sources, irrigating the cropland, creating habitat that was more suitable for Grasshopper Sparrows. Again, inter-

action detection provided us with evidence that human modification of landscapes affected their suitability to birds, allowing Grasshopper Sparrows to live in areas that would be unsuitable for them under natural conditions.

The last figure shows a case in which our analysis determined absence or insignificance of an interaction. This is the result of the analysis for Red-winged Blackbirds (*Agelaius phoeniceus*), in which two habitat-related influences are almost independent of each other (Figure 7.7). These blackbirds are inhabitants of marshlands and may be associated with cultivated crops, both as areas for nesting as well as for feeding. Thus, the analysis presents no surprises with the abundance of Red-winged Blackbirds increasing both as the density of patches of open water increase and as the proportion of cultivated crops increase. In the shortgrass prairies, areas under cultivation may provide some of the most suitable habitat for this species.

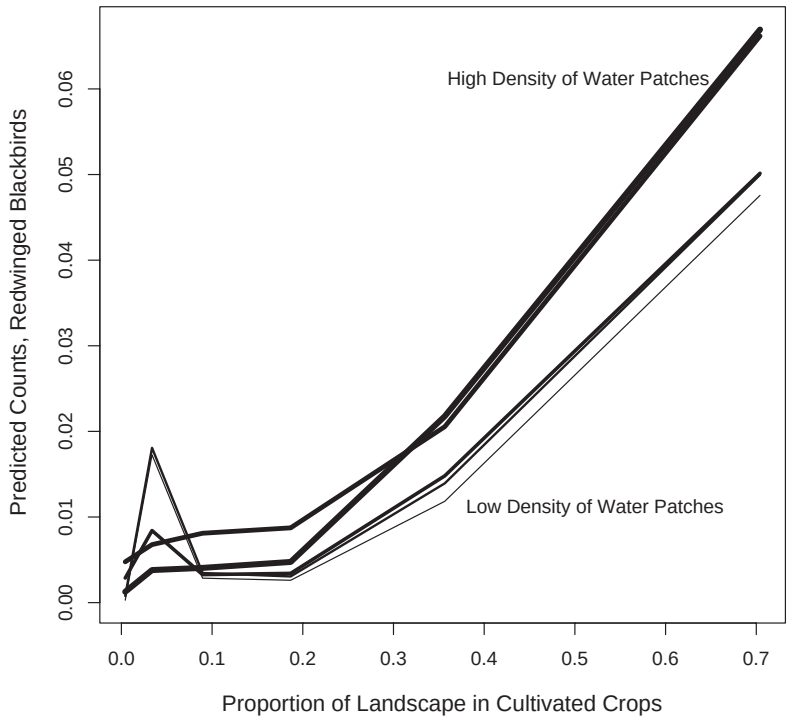

Figure 7.7: Red-winged Blackbird. Very low interaction between density of patches of open water and the proportion of the landscape in cultivated crops

Although the original interaction detection technique allows detection of higher-order interactions, we did not have an opportunity to conduct these tests for RMBO data sets. K-way interactions are possible only between those groups of variables that are involved in all possible $K(K-1)/2$ 2-way interactions between each other [26]. Such cliques of pairwise interactions never appeared during our analysis.

All interactions detected in RMBO data were very small. For comparison, most interactions in data sets described in Section 5.5 are larger by an order or two. This is a consequence of the fact that the data is noisy and difficult to model; small sized of interactions tell us that we can't improve performance much over the restricted models. However, as long as these small improvements are significant, they clearly indicate a presence of a real interaction in the data and in the domain.

7.5 Discussion

We have applied the process of interaction detection to noisy ecological data. We discussed several potential problems that can arise with this kind of real data, proposed possible solutions and presented the real results of applying this analysis to the data.

We have two general observations about the interactions that were detected in our analyses. First, the interactions were relatively subtle in nature, aside from the one identified for Lark Buntings (Figure 7.2). By this we mean that the effect of variation in one feature was only moderately altered by variation in the second feature in the interactions, as seen by the nearly parallel natures of the lines in the figures. This suggests that the analyses were able to detect modest interactions well. Our second observation is that most of the habitat types involved in the interactions were relatively uncommon in the shortgrass prairie region. For example, 99% of the areas around individual sites were composed of less than 4%

open water and less than 3% wooded wetlands. Two other habitat types were highly patchily distributed, with a median percentage of less than 2.5% (but a maximum in excess of 80%), and a median amount of cultivated crops on less than 18%, although some local areas had roughly 80% of their areas in cultivated crops. Thus, the interactions that we detected are describing biological phenomena that are occurring around only a small proportion of the sites. Again, this highlights the sensitivity of the interaction detection algorithm.

APPENDIX A

CONFIGURATIONS OF RMBO FEATURES

Table A.1 shows spatial extents at which data were used to describe habitat configuration.

Table A.1: Spatial extents.

Size of Area	GIS-layer Pixels
0.03×0.03 km	individual pixel
0.45×0.45 km	15 pixels across
1.5×1.5 km	50 pixels across
4.5×4.5 km	150 pixels across
15×15 km	500 pixels across
45×45 km	1500 pixels across

We describe spatial extents both in terms of the actual areas examined, and as a function of the number of pixels from which the configuration information was obtained. At the smallest scale, the individual GIS-layer pixel, only the identity of the habitat type could be obtained. At all larger scales, various metrics of habitat configuration were calculated.

Table A.2 presents habitat configuration metrics entered into the models.

Table A.2: Habitat configuration metrics.

Feature Description	Feature Name
Individual habitat	
Total Area	CA
Median Patch Size	$AREA\_MD$
Standard Deviation	$AREA\_SD$
Patch Density	PD
Patch Shape	LSI
Median Inter-patch Distance	ENN_MD
Edge Density	ED
All habitat	
Simpson's Diversity Index	$SIDI$
Median Fractal Dimensionality	$FRAC\_MD$
St. Dev. of Fractal Dimensionality	$FRAC\_SD$

We note both a brief description of the metrics, as well as the abbreviated names under which they are calculated in the FRAGSTATS software. Metrics were of two types. First, we used 7 different descriptions of configuration that were calculated

separately for each of the habitat types represented within a given spatial extent. Thus, for example with 4 habitat types, 28 such features would have non-zero values for a given data point. Second, we calculated 3 other metrics of configuration that summarized information across all habitat types present in an area. These latter three provide information on the overall complexity of arrangement and diversity of habitat types within an area. For descriptions of these metrics and their calculation, see the documentation for the software that was used to calculate the metrics at `www.epa.gov/mrlc/nlcd-2001.html`.

BIBLIOGRAPHY

[1] 2001 National Land Cover Data (NLCD 2001). www.epa.gov/mrlc/nlcd-2001.html.

[2] Avian Knowledge Network. www.avianknowledge.net.

[3] Rocky Mountains Bird Observatory. www.rmbo.org.

[4] M. Ayer, H. Brunk, G. Ewing, W. Reid, and E. Silverman. An Empirical Distribution Function for Sampling with Incomplete Information. *Annals of Mathematical Statistics*, 5:641–647, 1955.

[5] E. Bauer and R. Kohavi. An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. *Machine Learning*, 36:105–139, 1999.

[6] L. Breiman. Bagging Predictors. *Machine Learning*, 24:123–140, 1996.

[7] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[8] S. T. Buckland, D. R. Anderson, K. P. Burnham, J. L Laake, D. L. Borchers, and L. Thomas. *Introduction to Distance Sampling*. Oxford University Press, 2001.

[9] T. Bylander. Estimating Generalization Error on Two-Class Datasets Using Out-of-Bag Estimates. *Machine Learning*, 48:287–297, 2002.

[10] R. Camacho. Inducing Models of Human Control Skills. In *Proceedings of European Conference on Machine Learning (ECML'98)*, 1998.

[11] R. Caruana, M. Elhawary, D. Fink, W. M. Hochachka, S. Kelling, A. Munson, M. Riedewald, and D. Sorokina. Mining Citizen Science Data to Predict Prevalence of Wild Bird Species. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD'06)*, 2006.

[12] R. Caruana and A. Niculescu-Mizil. Data Mining in Metric Space: an Empirical Analysis of Supervised Learning Performance Criteria. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD'04)*, 2004.

[13] R. Caruana and A. Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of International Conference on Machine Learning (ICML'06)*, 2006.

[14] H. Chipman, E. George, and R. McCulloch. Bayesian Ensemble Learning. In *Proceedings of Neural Information Processing Systems Conference (NIPS'07)*, 2007.

[15] R. Christensen. *Plane Answers to Complex Questions, The Theory of Linear Models*. Springer, 2 edition, 1996.

[16] M. Collins, R. E. Schapire, and Y. Singer. Logistic Regression, AdaBoost and Bregman Distances. *Machine Learning, Special Issue on New Methods for Model Selection and Model Combination*, 48:253–285, 2002.

[17] P. Domingos. Bayesian Averaging of Classifiers and the Overfitting Problem. In *Proceedings of International Conference on Machine Learning (ICML'00)*, 2000.

[18] J. Friedman. Stochastic Gradient Boosting. *Computational Statistics and Data Analysis*, 38:367–378, 2002.

[19] J. Friedman, T. Hastie, and R. Tibshirani. Additive Logistic Regression: a Statistical View of Boosting. *Annals of Statistics*, 38:337–374, 2000.

[20] J. H. Friedman and B. E. Popescu. Predictive Learning via Rule Ensembles. Technical report, Stanford University, 2005.

[21] J.H. Friedman. RuleFit with R. www-stat.stanford.edu/~jhf/R-RuleFit.html.

[22] C. Gu. *Smoothing Spline ANOVA Models*. Springer, 2002.

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

[24] R. S. Hames, K. V. Rosenberg, J. D. Lowe, S. E. Barker, and A. A. Dhondt. Adverse Effects of Acid Rain on the Distribution of the Wood Thrush Hylocichla Mustelina in North America. *Proceedings of the National Academy of Sciences of the USA*, 99:11235–11240, 2002.

[25] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[26] G. Hooker. Discovering ANOVA Structure in Black Box Functions. In *Proceedings of International Conference on Knowledge Discovery and Data Mining (KDD'04)*, 2004.

[27] G. Hooker. Generalized Functional ANOVA Diagnostics for High Dimensional Functions of Dependent Variables. *Journal of Computational and Graphical Statistics*, 16:709–732, 2007.

[28] J. Friedman. Greedy Function Approximation: a Gradient Boosting Machine. *Annals of Statistics*, 29:1189–1232, 2001.

[29] A. Jakulin and I. Bratko. Testing the Significance of Attribute Interactions. In *Proceedings of International Conference on Machine Learning (ICML'04)*, 2004.

[30] K. Kira and L. Rendell. The Feature Selection Problem: Traditional Methods and a New Algorithm. In *Proceedings of Conference on Artificial Intelligence (AAAI'92)*, 1992.

[31] R. Kohavi and G. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97:273–324, 1997.

[32] E. L. Lehmann. *Nonparametrics: Statistical Methods Based on Ranks*. Chapman & Hall/CRC, 1989.

[33] O. Linton and J. P. Nielsen. A Kernel Method of Estimating Structured Nonparametric Regression Based on Marginal Integration. *Biometrika*, 82:337–374, 1995.

[34] K. McGarigal, S. A. Cushman, M. C. Neel, and E. Ene. FRAGSTATS: Spatial Pattern Analysis Program for Categorical Maps. www.umass.edu/landeco/research/fragstats/fragstats.html, 2002.

[35] M. Meyer and P. Vlachos. StatLib. Department of Statistics at Carnegie Mellon University. lib.stat.cmu.edu.

[36] A. Niculescu-Mizil and R. Caruana. Obtaining Calibrated Probabilities from Boosting. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI'05)*, 2005.

[37] A. Niculescu-Mizil and R. Caruana. Predicting Good Probabilities with Su-

pervised Learning. In *Proceedings of International Conference on Machine Learning (ICML'05)*, 2005.

[38] R. K. Pace and R. Barry. Sparse Spatial Autoregressions. *Statistics and Probability Letters*, 33:291–297, 1997.

[39] B. G. Peterjohn and J. R. Sauer. *Ecology and Conservation of Grassland Birds of the Western Hemisphere*. Cooper Ornithological Society Studies in Avian Biology, 1999.

[40] J. Platt. Probabilistic Outputs for Support Vector Machines and Comparison to Regularized Likelihood Methods. *Advances in Large Margin Classifiers*, pages 61–74, 1999.

[41] F. Provost and T. Fawcett. Robust Classification for Imprecise Environments. *Machine Learning*, 42:203–231, 2001.

[42] C. E. Rasmussen, R. M. Neal, G. Hinton, D. van Camp, M. Revow, Z. Ghahramani, R. Kustra, and R. Tibshirani. Delve. University of Toronto. www.cs.toronto.edu/~delve.

[43] S. K. Robinson, F. R. Thompson, T. M. Donovan, D. R. Whitehead, and J. Faaborg. Regional Forest Fragmentation and the Nesting Success of Migratory Birds. *Science*, 267:1987–1990, 1995.

[44] D. Ruppert, M. P. Wand, and R. J. Carroll. *Semiparametric Regression*. Cambridge, 2003.

[45] R. E. Schapire. The Boosting Approach to Machine Learning: An Overview. *Nonlinear Estimation and Classification*, 2003.

[46] T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, 1990.

[47] Luís Torgo. Regression DataSets. www.liacc.up.pt/~ltorgo/Regression/DataSets.html.

[48] M. Winter, D. H. Johnson, J. A. Shaffer, T. M. Donovan, and W. D. Svedarsky. Patch Size and Landscape Effects on Density and Nesting Success of Grassland Birds. *Journal of Wildlife Management*, 70:158–172, 2006.

[49] D. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.

[50] S. N. Wood. *Generalized Additive Models: An Introduction with R.* Chapman & Hall/CRC, 2006.

[51] B. Zadrozny and C. Elkan. Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers. In *Proceedings of International Conference on Machine Learning (ICML'01)*, 2001.

[52] B. Zenko, L. Todorovski, and S. Dzeroski. A Comparison of Stacking with MDTs to Bagging, Boosting, and other Stacking Methods. In *Proceedings of International Conference on Data Mining (ICDM'01)*, 2001.