A Look at Modern Classification Algorithms for Structural Data

HIOF University Colledge

Freddy Christoffer Suther & Frank Daniel Suther

May 15, 2018

Abstract

We tested 108 classification algorithms from caret and WEKA. The classification algorithms were tested on 10 fold cross-validation repeated 100 times, on 89 datasets. The algorithms were tuned once per dataset, with a manual tunegrid. We used three metrics, accuracy, kappa and Friedman rank. The best algorithm, catboost, got an average accuracy of 83.96%, only 1.5 percentage point behind the maximum accuracy(the average of the best average result for each dataset across all algorithms) of 85.5%. The catboost algorithm overcomes 90% of the maximum accuracy on every single dataset, and overcomes 95% of the maximum accuracy in 92% of the datasets. The catboost algorithm achieved on average 98.15% of the maximum accuracy over all the datasets. There is no statistically significant difference between catboost and xgbTree, and catboost and parRF, although parRF is close to be significantly different from catboost. The boosting ensembles, tree ensembles, combined classifiers and SVMs are significantly better than the other families(bayesian classifiers, discriminant analysis, classification trees, rule-based models and instance based models). Across the 89 datasets, we have 46 different algorithms being the best, many of which are shared positions . Most of the top classification algorithms are from the caret package, with the top 7 coming from caret, and 8/10 best using caret, all tuned with manual tunegrids, WEKA is not far behind however, with 9 of the top 20 on Friedman Rank, and 10 of the top 20 on accuracy.

Keywords: machine learning, classification, boosting, bagging, support vector machines, neural networks, automatic machine learning.

Declaration

We have no horse in this race.

Acknowledgements

We would like to thank our thesis supervisor, Dr. Roland Olsson for his support and advice.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction and Motivation

Machine learning can be defined as the ability of making a machine learn without explicitly programming them. Tom M. Mitchell defined learning as "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [1].

A few state of the art achievements may provide a glimpse into the possibilities of the field: programs which can successfully learn to recognize spoken words were made in 1989 [2], predict recovery rates of pneumonia patients in 1997 [3], predict Airfare prices in 2017 [4]. Machine Learning can be split into two directions, supervised learning, and unsupervised learning. Supervised learning can be split into three directions, Classification, Regression and Reinforcement Learning. We want to take a look at classification algorithms.

What can be considered the best classification algorithm? Wolpert's No Free Lunch Theorem states that there is no one model that works best for every problem [5]. While this is true, it does not stop us from considering one method generally better than the others. The search for the best classification algorithms is useful for two groups, the researchers and for those who practice classification. Researchers will typically benefit from knowing what kind of family the best algorithms come from, in order to know what to focus on when they develop their next algorithm. E.g. if a group of researchers want to create the new best algorithm it is more useful to try to create an improved version of a boosted tree than a rule-based model. For those who practice classification, through the data science website Kaggle [6] or at work, this research can be more important, because our results show the potential of multiple algorithms, and the strengths of some not so common ensembles.

1.2 Research Question

This thesis aims to compare classification algorithms for structured data. The classification algorithms are measured by the metrics accuracy and kappa, and the Friedman Rank. They will be compared against every other classification algorithm as well as every other classification algorithm within their family, using paired T-tests, heat maps, and compared to the maximum accuracy achieved across classification algorithms.

The goal of this research was to answer the following research question:

RQ 1: What are the best classification algorithms?

Secondary relevant research questions are:
RQ 1.1: What do the better classification algorithms have in common, are they of the same family, or do they share other characteristics?
RQ 1.2: Does Wolpert's No Free Lunch Theorem still hold true?
RQ 1.3: How does Auto-WEKA perform compared to the other classification algorithms?
RQ 1.4: For the bagging ensembles, what are the effects of bagging compared to the base-classifiers?
RQ 1.5: Do our results differ from the results from similar research?

1.3 Thesis Outline

- In Chapter 2 we mention related work, and give a brief introduction to the different "families" of classification algorithms. The families are somewhat arbitrary, for instance the classification algorithm DTNB(decision table naive Bayes) can be considered both a bayesian model and a rule-based model, as it is a hybrid model. Chapter 2 also contains a list of each classification algorithm we used, and a brief description of them.

- In Chapter 3 we describe how the experiment was designed and performed. The datasets used are mentioned and a link to their location is mentioned. Different software, which versions of aforementioned software, scale of the experiment and hardware we used is mentioned. The chapter also contains a description of why we chose to run the experiment the way we did, and other information needed for replication of the experiment such as what seed we used etc..

- In Chapter 4 we describe our results. This includes the average accuracy, average kappa and Friedman rank of each classification algorithm. We also include a list of which classification algorithms were the best and the score of the best on each dataset.

- In Chapter 5 we discuss our findings, compare our results to similar research and answer the Research Questions.
- In Chapter 6 contains our conclusion, where we sum up our results and what we have done.

Chapter 2

Background

2.1 Related work

In 1999, Eric Bauer and Ron Kohavi had a large-scale comparison of Bagging, Bagging variants, AdaBoost, and Arc-x4 on two families of induction algorithm, decision trees and Naive-Bayes. The boosting algorithms were generally better than Bagging, but not uniformly better. They looked at how boosting and bagging would reduce the error the induction algorithms had, the best algorithm AdaBoost caused an error reduction of 27% for MC4, 31% for MC4(1)-disc, and 24% for Naive-Bayes. [7].

In 2000, Tjen-Sien Lim, Wei-Yin Loh & Yu-Shan Shih compared twenty-two decision trees, nine statistical and two neural network algorithms on 33 datasets in terms of classification accuracy, training time and number of leaves(decision trees only). Classification accuracy was measured by mean error rate and mean rank of error rate. The statistical algorithm POLYCLASS was the best in both criteria. The second best algorithm was logistic regression. Best decision tree algorithm was QUEST with linear splits. The POLYCLASS algorithm had the third longest training time, and was significantly slower than the QUEST algorithm(3.2 hours vs 5.9 minutes) [8].

In 2005, Olcay Taner Yildiz and Ethem Alpaydin tested seven different Decision Tree algorithms(C4.5, Classification And Regression Tree, ID-LP, Linear Machine Decision Tree, Oblique Classifier, Quick Unbiased Efficient Statistical Tree and Linear Discriminant Tree) on 20 datasets from the UCI database. The algorithms were tested on percentage correctly classified instances, learning times and tree sizes by the number of nodes. The most accurate algorithms were ID-LP(Neural Tree with a linear perceptron), Linear Machine Decision Tree, C4.5, Linear Discriminant Tree and Quick Unbiased Efficient Statistical Tree being nearly equal. The fastest algorithm was C4.5 and Linear Discriminant Tree. The tree size of C4.5 was the smallest with LDT and ID-LP as number 2 and 3 [9].

In a paper by Xindong Wu et. al from 2007 presents the top 10 most influential classification algorithms in data mining, according to a panel of 145 attendees on the International Conference of Data Mining. The top 10 was C4.5, k-Means, Support Vector Machine, Apriori, Expectation Maximization, PageRank, AdaBoost, kNN, Naive Bayes and Classification and Regression Tree [10].

In 2012, Gouda I. Salama, M. B. Abdelhalim and Magdy Abd-elghany Zeid compared five different classification algorithms from WEKA on three different databases of breast cancer, Wisconsin Breast Cancer(WBC), Wisconsin Diagnosis Breast Cancer(WDBC) and Wisconsin Prognosis Breast Cancer(WPBC). The classification algorithms were J48 classification tree, Multilayer Perceptron(MLP), Naive Bayes(NB), Sequential Minimal Optimization(SMO) and Instanced Based for K-Nearest neighbour(IBK). They also combined the models using voting to determine the class. A combination of MLP and J48 with feature selection was superior to the other classifiers on the WBC database. On the WDBC database SMO was superior and on the WPBC database an ensemble of J48, SMO, MLP and IBK was the best classifier [11].

In 2013, K. Wisaeng presented a comparison of different classification techniques on a set of bank direct marketing dataset. The classification algorithms tested were J48-grafted decision tree, Logical Analysis of Data tree(LAD Tree, Radial Basis Function Network, and LibSVM, all from WEKA. K. Wisaeng tested the algorithms on Sensitivity $\frac{TP}{TP+FN}$, Specificity $\frac{TP}{TP+FP}$ and accuracy(TP= true positive, FP= false positive and FN= false negative). LibSVM got the best results with a Sensitivity of 87%, a Specificity of 86.70% and an accuracy of 86.95%. J48-graft got a Sensitivity of 76.5%, a Specificity of 78.6% and an accuracy of 76.52%. LAD Tree got a Sensitivity of 76.1%, a Specificity of 75% and a accuracy of 76.08%. The worst results came from the RBF Network with a Sensitivity of 74.3%, Specificity of 73.5% and accuracy 74.34% [12].

In 2014, Diego Raphael Amancio et.al. compared nine classification algorithms from WEKA, Naive Bayes(NB), Bayesian Network(BN), C4.5, Random Forest(RF), Simple Classification and Regression Tree(CART), k-Nearest Neighbors(kNN), Logistic(Log), Multilayer Perceptron(MLP) and Support Vector Machine(SVM) on an artificial dataset. They also looked at the effects of parameter tuning for these classification algorithms. Without parameter tuning the kNN algorithm performed better than the rest. SVM performed worse than the rest with default parameters but was significantly stronger when tuned. When they increased the number of attributes in the dataset kNN and SVM performed better than the rest with and without tuning [13].

Also in 2014, Manuel Fernández-Delgado, Eva Cernadas and Senén Barro tested 179 classifiers from 17 families(Bagging, Bayesian, Boosting, Decision Trees, Discriminant Analysis, Generalized Linear Models, Logistic and Multinomial Regression, Multiple Adaptive Regression Splines, Nearest Neighbor, Neural

Networks, Partial Least Squares, Principal Component Regression, Random Forest, Rule-based classifiers, Stacking, Suport Vector Machines and other methods) implemented in WEKA, R, C and Matlab on 121 datasets from the UCI database. The Random Forest classifiers were most likely to be the best versions, with the best(implemented in R with the caret package) achieved 94.1% of the maximum accuracy overcoming 90% in 84.3% of the datasets. Three out of the five best classifiers were implementations of Random Forest, making it the best family of classifiers. The second best family was support vector machines with four classifiers in the top ten, and the second best classifier, Support Vector Machine with Gaussian kernel implemented in C using LibSVM, with 92.3% of the maximum accuracy [14].

In 2016, Jacques Wainer et.al. tested 14 different classification algorithms(random forest, gradient boosting machines, SVM-linear,polynomial, and RBF-1-hidden-layer neural nets, extreme learning machines, k-nearest neighbours and a bagging of knn, naive Bayes, learning vector quantization, elastic net logistic regression, sparse linear discriminant analysis and a boosting of linear classifiers) on 115 real life binary datasets. The algorithms' parameters were tuned on a subset of each dataset using five fold cross-validation. They found that the three best classifiers(random forest, gbm and support vector machine with radial basis function) were not significantly different from each other in terms of precision. They also verified the execution speed of standard implementations of these algorithms and found support vector machine with radial basis function to be the fastest among the three best, both in training time and in training plus testing time [15].

In 2017, Edwin Raczko and Bogdan Zagajewski compared three classification algorithms, Random Forest, Support Vector Machine and a Neural Network. The task was to classify trees in the northwesten part of the Karkonosze National Park in Poland, south of Szklarska Poreba. The National Park is located on the Polish-Czech border.The area contained five types of trees, spruce, larch, alder, beech and birch. The data was collected from two spectrometers which covered the spectral range of 380-2500 nanometers. The data was reduced from 288 spectral to 40. The data was split in a training set of 63.2% and a test set of 36.8%. The Support Vector Machine they chose was the ”svmRadial”, the Random Forest implementation was the ”rf”. The Artificial Neural Network is either ”nnet”, or a deep neural network, it is hard to say based on the information given in the journal. The ANN achieved the highest median overall classification accuracy with 77%, followed by SVM with 68% and RF with 62%. ANN had the highest median kappa at 0.72, SVM 0.61 and RF 0.52. All three algorithms had an easier time classifying Alder than the other treetypes. Edwin Raczko and Bogdan Zagajewski mention that the reduction in bands was made to lower the training time for ANN might have an impact on the result [16].

Since the Manuel Fernández-Delgado, Eva Cernadas and Senén Barro study, and the Jacques Wainer's follow-up study new algorithms have been made, the

Kaggle favorite Extreme Gradient Boosting(XGB) is known to do well(according to Anthony Goldbloom the, founder and CEO of Kaggle, who noted that after Random Forest was beaten by xgboost repeatedly "It used to be random forest that was the big winner, but over the last six months a new algorithm called XGboost has cropped up, and its winning practically every competition in the structured data category") [17], and two new boosting methods LightGBM [18] and catboost [19] have been published.

2.2 Supervised Learning & Classification

Supervised learning is something every human being has experienced, imagine being young and seeing a bonfire, what can we throw onto it and what effect will it have? There might be three outcomes, flames increase, flames are put out or flames increase rapidly, depending on what we throw on. We can create 3 classes for this example, objects that burn, objects that do not burn and objects which are inflammatory. With experience we learn that flasks with a skull on belongs in the inflammatory class. This is a very simple case of classification, we try to put an instance or object into a group based on its attributes. This is called supervised learning, because we have tried x objects before, and know their characteristics/attributes, and based on their attributes we create a model or a set of rules which can be used to predict new objects outcome. Classification is strictly postjudice, and is used to create a predictive model based data. Multiple "families" of classification algorithms exists, we chose to split them into 11 groups, artificial neural networks, Bayes classification methods, boosting ensembles, classification trees, discriminant analysis, instanced based learners,tree ensembles, rule-based models, support vector machines and misc classification algorithms.

2.2.1 Artificial Neural Networks

Artificial neural network models are created as very simplified models of the human brain. The human brain is composed of a large number(10^{11}) of processing units, or neurons, operating in parallel. Though the speed of the neurons in our brain is unknown, it is believed to be much slower than that of a computational processor unit. What is believed to provide the human brain its processing power is the synapses, or connections between neurons. A single neuron can have 10^4 synapses to other neurons, all operating in parallel [20].

The human brain's neuron is simulated in a neural network as the perceptron. The perceptron is the basic processing element, its input is gathered from the environment or from the output of other perceptrons. Each input(x_j) is given a weight w_j which makes the output function:

$$y = \sum_{j=1}^{d} w_j x_j + w_0$$

w_0 is the intercept value. It is the weight coming from a bias unit,x_0, which is always +1. If d=1 and x is given an input, we have the formula

$$y = wx + w_0$$

which is the formula for a line with w_0 is the intersect and w is the slope. So a perceptron with single input and single output can be used to create a linear discriminant. Nonlinear discriminant can be made with a Multilayer perceptron, where a hidden layer between the input and the output layer applies a function on the weighted sum of the outputs from the previous layer.

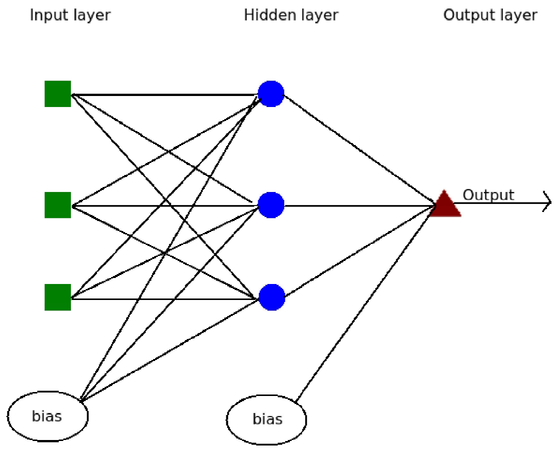


Figure 2.1: A Multilayer Perceptron.

2.2.1.1 Artificial Neural Network implementations

- **avNNet (R)** ”avNNet” is an implementation of an aggregated neural network from the nnet package, where the same neural network model is fit using different random number seeds, and the averaged result is translated to predicted classes. May use Bagging [21].
- **mlp (R)** ”mlp” is an implementation of a fully connected feedforward network from the RSNNS package [21].
- **mlpML** ”mlpML” is an implementation of a fully connected feedforward network from the RSNNS package, with three hidden layers [21].
- **mlpWeightDecay** ”mlpWeightDecay” is an implementation of a multilayer perceptron from the RSNNS package, which can be tuned on its decay parameter and size parameter, unlike ”mlp” which only has a size parameter [21].
- **nnet (R)** ”nnet” is an implementation of a single-hidden-layer neural network from the nnet package [21].

- **pcaNNet (R)** "pcaNNet" is an implementation of a multilayer perceptron from the nnet package, which is created after running principal component analysis on the dataset [21].
- **Elman (WEKA)** Builds a recurrent neural network with "context units" that stores the previous values of hidden units.
- **MultilayerPerceptron (WEKA)** A neural network that uses backpropagation to classify instances.

2.2.2 Automatic Classification

The goal for automated machine learning is to find the best algorithm and parameter setup for a given task, given its allotted time. Auto-WEKA searches through all classifiers and their parameters and returns the best classifier and attribute selector for your dataset. It uses the random-forest-based Bayesian Optimization algorithm SMAC to determine which classifier has the best performance on the given dataset [22]. Auto-WEKA is fully integrated in WEKA, with its own tab in the Explorer GUI.

2.2.2.1 Automatic Classification implementations

- **AutoWEKA** Set up to search for 45 minutes per dataset.

2.2.3 Bayes Classification methods

Bayesian classifiers are statistical classifiers. Bayesian classifiers can predict the class probability that a given instance belongs to a specific class. Bayesian classifiers are based on Bayes' theorem, named after the English statistician from the 18th century. Let $\boldsymbol{X}$ be a data tuple. In Bayesian terms, $\boldsymbol{X}$ is considered "evidence", described by measurements made on a set of n attributes. Let H be some hypothesis such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine $P(H|\boldsymbol{X})$, the probability that the hypothesis H holds given the "evidence" or observed data tuple $\boldsymbol{X}$. Or simply, we are looking for the probability that instance $\mathbf{X}$ belongs to class C, given the attributes of $\boldsymbol{X}$ [23].

$P(H|\boldsymbol{X})$ is the *a posteriori probability*, of H conditioned on $\boldsymbol{X}$. For instance, imagine a store having a dataset with information about their customers, attributes gender and annual income, and $\boldsymbol{X}$ is a female customer with an annual income of \$75 000. H is the hypothesis that a customer will buy a car. $P(H|\boldsymbol{X})$ is the probability that customer $\boldsymbol{X}$ will buy a car given that we know the customer's gender and annual income.

$P(H)$ is the *a priori probability*, of H. In the aforementioned instance this is the probability that a customer will buy a car. The posterior probability $P(H|\boldsymbol{X})$ is based on more information (here gender and annual income) than the prior probability, $P(H)$, which is independent of $\boldsymbol{X}$.

$P(X|\boldsymbol{H})$ is the a posteriori probability of $\boldsymbol{X}$ conditioned on H, the probability that a customer, $\boldsymbol{X}$, is female and earns $75 000, given that we know she will buy a car.
$P(\boldsymbol{X})$ is the prior probability of $\boldsymbol{X}$. In this instance it is the probability that a customer is female and earns $75 000. Bayes theorem shows how these probabilities are connected:

$$P(H|\boldsymbol{X}) = \frac{P(\boldsymbol{X}|H)P(H)}{P(\boldsymbol{X})}.$$

Bayesian classification methods use Bayes Theorem to classify instances. One of their main advantages is the runtime, and it has been shown to be comparable to decision trees in some domains [23].

2.2.3.1 Bayesian classification method implementations

- **bayesglm (R)** "bayesglm" is an implementation of a bayesian function for generalized linear modeling, from the arm package [24].
- **naive_bayes (R)** "naive_bayes" is an implementation of the naive bayes algorithm from the naivebayes package, which computes the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule [25].
- **nb (R)** "nb" is another implementation of naive bayes algorithm, from the klaR package.
- **A1DE (WEKA)** Builds an ensemble of one-dependence estimators, where one node is set as a parent to every other node, except the classnode.
- **Naive Bayes (WEKA)** WEKAs implementation of Naive Bayes.
- **Naive Bayes Updateable (WEKA)** An updateable version of Naive Bayes.

2.2.4 Classification Trees

A Classification Tree is a collection of decision nodes, connected by branches, extending downward from the root node until terminating in leaf nodes. The root node, displayed at the top of the decision tree diagram, creates leaf nodes by splitting the dataset in multiple subsets, split on an attributeś value. This split leads to a terminating leaf node or another decision node.

There are two kinds of decision trees, either ID3/C4.5/C5.0 or classification and regression trees(CART). CART creates binary trees, each split creates minimum

1 node, maximum 2 [26]. CART also uses gini impurity, $\sum_{k \neq i} P_k = 1 - p_i$ which is faster than the logarithmic Information Gain.

C4.5 is Quinlan's extension of his Identigraph Dichotomizer 3. Unlike CART, c4.5 is not strictly binary. For categorical(non numeric) attributes, c4.5 by default produces a separate branch for each value of the categorical attribute.

The split point and attribute the data is split in by is chosen by finding the highest Information Gain. Information Gain is found by comparing entropy before and after a split and the highest change in entropy gives you the highest information gain. The attribute and its value with the highest gain are marked in the top node and this process is repeated recursively for the left and right leafnode. This method is greedy, and thus might cause the model to be overfit. In order to escape overfitting c4.5 prunes the tree, removing leafnodes with a lower significance.

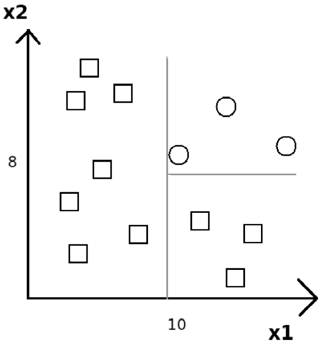


Figure 2.2: Figure shows a dataset which contains 2 classes, square and circle, attribute x1, attribute x2 and lines representing the split done by a decision tree.

2.2.4.1 Classification tree implementations

- **C5.0Tree (R)** "C5.0Tree" is an implementation of a classification tree model using Ross Quinlan's C5.0 algorithm, from the C50 and plyr packages [27].
- **ctree (R)** "ctree" is an implementation of a tree using recursive partitioning for continuous, censored, ordered, nominal and multivariate response variables in a conditional inference framework, from the party package [28].

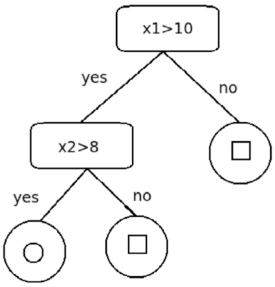


Figure 2.3: The decision tree created from Figure 2.2.

- **rpart (R)** ”rpart” is an implementation of a recursive and partitioning tree from the rpart package, following Breiman, Friedman, Olshen and Stone's 1984 book Classification and Regression Trees [29].
- **BFTree (WEKA)** A best-first decision tree classifier.
- **CDTree (WEKA)**A Decision Tree that is built on imprecise probabilities and uncertainty measures.
- **DecisionStump (WEKA)** Builds a decisionstump, usually used together with a boosting algorithm.
- **ExtraTree (WEKA)** Extremely randomised Tree, selects a random value for splitting, rather than by using gini/information gain.
- **FT (WEKA)** Builds trees with logistic regression functions at the inner nodes and/or leaves.
- **J48 (WEKA)** Generates a pruned or unpruned C4.5 decision tree.
- **J48consolidated (WEKA)** Class for generating a pruned or unpruned C45 consolidated tree.
- **J48graft (WEKA)** An extended version of j48 that may graft additional branches to the j48 tree during pruning, by considering different tests at the splits above the leaves that often miss-classify or never classify instances.
- **HoeffdingTree (WEKA)** Hoeffding trees are decision trees built on a small sample of the training set, it presupposes that the samples attribute values are quite similar to the full training sets.
- **NBtree (WEKA)** Generates a decision tree with naive bayes classifiers as leaves.

- **PCT (WEKA)** Probability calibration trees, a tree implementation of ZeroR (zero rules).
- **RandomTree (WEKA)** A tree that considers K randomly chosen attributes at each node with no pruning.
- **REPTree (WEKA)** Builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with backfitting).
- **SimpleCART (WEKA)** Builds a decision tree with minimal cost-complexity pruning.

2.2.5 Discriminant Analysis

Discriminant Analysis is a technique used to predict a categorical response variable. In Discriminant Analysis it is assumed that the independent variables follow a multivariate normal distribution. The Discriminant Analysis algorithms attempt to determine several discriminant functions, linear combinations of the independent variables, to separate the groups defined by the response variable [30].

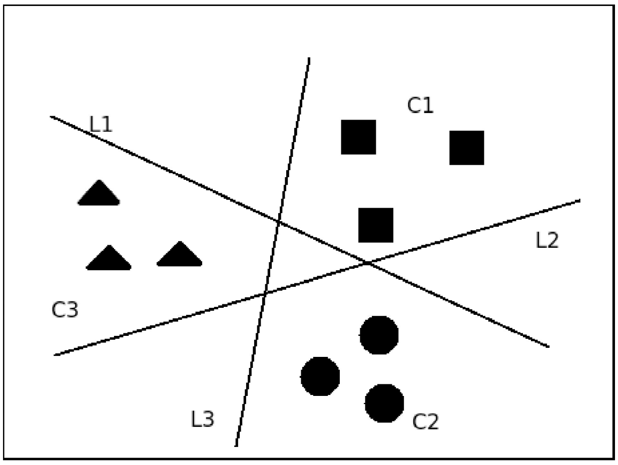


Figure 2.4: Image shows 3 linear discriminants L1,L2 and L3 and the corresponding classes C1,C2 and C3.

The simplest form of Discriminant Analysis is a linear discriminant. A linear discriminant algorithm creates a discriminant function which separates a class from the rest. E.g. on a dataset for vehicles class=car if g(x)>0, and not car if its below or equal 0. This assumes all classes are linearly separable, which may not be the case. If the classes are not linearly separable we can split into pairwise separation, which creates several smaller linear models, but the combined model is no longer linear [31].

2.2.5.1 Discriminant Analysis implementations

- **fda (R)** "fda" is an implementation of flexible discriminant analysis, from the earth and mda packages, using multivariate adaptive regression splines basis functions.
- **lda (R)** "lda" is an implementation of the linear discriminant analysis algorithm, from the MASS package.
- **pda (R)** "pda" is an implementation of penalized discriminant analysis, from the mda package.
- **rda (R)** "rda" is an implementation of regularized discriminant analysis, from the klaR package.
- **RFlda (R)** "RFlda" finds the coefficients of a linear discriminant rule based on a correlation (or covariance) matrix estimator that tries to approximate the true correlation (covariance) by the closest (according to a Frobenius norm) correlation (covariance) compatible with a q-factor model, from the HiDimDA package [32].
- **rlda (R)** "rlda" is an implementation of regularized linear discriminant analysis, from the sparsediscrim package.
- **sda (R)** "sda" is an implementation of shrinkage discriminant analysis, from the sda package.
- **slda (R)** "slda" is an implementation of the stabilized linear analysis algorithm, from the ipred package, based on left-spherically distributed linear scores.
- **LDA (WEKA)** Builds a Linear Discriminant Analysis model.
- **QDA (WEKA)** Builds a Quadratic Discriminant Analysis model.

2.2.6 Ensemble methods

Ensemble methods come from the idea that two heads think better than one. The basic idea behind ensemble methods is that having lots of learners that each get slightly different results on the dataset, some learning certain things well and some learning others, and then putting them together, the resulting model is significantly better than any of them on their own. Using ensemble methods we can split the training data into multiple different training sets, without negatively affecting the precision of the model [33]. Bagging, Boosting and Stacking are ways to include multiple models to a strong combined model, with astonishingly good performance [34]. Ensemble methods do not suffer the negatives human committees do. Human committees are poor at handling noise distractions, but noise may improve bagging models. Human committees filled with low skilled people will probably never come up with a significant working idea, but boosting can combine multiple weak model to a very strong model [34].

2.2.6.1 Bagging

Bagging is an ensemble method where multiple base-learners are created with their own slightly different dataset. These datasets are Bootstrap aggregations, that is randomly selected samples of the original dataset are taken with replacement, which will create multiple, often 50+, datasets with similar but not equal data. Bagging ensembles classify by voting, e.g. if we have 100 different models, 40 say class is 'A', 41 say 'B' and 19 say 'C' the bagging ensemble will select class 'B'. The models are created in parallel and are independent from each other. Bagging is essentially a democracy for learners, and the success of the ensemble is dependent on the strength of the base-learner [35]. The results of a bagging ensemble is based on the accuracy and stability of the base learning algorithm, bagging works best when used with an unstable yet accurate base learner(learners with a random element such as RandomTree and ExtraTree can be considered unstable), stable base learners will see little benefit to bagging [36].

2.2.6.2 Boosting

Boosting is an ensemble method where multiple weak learners are created in sequence, with each learner is built on the errors of the former learners and is given a weight to reflect when it is necessary when testing. Unlike bagging, where the strength of the ensemble is based entirely on the strength of the classifier, boosting does not require the classifier to be good, multiple weak learners,weak being slightly better than chance, can correct the former learners mistakes and together create a strong ensemble. Each learner is created in sequence, and their weakness is passed on to the next learner to improve where the former is performing poorly [37].

2.2.6.3 Combined Classifiers

Combined classifiers trains algorithms to work together to combine their predictions in some way, either by voting, by picking the best classifier from the base classifiers, by using meta classifiers to weight them or use the other learners predictions in some ways. Unlike bagging and boosting, combiners usually stack different types of models, i.e. you can combine instance based learners with naive bayes learners and/or decision trees. The models may be combined by simple voting, but if the models do not perform comparably well they may end up predicting wrong even if one or some of the other learners predict correctly.

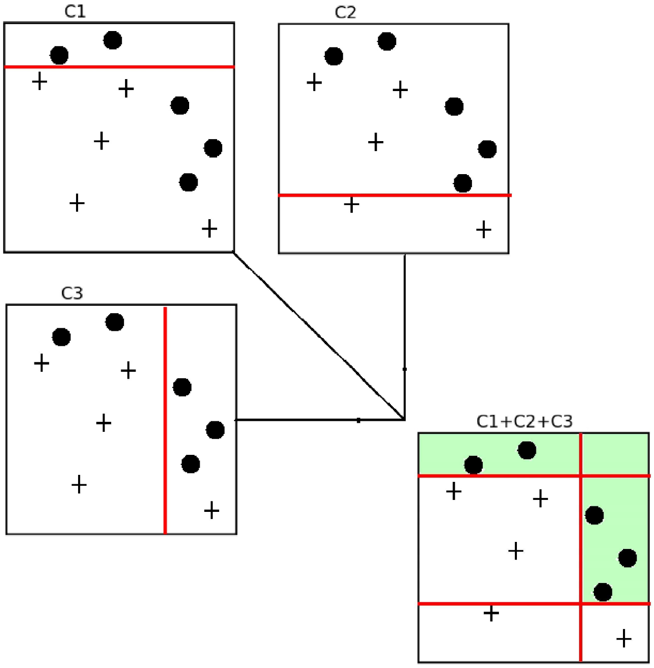


Figure 2.5: Image shows 3 weak classifiers C1,C2 and C3. Individually they perform poorly, but combined they create a strong classifier.

Stacking
Stacking adds another learner to the group, the meta classifier, which tries to learn how to combine the output of the others in the best possible way. The input to the meta classifier will be the predictions of the base classifiers, and the output will be its prediction based on the input. If the most precise predictor is chosen, the ensemble may learn to prefer to use base classifier that overfit the training data. One way to solve this is to withhold some data from the base classifiers, so that their predictions will be unbiased, but they will also get less data to work with. A different approach is to use k-fold cross-validation, so that the meta classifier can use the full dataset for learning. A different approach is to let the base classifiers output probabilities for each class given an instance, instead of predicting the class. The meta learner can now consider the base learners level of confidence before making its prediction [39].

Cascade Generalization
Cascade Generalization is a stacking method that uses its base classifiers in sequence, extending the dataset by creating and adding new attributes at each step. The new attributes are created by the class probabilities given by the previous learner [40].

Grading
Grading tries to identify and correct its base classifiers incorrect predictions.

MultiScheme
MultiScheme runs each of its base classifiers through x folds, and picks the classifier that performs the best on the current dataset.

Vote
A simple collection of classifiers that uses its probability classifiers estimates to predict classes. May also use majority vote.

One downside to combining different learners is that it can be difficult to tune all the parameters of all the different base learners, i.e. with five classifiers with an average of three parameters each will give you at minimum 32768(2^{15}) different combinations, even more if the stacking algorithm itself offers some parameters.

2.2.6.4 Tree Ensembles

RandomCommittee
A RandomCommittee is an ensemble of randomizable base classifiers, like RandomTree or ExtraTree, that are built with a different random number seed. The committee makes predictions by taking the average value of its base classifiers predicted values.

Random Forest

Random Forest is an ensemble learning method created by Leo Breiman [38].Random Forest consists of multiple Random Trees, which are created on randomly selected bootstraps of the dataset and the mode is used to determine the class. For a dataset with X objects and M attributes, N random trees are created with a subset with x(x<X) objects. For each of the N random trees, m(m<M) random attributes are chosen, and used to split the subset of data, creating a fully grown classification tree. When new objects are to be classified, the class is voted on by taking the mode(the value that occurs most often) of all the random classification trees [38].

1. 1. For b = 1 to B
    - Select random subset from training data.
    - Grow a random tree T(b) with the subset by recursively repeating the following steps:
        - Select m variables at random from the M variables.
        - Select the best variable for splitting.
        - Split the node in two leafnodes.
    - Output the ensemble of trees.

At each split a subset of randomly selected attributes are considered. The most computation intensive part of classification trees is calculation on the split node, which attribute to split on and at what value. Random Tree only considers a random subset of the attributes ($\sqrt{(}2n+1)$), and is used on a subset of the dataset, which makes Random Tree significantly faster than other classification tree methods.

RandomSubSpace
RandomSubSpace is an ensemble method that trains its base classifiers on randomly chosen subsets of the input attributes. It can be combined with bagging to randomize the datasets attributes and instances.

2.2.6.5 Ensemble Methods

- **Boosting**
    - **C5.0 (R)** ”C5.0” is an implementation of a boosting ensemble of classification tree models or rule-based models using Ross Quinlan’s C5.0 algorithm [27]. Requires the C50 and plyr packages.
    - **catboost (R)** ”catboost” is an open-source gradient boosting on decision trees library created by Yandex. It is the successor of the MatrixNet algorithm. Requires the catboost package.
    - **gbm (R)** ”gbm” generalized boosting regression models, is an implementation of extensions to Freund and Schapire’s AdaBoost algorithm and Friedman’s gradient boosting machine [43]. Requires the gbm and plyr package.

  - **LightGBM (R)** ”LightGBM” is a gradient boosting histogram based algorithm [18].
  - **xgbLinear (R)** ”xgbLinear” is an implementation of the extreme gradient boosting using a linear discriminant, from the xgboost package [44].
  - **xgbTree (R)** ”xgbTree” is an implementation of the extreme gradient boosting using trees, from the xgboost package [44].
  - **AdaBoostM1(SimpleCart) (WEKA)** adaboost: implements Freund and Schapires [45] AdaboostM1 method to boost classifiers, in our case, SimpleCart.
  - **LADTree (WEKA)** Creates a multi-class alternating decision tree using the LogitBoost strategy.
  - **LogitBoost(DecisionStump) (WEKA)** Boosts base learners that can handle regression tasks, in our case DecisionStump, using additive logistic regression.
  - **MultiBoostAB(SimpleCart) (WEKA)** Combines boosting with Wagging(a variant of bagging) to reduce overfitting.

- **Combined Classifiers**
  - **CascadeGeneralization(combo1) (WEKA)** A CascadeGenerlization with the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1.
  - **Grading(combo1)[ZeroR] (WEKA)** Grading using the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1, and the metaclassifier ZeroR.
  - **Grading(combo1)[IBk] (WEKA)** Grading using the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1 and the metaclassifier IBk.
  - **MultiScheme(W)** MultiScheme using the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1.
  - **StackingC(combo1) (WEKA)** StackingC using the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1.
  - **StackingC(combo2) (WEKA)** StackingC using the base classifiers RandomTree, ExtraTree, LDA and libLINEAR.
  - **Vote(combo1) (WEKA)** Vote using the base classifiers RandomTree, LDA, REPTree, NaiveBayes and IB1.
  - **Vote(combo2) (WEKA)** Vote using the base classifiers RandomTree, ExtraTree, LDA and libLINEAR.

- **Tree Ensembles**

- **cforest (R)** ”cforest” is an implementation of random forest which uses conditional inference trees as base learners, not random trees. From the party package.
- **parRF (R)** ”parRF” is an implementation of Breiman’s random forest algorithm, with trees grown in parallel, requires the e1071, randomForest, foreach and import packages [41].
- **rf (R)** ”rf” is an implementation of Breiman’s random forest algorithm (based on Breiman and Cutler’s original Fortran code) for classification and regression. It can also be used in unsupervised mode for accessing proximities among data points [41]. Requires the randomForest package.
- **treebag (R)** ”treebag” is an ensemble method with bagged classification and regression trees(rpart). Requires the e1071, plyr and ipred package.
- **wsrf (R)** ”wsrf” is an implementation of random forest that uses variable weighting for variable subspace selection in place of the traditional random variable sampling. Requires the wsrf package.
- **Bagging(ExtraTree) (WEKA)** a non-random committee of extratrees.
- **ForestPA (WEKA)** Builds a forest of simplecart trees with penalized attributes.
- **RandomCommittee(ExtraTree) (WEKA)** Builds a RandomCommittee of extremely randomised trees.
- **RandomCommittee(RandomTree) (WEKA)** An ensemble of RandomTrees created with different seeds for each RandomTree.
- **RandomForest (WEKA)** WEKA’s implementation of random forest.
- **RandomSubSpace(RandomTree) (WEKA)** RandomSubSpace with Random Tree as base classifier.
- **RandomSubSpace(ExtraTree) (WEKA)** RandomSubSpace with ExtraTree as base classifier.
- **RandomSubSpace(REPTree) (WEKA)** RandomSubSpace with REPTree as base classifier.

2.2.7 Instance based learners

Unlike other methods for classification, the instanced based learners do not create a model for classification, or do any work whatsoever before attempting to classify a new instance. Rather they do all the work during classification of a new instance, hence why they are nicknamed lazy methods. Lazy methods use the distance between instances to determine how similar they are, and use the similar instances to determine the class. The distance used is either the euclidean distance:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n}(a_r(x_i) - a_r(x_j))^2},$$

the Manhattan distance:
$d(\mathbf{p},\mathbf{q}) = \sum_{i=1}^{n} |p_i - q_i|$,
where $\mathbf{p}$=$p_1, p_2, ..., p_n$ and $\mathbf{q}$=$q_1, q_2, ..., q_n$
or the Minkowski distance:
$D(X, Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$
where $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$.

The formula for Euclidean distance requires all values to be numbers between 1 and 0, for nominal values that means the instance used for classification gets the value 1 for every nominal attribute and any instance it is compared to gets 1 if equal, 0 if not equal. Numerical attributes are converted to a number between 0 and 1 through standardization. This is done by determining the mean($\overline{x}$) and standard deviation(σ) for the attribute, then for each instance subtract the mean and divide by the standard deviation. $x' = \frac{x-\overline{x}}{\sigma}$

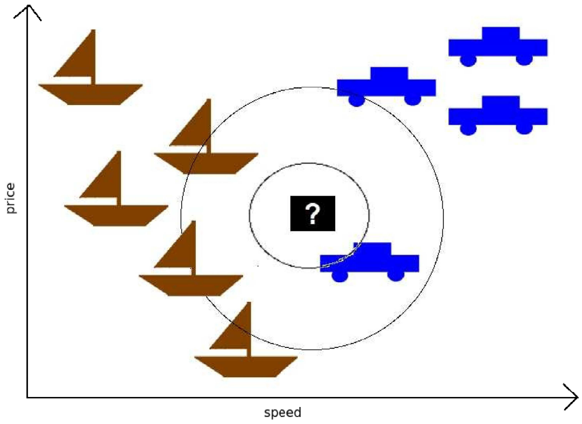


Figure 2.6: Figure shows the importance of the value K in K-nearest neighbors. With a k of 5, the object is classified as a boat, with a k value of 1, it is classified as a car.

Everyone who has recommended something for someone else has used the principle of an instanced based learner. When buying friends gifts or recommending something your thought process probably goes like this: he/she likes a,b,c and will probably enjoy d. In order to use lazy methods to get good results one has to weight the attributes, imagine a dating app runs into the following instances:

Name	Age	Sex	fav food	political leaning	Religion	Sex. orientation
Agnes	83	Female	Pizza	Conservative	Rastafari	Straight
Tobias	17	Male	Pizza	Conservative	Rastafari	Gay
Steve	30	Male	Chicken	Communist	Aasatru	Straight
Jack	24	Male	Cod	Libertarian	Sikhism	Straigh
Stacy	19	Female	Salad	Socialist	Roman Catholic	Straight

Note Table consisting of data for dating app.

Table 2.1: Instanced Based Learning Table

Looking at the instances in Table 2.1, without attribute weights the suggested date partner for Tobias would be Agnes.

2.2.7.1 Instance based learner implementations

- **knn (R)** "knn" is an implementation of k-nearest neighbour algorithm.
- **lvq (R)** "lvq" is Learning Vector Quantization, from the class package, a prototype based supervised learning algorithm.
- **IB1 (WEKA)** A nearest neighbour classifier.
- **IBk (WEKA)** A k nearest neighbour classifier.

2.2.8 Rule-based models

There are multiple ways to create a rule-based model. The simplest way to create a rule-based model is to create a decision tree and then convert every leaf to a rule. These rules can then be combined to predict the class. The decision tree in Figure: 2.2 can be converted to the following set of rules:

- **IF** x1<=10 **THEN** SQUARE
- **IF** x1>10 **AND** x2>8 **THEN** CIRCLE
- **IF** x1>10 **AND** x2<=8 **THEN** SQUARE

Rule-based models created from trees consist of mutually exclusive and exhaustive rules. Mutually exclusive because an instance cannot be the target of two rules, and exhaustive because there is one rule per attribute, which means they are not ordered(no main rule all others have to acknowledge) [47].

Another way to create a rule-based model is to use Sequential Covering. Sequential Covering algorithms learn one rule at a time, and removes the instances in the dataset the rule covers. This process can be repeated as many times as desired. Finally the set of rules can be sorted to make the more accurate rules will be used first when a new instance is to be classified [48].

Using a Rule-based model over Decision Trees comes with an obvious advantage for us human beings, it is easy to understand the model. Another advantage with rules compared to Decision Trees is that rules can be recursive, Decision Trees typically have a maximum depth. Rule-based models also have an easier time with changes in data, for a Decision Tree a new discovery might lead to a change in the tree structure, rule-based models can just add another rule [49].

2.2.8.1 Rule-based implementations

- **C5.0Rules (R)** "C5.0Rules" is an implementation of a rule-based model using Ross Quinlan's C5.0 algorithm [27]. Requires the plyr and C50 packages.
- **CSimca (R)** CSimca performs the (classical) SIMCA method from the rrcov and rrcovHD packages. This method classifies a data matrix x with a known group structure. To reduce the dimension on each group a PCA analysis is performed. Afterwards a classification rule is developed to determine the assignment of new observations [50].
- **Conjunctive (WEKA)** Creates one conjunctive rule that predicts one class. Can be used as a meta learner for boosters.
- **DecisionTable** Builds a simple decision table majority classifier.
- **DTNB (WEKA)** A decisiontable/Naive Bayes hybrid.
- **FURIA (WEKA)** FURIA builds an unordered set of fuzzy rules, that is, rules that gradually drop their validity outside their core
- **JRip (WEKA)** Implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER).
- **MODLEM (WEKA)** Builds rules using the rule-extraction algorith(REA) [?]
- **NNge (WEKA)** Nearest Neighbor With Generalization. A nearest-neighbour-like algorithm that uses non-nested generalized exemplars(hyperrectangles that can be viewed as if-then rules).
- **OLM (WEKA)** An implmementation of the Ordinal Learning Method.
- **OneR (WEKA)** Simply makes the single best rule it can.

- **PART (WEKA)** Class for generating a PART decision list. Builds a partial C4.5 decision tree in each iteration and makes the "best" leaf into a rule.

- **Ridor (WEKA)** An implementation of a RIpple-Down Rule learner, which generates a default rule first, then all the exceptions to that rule.

- **Vfdr (WEKA)** Builds decision rules incrementally from incoming data, expanding its rules over time.

- **ZeroR (WEKA)** A simple guess on the class which appears the most in the training set.

2.2.9 Support Vector Machines

Support Vector Machine is an algorithm that uses a nonlinear mapping to transform the original training data into a higher dimension, and within this new dimension, searches for the linear optimal separating hyperplane(i.e. a "decision boundary" separating the instances of one class from another) [51]. The weakness of support vector machines is that the training time can be very slow compared to other methods of classification(support vector machines consider every instance in the dataset to be a support vector), but they are very accurate [14], and are less prone to overfitting, because the Support Vector Machines rely only on the support vectors and not the complete training set [51].

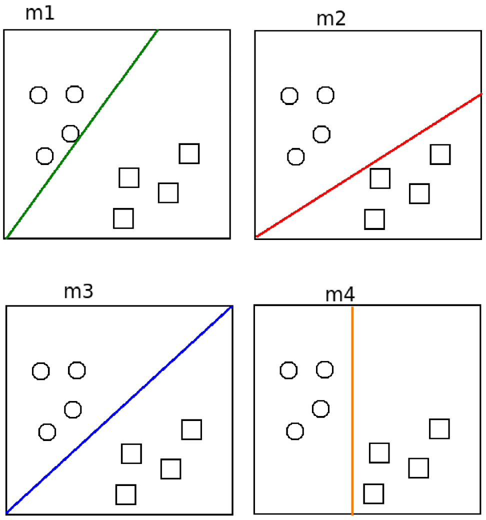


Figure 2.7: Figure shows 4 different models on a dataset, each with a hyperplane separating the classes.

Four main svm kernels are used, linear, polynomial, radial and sigmoid.

kernel	**Function**
Linear kernel	$k(x_i, x_j) = x_i^T x_j + 1$
Polynomial kernel	$k(x_i, x_j) = (x_i * x_j + 1)^d$
Gaussian radial basis function(RBF)	$k(x_i, x_j) = exp(-\gamma \vert\vert x_i - x_j \vert\vert^2)$
Sigmoid kernel	$k(x_i, x_j) = tanh(\alpha x_i^T * x_j + c)$

Note Table consisting of different kernels and their functions.

Table 2.2: Different kernels

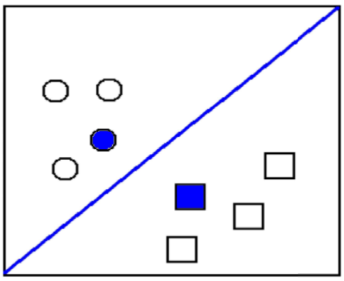


Figure 2.8: Figure shows which instances in the dataset are chosen as support vectors.

2.2.9.1 Support vector machine implementations

- **svmLinear (R)** ”svmLinear” is an implementation of a support vector machine with a linear kernel, from the kernlab package [52].
- **svmPoly (R)** ”svmPoly” is an implementation of a support vector machine with a polynomial kernel, from the kernlab package [52].
- **svmRadial (R)** ”svmRadial” is an implementation of a support vector machine with radial basis function kernel, from the kernlab package. [52]
- **svmRadialCost (R)** ”svmRadialCost” is another implementation of a support vector machine with radial basis function from the kernlab package, svmRadialCost only have one tuning parameter, cost [52].
- **LibLINEAR (WEKA)** libLINEAR: A wrapper for the libLINEAR library which includes multiple linear support vector machines.
- **LibSVM (WEKA)** A wrapper for the libsvm library created by Chang and Lin [53].
- **SMO (WEKA)** SMO trains a support vector machine using John Platt’s sequential minimal optimization algorithm [54].

2.2.10 Misc. classifiers used

- **glmnet (R)** ”glmnet” is an implementation of a Lasso or elasticnet generalized linear model [55]. Requires the glmnet and Matrix packages.
- **rocc (R)** ”rocc” is a receiver operating characteristics classifier from the rocc package, where features are selected according to their ranked AUC value in the training set. The selected features are merged by the mean

value to form a metagene. The samples are ranked by their metagene value and the metagene threshold that has the highest accuracy in splitting the training samples is determined. A new sample is classified by its metagene value relative to the threshold [56].

- **HyperPipes (WEKA)** Creates a hyperpipe for each class, that records the attribute bounds observed for that class. Test instances are classified according to the category that the instance seems to fit best in.
- **Logistic (WEKA)** Generates a multinomial logistic regression model with a ridge estimator.
- **SimpleLogistic (WEKA)** Builds linear logistic regression models. Uses LogitBoost with simple regression functions as base learners for fitting the models.
- **VFI (WEKA)** Classification by voting feature intervals.

Chapter 3

Method

3.1 Method

Our methodology is the following. For each classification algorithm, for each dataset, multiple runs with ten fold cross-validation is used for parameter tuning. This produces a list of parameter values, and the corresponding accuracy result from them. The run with the best accuracy is stored, with the parameter values. A ten fold cross-validation repeated 100 times is then executed using the parameter values from the best run. Results are stored in a database.

1. For each Classification Algorithm
    - For each Dataset
        (a) Perform a ten fold cross-validation run for each unique parameter value combination.
        (b) Order results on Accuracy
        (c) Store best run's Accuracy, Kappa and parameter values.
        (d) Perform a ten fold cross-validation repeated 100 times with the best run's parameter values.
        (e) Store results
    - end For each Dataset
2. end For each Classification Algorithm
3. Look at results

Software used:

- R version 3.4.4
- caret version 6.0-78
- WEKA 3.8.2
- MySQL version 14.14 distrib 5.7.21

hardware used:

- Desktop with i7-7700 3.6 GHz 16 GB RAM
- Asus X550J Notebook with intel core i7 processors (4th Gen) 4720HQ / 2.6 / 3.6 GHz and 8GB RAM
- lenovo g510 i7 4700 mq 2.4 GHz 8GB RAM

We chose to use R and WEKA as the software package. R is a language and environment for statistical computing and graphics. R is a dialect of the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers, who was awarded the 1998 ACM Software Systems Award for S. R is according to Drew Conway and John Myles White the lingua franca for analytics. Polls made by Kaggle and Knuggets puts R in the second spot in popularity for machine learning tasks. R was made by statisticians and for statisticians [57].

R can provide a wide variety of statistical and graphical techniques such as linear and nonlinear modelling, classical statistical tests, classification and clustering. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity [58]. R is free software under the terms of the Free Software Foundation's GNU General Public License in source code form.

WEKA is an open source data mining toolbox developed at the University of Waikato in New Zealand [59]. WEKA has support for all the tasks usually performed in data mining through numerous algorithms for pre-processing, classification, regression and clustering. The WEKA system is written in Java and distributed under the terms of the GNU General Public License.

With the WEKA workbench you can quickly try out different algorithms and methods on a dataset, and compare them. The workbench provides tools for comparing the results from different algorithms, allowing the user to identify the most appropriate method for learning.

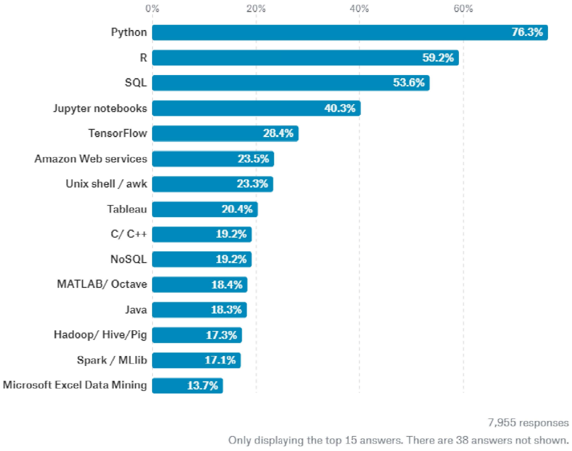


Figure 3.1: Figure shows how many of the participants in the poll use each software. Python and R are top 2. Almost 8000 participated in the poll.

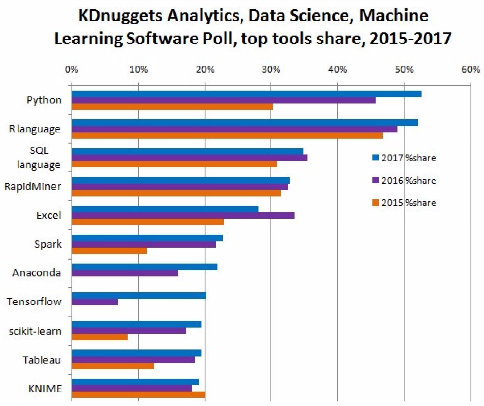


Figure 3.2: Figure shows Kdnuggets analytics for 2015, 2016 and 2017. The % represents how many of the participants in the poll used the software for machine learning. Python and R language are top 2. Almost 16000 participated in the poll.

WEKA's main user interface is the graphical user interface Explorer, which guides the user through multiple steps, preprocessing, classify, associate, cluster, select and visualize. WEKA may also be run through commandline, or another graphical user interface such as Knowledge Flow.

WEKA requires the data to be given in the ARFF format (Attribute Relation File Format), which it then uses to convert the data into a database table. Depending on the task, let's say create a well structured decision tree, the data may be restructured, and organized in a decision tree. Then WEKA will attempt to find association rules and attempt clustering to create a decision tree more suited for the task. When this is done, WEKA will select the best attributes to build the tree with, and visualize the final tree or compare the different trees if more than one method is attempted.

The main difference between R and WEKA is how the classification algorithms are used. E.g. if you want to use a support vector machine in WEKA you can choose SMO or libSVM and then have to choose polynomial, radial, sigmoid or linear kernel. R skips the first step and lets you choose kernel right away, e.g. method="svmRadial", "svmPoly", "svmLinear". In caret alone you have 230+ methods to choose between, while WEKA gives you a stepwise choice to select a classification algorithm.

We use 63 classifiers implemented in WEKA and 41 implemented in R. We used all the classifiers implemented in WEKA. The classifiers we chose implemented in R were chosen from the Manuel Fernández-Delgado et.al. study [14] where we picked the top 10 from caret(parRF, rf, svmPoly, svmRadialCost, svmRadial, C5.0, avNNet, nnet, pcaNNet and mlp), and some select others. We also chose new implementations which were not in the Manuel Fernández-Delgado et.al. study [14] such as extreme gradient boosting(xgboost), catboost, gbm and LightGBM.

3.1.1 Tuning in R

The R classifiers are run through the **train** interface from the caret package [60]. This interface also allows us to set up cross validation and repeated cross-validation. The train function also allows us to choose between two tuning options, the tuneGrid and the tuneLength. The tuneLength option is also known as randomsearch, which randomly selects # values for each tuning parameter. The tuneGrid option will attempt all values in the grid you provide.

When deciding the values for the parameters in the tuning grid we looked up what the parameters do in the classification algorithm's documentation. Some parameter's values were self-explanatory, the values of these parameters were the easiest to set up, e.g. where the parameters values are boolean such as naive_bayes' usekernel parameter(TRUE/FALSE), the options are known in advance such as C5.0's model type parameter(trees or rules), or such as randomFern's depth parameter which according to its documentation is between 1 and 16.

When the algorithm's parameters were not self-explanatory we used two approaches to find good values. The first approach is the caret package's random search option which chooses # number of values per parameter. This option is set up by replacing the tuneGrid option with the tuneLength=# option. We used tuneLength=5 on all the datasets. The second approach is to look online for good tuning parameter values, some competitors of Kaggle share their tuneGrid values. The final tuning values we chose were chosen by amalgamating the results from the two aforementioned approaches. As the number of parameters increased we decreased the number of values tested per parameter, e.g. for "mlp" we tried 19 different values for the number of neurons in the hidden layer, but for "mlpML" we only tried five values of neurons in each of the three hidden layers. We did this for no other reason than to save time, the mlp method had a tuning runtime of approximately two hours on 19 attempts, with 19 attempts in each hidden layer on "mlpML" we would reach $19^3 = 6859$ attempts, which is very time consuming.

WEKA requires the data files to be of ARFF format. We wanted to use the same data files for R and WEKA so we used the R package RWEKA to read the ARFF data files.

3.1.2 Tuning in WEKA

There are three optimization classifiers in weka; CVParameterSelection, GridSearch and MultiSearch.

3.1.2.1 CVParameterSelection

CVParameterSelection lets you optimize an arbitrary number of parameters, as long as they belong to the main classifier(i.e. you can not optimize any of the base classifiers parameters when tuning LogitBoost). The parameters are set up with a string, e.g. "P n_{min} n_{max} c" where P is the parameter token(the same as its command line argument, e.g. "-L" for the learning rate of the MultiLayerPerceptron), n_{min} is the lower bound, n_{max} is the upper and c is the number of steps. CVParameterSelection can only optimize numbers. CVParameterSelection can only optimize accuracy(classification) or mean squared error(regression).

3.1.2.2 GridSearch

Gridsearch lets you optimize two parameters, from both classifiers or filters, as well as nested parameters, like the base classifiers parameters when tuning LogitBoost. GridSearch also lets you pick the measure you want to optimize on, like accuracy, kappa or mean absolute error. The base classifiers parameters are accessed by prefacing them with "classifier.", and the filters parameters are prefaced with "filter.", arrays of objects can be accessed with "classifier[<index>]." or "filter[<index>].". GridSearch will automatically widen the search if the best parameter found was on the edge of the grid. GridSearch can only optimize numbers.

3.1.2.3 MultiSearch

MultiSearch lets you optimize an arbitrary number of parameters, from both classifiers or filters, as well as nested parameters. You have to specify what kind of search parameter you are going to optimize, either a MathParameter, which lets you set a min, max and step value, or a ListParameter, which lets you set a string of values separated by space. The ListParameter lets you specify not only strings or numbers, but also classes or boolean values(represented by strings). Like GridSearch, MultiSearch also lets you pick which measure you want to optimize on. Unlike GridSearch, MultiSearch does not automatically widen the search space.

Capability	**CVP**	**GS**	**MS**
More than two parameters	X	O	X
Nested parameters	O	X	X
Non numeric parameters	O	O	X

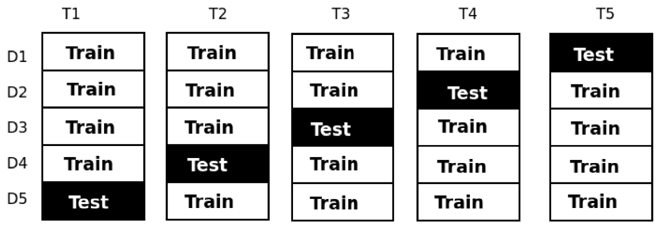
Table 3.1: Tuning-method-comparison

3.1.3 Chosen optimization package

We chose the MultiSearch package because it let us optimize however many parameters we wanted, both on the classifier itself as well as its nested parameters. MultiSearch comes with two search algorithms, RandomSearch and DefaultSearch. We chose the DefaultSearch class for tuning, set up with 8 threads. DefaultSearch searches through all the different parameters the user sets up, with one search for each unique combination. This can quickly lead to an overwhelming amount, as five different values for five different parameters will lead to 3125(5^5) unique combinations. DefaultSearch first runs through all combinations using two-fold cross-validation, then it sets the best combination of parameters found to be the centre of the search and searches through the closest parameters using ten-fold cross-validation. If a better option is found that option is set as the centre of the search and another ten fold search will be performed, this process is repeated until no better option can be found or the current best option is located at the edge of the parameter searchspace. If multiple combinations perform the same, the first found is used as centre of search/returned as best option found. This can be a problem on easy datasets, if multiple options return a perfect score, a suboptimal combination may be chosen.

3.1.4 Method -cont.

Rather than splitting the data into a training part and a test part, we chose to use k-fold cross-validation, also known as rotation estimation. In k-fold cross-validation the Dataset D is randomly split into k mutually exclusive subsets $D_1, D_2, ..., D_k$, of approximately same size. These subsets are known as folds. The classifiers is trained and tested k times, each time t 1, 2,..., k, it is trained on D without the fold D_t, and is then tested on D_t [61].



	T1	T2	T3	T4	T5
D1	Train	Train	Train	Train	Test
D2	Train	Train	Train	Test	Train
D3	Train	Train	Test	Train	Train
D4	Train	Test	Train	Train	Train
D5	Test	Train	Train	Train	Train

Figure 3.3: Figure shows 5 fold cross-validation. The original dataset is split randomly into 5 groups D1, D2, D3, D4 and D5 of equal size. Four groups are used for training and the last group for testing. This process is repeated five times.

The split into folds may not be representative of the dataset. Imagine if the dataset is for predicting human beings age and one fold comes with only children between 6 and 8 years old. A classifier being trained on that fold can not be expected to perform well on the test data, especially if the test data is poorly skewed as well, perhaps with only human beings from age 80+. To prevent this problem the folds should be stratified. Stratified folds contain a proper representation of each class, which makes the fold represent the whole dataset. The number k in k-fold cross-validation is arbitrary, there is no formula or rule for selecting it. The higher the number, the more work as training and testing is repeated k times, which takes more time. The number 10 and stratified is suggested in the book Data Mining, Concepts and Techniques [61], from a study by Kohavi et.al. [62]. Kohavi also suggested repeated cross-validation. We ended up selecting 10-fold stratified cross-validation repeated 100 times.

We use three metrics for the results, accuracy, Kappa and Friedman Rank. Accuracy on its own is not sufficient, for instance when trying to find false passports at an airport you can make an algorithm which always responds real passport and you will be correct 99.9% of the time, assuming 99.9% of the passports are real. 99.9% is a great result but does not help the custom regulators. Kappa is a number which shows to which extent of agreement two raters have, in this case two classification algorithms. Kappa ranges from 1 to -1, where 1 is complete agreement among the raters, and -1 is complete disagreement, a kappa of 0 shows an agreement that can be expected from random chance among the raters [63]. Kappa results will help as this algorithm will have a very low Kappa(near or below 0) and will not be used, the algorithms ZeroR(zero rules) and PCT(Probability Calibration Tree) gives a good indication of how low the kappa would be for simply putting all your eggs in one basket.

To sum it up, in R we first tune with a tuneGrid using caret's train interface, save the accuracy, Kappa and tuning parameter values of the best run, and then run a repeated 10 fold cross-validation repeated 99 times with the tuning parameters from the first run. This is repeated for each dataset. In WEKA they are tuned by searching over the parameterspace with twofolds cross-validation, after which it sets the best combination as the centre of the search, and runs subsequent ten folds cross-validations until the best option is found, repeating ten folds cross-validation until it has tested all nearby values. After the tuning it runs tenfold cross-validation 100 times with a seed from 0 to 99.

We used parallel execution to speed up the process. In R this is done with the doMC(Linux) or doParallel(Windows) packages. The rFerns method did not use parallel execution because it is a memory hog. Because multi-threading in WEKA is only available on some classifiers, we chose to use two different approaches. The learners that do support multi-threading ran one hundred attempts with eight cores through a simple loop, while those that did not ran hundred threads through a ThreadPoolExecutor, eight simultaneously, each with a unique seed, that returned their results back to the main thread. This is not a perfect solution, as this may lead to memory problems(We had to run ForestPA on the desktop, as seven GB of ram was not enough), especially on the ensemble learners, but luckily most of them support multi-threading. We chose not to look at the runtimes as WEKA and R run different code and the experiment was ran on three different computers.

3.2 Datasets

We use the datasets from the study performed by Manuel Fernández-Delgado et.al. [14]. They made three changes to the datasets:

1. The nominal values were converted to numeric values.
2. Data was preprocessed to have zero mean and standard deviation of one.
3. Missing values were converted to zero.

Given that we run ten fold cross-validation repeated 100 times per classification algorithm, we decided to remove some of the larger datasets as time is a limited resource. We also removed some of the smaller datasets. The following datasets were removed either because the execution time of these was too long, or because they were deemed too small.

- Adult
- Annealing
- Audiology-std
- Balloons
- Chess-krvk
- Conn-bench-vowel-deterding
- Connect-4
- Hayes-Roth
- Hill-valley
- Horse-colic
- Image-segmentation
- Lenses
- Letter
- Lung-cancer
- Miniboone
- Monks-1
- Monks-2
- Monks-3
- Optical
- Pendigits
- Pittsburg-bridges-SPAN
- Plant-margin

- Plant-shape
- Plant-texture
- Post-operative
- Soybean
- Spect
- Spectf
- Statlog-landsat
- Statlog-shuttle
- Thyroid
- Trains

Which leaves us with the following 89 datasets. Attributes range from 3(Titanic) to 262(Arrhythmia). Number of instances range from 100(Fertility) to 19020(Magic). Number of classes range from 2(several datasets) to 15(Libras and primary-tumor). The % majority of classes is 52.5. The datasets cover many different areas, we have 4 real life datasets(Oocytes...), multiple datasets cover biology, economy, medicine and statistics.

Table 3.2: List of Datasets

Name	**Instances**	**Classes**	**Attributes**	**%Maj**
Abalone	4177	3	8	34.6
Acute-inflammation	120	2	6	50.8
Acute-nephritis	120	2	6	58.3
Arrhythmia	452	13	262	54.2
Balance-scale	625	3	4	46.1
Bank	4521	2	16	88.5
Blood	748	2	4	76.2
Breast-cancer	286	2	9	70.3
Breast-cancer-wisc	699	2	9	65.5
Breast-cancer-wisc-diag	569	2	30	62.7
Breast-cancer-wisc-prog	198	2	33	76.3
Breast-tissue	106	6	9	20.7
Car	1728	4	6	70.0
Cardiotocography	2126	3	21	27.2
Cardiotocography-10	2126	10	21	77.8
Chess-krvkp	3196	2	36	52.5
Congressional-voting	435	2	16	61.4
Conn-bench-sonar-mines-rocks	208	2	60	53.4
Contrac	1473	3	9	42.7
Credit-approval	690	2	15	55.5
Cylinder-bands	512	2	35	60.9
Dermatology	366	6	34	30.6
Echocardiogram	131	2	10	67.2

Continued on next page

Table 3.2 – continued from previous page

Name	Instances	Classes	Attributes	%Maj
Ecoli	336	7	8	42.6
Energy-y1	768	3	8	46.9
Energy-y2	768	3	8	49.9
Fertility	100	2	9	88.0
Flags	194	8	28	30.9
Glass	214	6	9	35.5
Haberman-survival	306	2	3	73.5
Heart-cleveland	303	5	13	54.1
Heart-hungarian	294	2	12	63.9
Heart-switzerland	123	5	12	39.0
Heart-va	200	5	12	28.0
Hepatitis	155	2	19	79.3
Ilpd-indian-liver	583	2	9	71.4
Ionosphere	351	2	33	64.1
Iris	150	3	4	33.3
Led-display	1000	10	7	11.1
Libras	360	15	90	6.7
Low-res-spect	531	9	100	51.9
Lymphography	148	4	18	54.7
Magic	19020	2	10	64.8
Mammographic	961	2	5	53.7
Molec-biol-promoter	106	2	57	50.0
Molec-biol-splice	3190	3	60	51.9
Mushroom	8124	2	21	51.8
Musk-1	476	2	166	56.5
Musk-2	6598	2	166	84.6
Nursery	12960	5	8	33.3
Oocytes_merluccius_nucleus_4d	1022	2	41	67.0
Oocytes_merluccius_states_2f	1022	3	25	68.7
Oocytes_trisopterus_nucleus_2f	912	2	25	57.8
Oocytes_trisopterus_states_5b	912	3	32	57.6
Ozone	2536	2	72	97.1
Page-blocks	5473	5	10	89.8
Parkinsons	195	2	22	75.4
Pima	768	2	8	65.1
Pittsburg-bridges-MATERIAL	106	3	7	74.5
Pittsburg-bridges-REL-L	103	3	7	51.5
Pittsburg-bridges-T-OR-D	102	2	7	86.3
Pittsburg-bridges-TYPE	105	6	7	41.9
Planning	182	2	12	71.4
Primary-tumor	330	15	17	25.4
Ringnorm	7400	2	20	50.5
Seeds	210	3	7	33.3

Continued on next page

Table 3.2 – continued from previous page

Name	Instances	Classes	Attributes	%Maj
Semeion	1593	10	256	10.2
Spambase	4601	2	57	60.6
Statlog-australian-credit	690	2	14	67.8
Statlog-german-credit	1000	2	24	70.0
Statlog-heart	270	2	13	55.6
Statlog-image	2310	7	18	14.3
Statlog-vehicle	846	4	18	25.8
Steel-plates	1941	7	27	34.7
Synthetic-control	600	6	60	16.7
Teaching	151	3	5	34.4
Tic-tac-toe	958	2	9	65.3
Titanic	2201	2	3	67.7
Twonorm	7400	2	20	50.0
Vertebral-column-2clases	310	2	6	67.7
Vertebral-column-3clases	310	3	6	48.4
Wall-following	5456	4	24	40.4
Waveform	5000	3	21	33.9
Waveform-noise	5000	3	40	33.8
Wine	178	3	13	39.9
Wine-quality-red	1599	6	11	42.6
Wine-quality-white	4898	7	11	44.9
Yeast	1484	10	8	31.2
Zoo	101	7	16	40.6
Combined average %Maj				**52.5**%

3.3 Database

The database consists of seven tables, Dataset, Attribute, Classifier, Software, Family, Parameters and Results. The Dataset table contains the name, source location, number of instances, number of attributes and the number of classes of each dataset. The attribute table contains each attributes name, its possible values and a link to its dataset. The Classifier table contains the name of the classifier and a link to its software and family type. The Software table contains the name of the software, in our case R and WEKA. The Family table contains the name of the family types. The Result table contains the accuracy and kappa score, as well as a link to its parameters. The Parameters table contains a link to the classifier, a link to the dataset and the parameters the classifier found to be the best when it was tuned on the corresponding dataset.

The Dataset and Attribute table was populated by a php script that read through all the datasets we were going to use. The family table contains the name of each ”family” of classifiers, like SVM(Support Vector Machine) or

ANN(Artificial Neural Network). The Attribute table and the num_instances, num_attributes and num_classes columns of the Dataset table er not strictly necessary for our task, but it was included to give us a way to quickly look up that information in the case we wanted to see it(e.g. if a bayesian classifier is running slowly on semeion, we can look it up and see that it is because semeion has a lot of attributes).

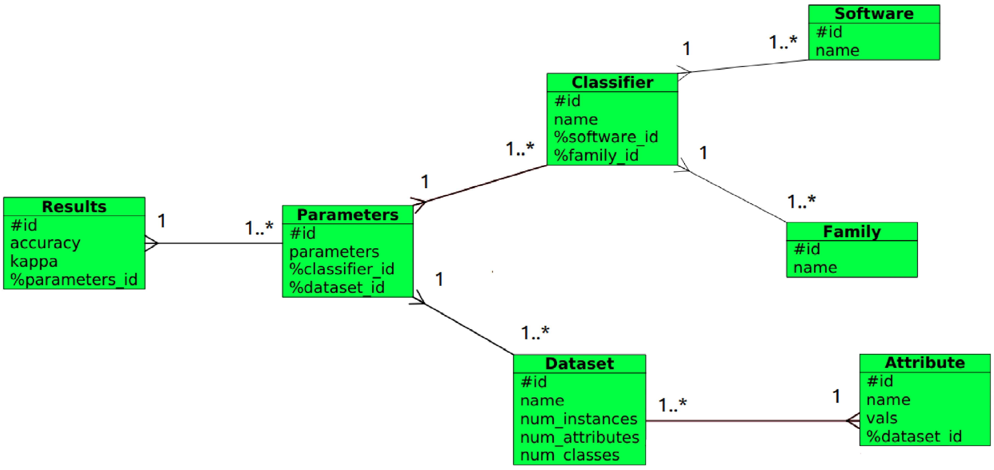


Figure 3.4: Image shows how our database is set up.

Table Dataset

Field	*Type*	*Key*
id	int	primary key AI
name	varchar(255)	
src	varchar(1024)	
instances	int(8)	
attributes	int(8)	
classes	int(8)	

Table Attribute

Field	*Type*	*Key*
id	int	Primary key AI
dataset_id	int	Foreign key
name	varchar(255)	
vals	varchar(1024)	

Table Classifier

Field	*Type*	*Key*
id	int	primary key AI
name	varchar(255)	
software_id	varchar(50)	Foreign key
family_id	varchar(50)	Foreign key

Table Software

Field	*Type*	*Key*
id	int	primary key AI
name	varchar(255)	

Table Family

Field	*Type*	*Key*
id	int	primary key AI
name	varchar(255)	

Table Parameters

Field	*Type*	*Key*
id	int	primary key AI
parameters	varchar(1024)	

Table Results

Field	*Type*	*Key*
id	int	primary key AI
parameters_id	int	foreign key
accuracy	double	
kappa	double	

Table 3.3: Database

3.4 Classification Algorithms

The following is a list of classification algorithms we tested.

Table 3.4: Classification Algorithms

Classification Algorithm	**Classification Algorithm**
A1DE	AdaBoostM1(SimpleCart)
AutoWEKA	avNNet
Bagging(ExtraTree)	bayesglm
BFTree	C5.0
C5.0Rules	C5.0Tree
CascadeGeneralization(combo1)	catboost
CDT	cforest
ConjunctiveRule	CSimca
ctree	DecisionStump
DecisionTable	Elman
ExtraTree	fda
ForestPA	FT
FURIA	gbm
glmnet	Grading(combo1)[IBk]
Grading(combo1)[ZeroR]	HoeffdingTree
HyperPipes	IB1
IBk	J48
J48Consolidated	J48graft
JRip	knn
LADTree	lda(R)
LDA(W)	LibLINEAR
LibSVM	LightGBM
Logistic	LogitBoost(DecisionStump)
lvq	mlp
mlpML	mlpWeightDecay
MODLEM	MultiBoostAB(SimpleCart)
MultilayerPerceptron	MultiScheme(combo1)
NaiveBayes	NaiveBayesUpdateable
naive_bayes	nb
NBTree	nnet
NNge	OLM
OneR	parRF
PART	pcaNNet
PCT	pda
QDA	RandomCommittee(ExtraTree)
RandomCommittee(RandomTree)	RandomForest
RandomSubSpace(ExtraTree)	RandomSubSpace(RandomTree)
RandomSubSpace(REPTree)	RandomTree

Continued on next page

Table 3.4 – continued from previous page

Classification Algorithm	Classification Algorithm
rda	REPTree
rf	RFlda
Ridor	rlda
rocc	rpart
sda	SimpleCart
SimpleLogistic	slda
SMO	StackingC(combo1)
StackingC(combo2)	svmLinear
svmPoly	svmRadial
svmRadialCost	treebag
Vfdr	VFI
Vote(combo1)	vote(combo2)
wsrf	xgbLinear
xgbTree	ZeroR

Note Table consists of the Classification Algorithms tested in this research.

3.5 Potential Pitfalls

Colour	Length	Width	WI
Pink	30	30	F
Yellow	36	36	F
Blue	34	40	T
Black	30	32	T
Red	34	34	F
Blue	32	32	T

Note Table shows a dataset with information about pants, and whether the pants are a Worthwhile Investment.

Table 3.5: Clothes Dataset

The study performed by Manuel Fernández-Delgado et.al. [14] chose to convert all nominal values to numeric values, this is not ideal for instance based learners, classification trees or rule-based models. To demonstrate why we will use a made up dataset in Table 3.5. Imagine a clothes branch stores information about the pants they sell, width, length and colour, and if the pants are worth making. The **T**rue and **F**alse values would be converted to 0 and 1, which has no impact. The colour would be converted differently, Pink=-1, Yellow=-0.5, Blue=0, Black=0.5 and Red=1. Without the conversion a decision tree model, rule-based model or instance based learner would treat each colour distinctively, e.g. for instance based learners the difference between Blue and Black=1, the difference between Pink and Red=1 and so on. With the conversion the difference between Pink and Blue is lower than the difference between Pink and Black which again is lower than the difference between Pink and Red.

For classification trees and rule-based models the splits may possibly be changed. Before the conversion a branch in a classification tree may have the test if colour is Black or Blue, creating two leaves one for Black or Blue and one for the rest. After conversion the branch could be changed to if Pants ≥ 0, after all red pants are rare so it might not show up in the training set. The conversion made by Manuel Fernández-Delgado et.al. may not have a huge impact on the decision tree and rule-based models, but the instance based learners may have a big impact in the results due to the change in representation.

Maciá and Bernadó-Mansilla published an analysis in 2014 about the use of the UCI repository [64]. The analysis contained a critique of usual practice in experimental comparisons. Their criticisms are in italics, and our response following:

1. *The choice of datasets can impact the results, some datasets may be more suited for a specific family of classifiers.* We use 89 datasets, any suitable combination of dataset and algorithm that creates a favourable situation for that algorithm may be offset by the next dataset being more suitable for a different algorithm. If a single algorithm is more suited for multiple datasets, perhaps this algorithm is simply better.

2. *Is the selection of classification algorithms representative enough and are they properly configured to work at their best performance.* In our test, all algorithms with tunable parameters are tested with several values. It is possible better values for the tuning parameters exist, but testing too many values of a parameter will take considerable time and algorithms with many parameters will create the cross product of the number of each parameter's values models. E.g. if we try ten values per parameter for xgbTree we end up with 10^7 models, and we simply do not have the time for that.

3. *It is still impossible to determine the maximum attainable accuracy for a data set, so that it is difficult to evaluate the true quality of each classifier.* We have no evidence that a higher accuracy can be achieved on a given dataset than the best we achieve in our test. The possibility of higher accuracy exists, but until it is shown to be possible we have no reason to believe it exists. Several of the algorithms reach the maximum accuracy on acute-inflammation, acute-nephritis and mushroom so the classifiers who do not reach 100% on these have something to improve.

4. *The data set complexity(measured somehow by the maximum attainable accuracy) is unknown, we do not know if the classification error is caused by unfitted classifier design(learner's limitation) or by intrinsic difficulties of the problem(data limitation).* Since we consider the maximum accuracy

to be the best accuracy we achieve with some algorithm, we can consider lower accuracies achieved by other classifiers to be caused by classifier limitations.

5. *The lack of standard data partitioning, defining training and testing data for cross-validation trials. Simply the use of different data partitioning will eventually bias the results, and make the comparison between experiments impossible.* We use stratified 10 fold cross-validation repeated 100 times, any lucky draw generated for an algorithm on a specific set will potentially be offset by a bad draw on another or a lucky draw by the rest of the algorithms elsewhere. The average result that is obtained by the trial should be closer to the expected value than a single ten fold cross-validation.

3.6 Replication

We used seed=765 for the tuning and seed=765 for the repeated cross-validation. The datasets can be found at
http://persoal.citius.usc.es/manuel.fernandez.delgado/papers/jmlr/data.tar.gz. The WEKA packages are not available in WEKA's graphical user-interface by default, except with WEKA itself. We recommend more than 8GB RAM. In order to replicate this experiment you will need the following R and WEKA packages:

Table 3.6: Classification Algorithms

R	**WEKA**
arm	alternatingDecisionTrees
C50	AnDE
caret	AutoWEKA
catboost	bestFirstTree
class	conjunctiveRule
doParallel	discriminantAnalysis
earth	Elman
e1071	extraTrees
foreach	ForestPA
gbm	functionalTrees
glmnet	fuzzyUnorderedRuleInduction
import	Grading
ipred	hyperPipes
kernlab	J48Consolidated
klaR	J48graft
MASS	JCDT
Matrix	LibLINEAR
mda	LibSVM
naivebayes	MODLEM

Continued on next page

Table 3.6 – continued from previous page

R	WEKA
party	multiBoostAB
plyr	multisearch
randomForest	naiveBayesTree
rFerns	NNge
RLightGBM	ordinalLearningMethod
rocc	probabilityCalibrationTrees
rpart	RBFNetwork
RWEKA	ridor
rrcov	simpleCART
rrcovHD	SimpleEducationalLearningSchemes
RSNNS	stackingc
sda	vfdr
sparsediscrim	votingFeatureIntervals
wsrf	weka.jar
xgboost	

Note Table consists of the packages required to replicate this research.

3.6.1 Changes made to R

The ”gbm” package has a memory leak. Trying to run repeated ten fold cross-validation will use up all your memory, and start eating your hard disk drive. To fix this we ran the ten fold cross-validations in a loop of size 99 instead as the memory is released once the cross-validation run is completed, seed values from 765-50 to 765+49.

3.6.2 Changes made to WEKA

1. **MODLEM**
   We changed the setOptions method in the MODLEM class so that it now accepts both numbers and letters as input for the RT, CM, CS and AS parameters(the GUI and MultiSearch remaps the numbers to letters). While its not strictly necessary to edit and recompile the package just for that, we wanted the code to be consistent.

2. **A1DE, CascadeGeneralization, Grading, MultiSCheme and StackingC**
   For some reason the A1DE algorithm feeds WEKA’s normalize function a null instead of a double. We added a nullcheck and made the method return if that was the case. The other algorithms fail on the lymphography dataset without this change.

3. **FURIA**
   Changed line 2253 which previously threw an exception(which according to its message ”should never happen:...”) when a double is either null or

infinite. We changed it to setting the aforementioned double to Double.minValue. This variable is later used in a comparison to find the best rule, so we thought it would be best if the algorithm picked the other option. The exception was thrown only once when we tuned FURIA on the fertility dataset.

4. **VFDR**
   The VFDR package we found was filled with bugs,
   the hoeffdingTieThreshold(hTT) input was mapped to the hoeffdingConfidence(hC) variable, nothing was mapped to the hTT variable, and the C parameter(hC variable) was not parsed by the setOptions method. We fixed the mapping and added a get method for the hTT. **Auto-WEKA** Added a getBestClassifier method that returned the best classifier found.

Chapter 4

Results

4.1 Classification Algorithm Results

We tested 104 classification algorithms, 41 from caret and 63 from WEKA. This includes 8 artificial neural networks, 6 bayes classification methods, 10 boosting ensembles, 17 classification trees,10 discriminant analysis methods,8 combined classifiers, 6 misc classification algorithms, 4 instance-based algorithms, 14 rule-based models, 7 support vector machines, 13 tree ensembles and 1 automatic algorithm. We chose not to show the complete table, because a table with 104(algorithms)x100(10 fold cross-validation repeats)x89(datasets)=925600 rows is not comprehensible. The catboost algorithm achieved the best Friedman Rank with 17.938, with a gap to the second best which is xgbTree with 20.747. The three algorithms parRF, svmPoly and follow rf within 1 point. The maximum average accuracy across all classification algorithms on all datasets is 85.5%, the best classification algorithm is catboost with 83.9%, followed by xgbTree with 83.7%. The xgbTree algorithm has the best kappa at 0.6654. The best classification algorithm for WEKA was AutoWEKA, followed by RandomForest and Bagging(ExtraTree). It is not that surprising that AutoWEKA is the best among the WEKA classification algorithms, after all it is the Swiss army knife of classification algorithms, it has every tool in the toolbox, and given enough time it will be able to figure out which tool works best. The tic-tac-toe dataset is the best example for this as none of the tree ensembles or boosting ensembles get 100% accuracy on that dataset.

The lowest accuracy achieved was 52.5% by the classification algorithms ZeroR and PCT. This is expected as they simply guess on the class which occurs the most, the % majority is 52.5. ZeroR and PCT also have the lowest Kappa with -0.001, which again is expected for the same reason.

Compared to the other classification algorithms among the top 10, catboost is better on atleast 46 datasets. The svmPoly is better than catboost on 42 of the datasets, which is the highest of the challengers, see Table 4.1.

Algorithm	**catboost better**	**catboost worse**	**equal**
xgbTree	49	39	1
parRF	52	36	1
rf	53	35	1
LightGBM	63	25	1
gbm	59	30	0
AutoWEKA	50	39	0
RandomForest	68	20	1
wsrf	64	24	1
svmPoly	46	42	1

Note Table shows on how many datasets catboost is better than the other classification algorithms.

Table 4.1: Head to head comparison catboost vs top 10

The classification algorithm lda got errors on two datasets, arrhythmia and zoo. Both zoo and arrhythmia contains more attributes than instances which might cause collinear data errors. The pcaNNet algorithm also got errors on arrhythmia. The RFlda algorithm got errors on led-display and zoo. WEKA algorithms had no errors. We had four options on how to handle the errors:

1. Give them 0 on accuracy and kappa.
2. Give them the average of all classification methods on accuracy and kappa.
3. Exclude the errors from the results(e.g. divide the combined accuracy on 8800 and not 8900 on the method which failed on 1 dataset).
4. Give them ZeroR's accuracy and kappa.

We chose option 1 because a non-existing model has zero predictive capabilities. The other options also come with some negative effects, option 2 will lower the Friedman rank of all methods with a score below the average on the dataset where a method failed, and with option 3 a classification algorithm that gets an error on heart-va(best=0.365) might get a better accuracy than if they managed to get a result on heart-va. We tried multiple other methods, like A2DE, protoclass, mda, hlda, RotationForest(caret), but had to cancel those because of too many errors, the working rule was, classification algorithms which failed on more than two datasets were stopped. On two classification algorithms,elm(extreme learning machine) and rFerns(random ferns) we only had time to run one 10 fold cross-validation. The result for rFerns was an accuracy of 0.7111 and for elm an accuracy of 0.784.

The combined runtime of this experiment is more than four months with three computers running simultaneously. Typically, the datasets magic and musk-2 were the most time-consuming, however, for some algorithms(nb, naive_bayes, fda...) semeion and arrhythmia were the most time consuming.

The caret package's methods (R) dominate the Friedman Rank. Top seven are all from caret, and 8/10 are from caret. WEKA is quite bottom heavy, having the nine worst Friedman Ranks, however this could easily have been changed if we ran the zeroR, etc... algorithms in R.

Interesting to note is the difference between RandomTree and ExtraTree. RandomTree has a higher Friedman Rank, higher accuracy and higher Kappa than ExtraTree.
On their bagged versions Bagging(ExtraTree) has a higher Friedman Rank but lower accuracy and kappa. On their RandomSubSpace versions RandomSubSpace(ExtraTree) is better on all metrics, and on their RandomCommittee versions RandomCommittee(ExtraTree) is again better on all metrics.

Table 4.2: Classification Algorithm Comparison, ordered by Friedman ranking.

Rank	Accuracy	Kappa	Classifier	Software
17.938	**0.8396**	0.662	catboost	r
20.747	0.8378	**0.6654**	xgbTree	r
20.966	0.8372	0.6536	parRF	r
21.157	0.8314	0.6458	svmPoly	r
21.421	0.837	0.6543	rf	r
23.163	0.8304	0.6442	svmRadial	r
24.5	0.8341	0.6571	gbm	r
25.169	0.8328	0.6532	AutoWEKA	weka
25.826	0.8344	0.6562	LightGBM	r
27.573	0.8307	0.6504	Bagging(ExtraTree)	weka
28.5	0.8314	0.6459	wsrf	r
28.697	0.8321	0.6526	RandomForest	weka
28.983	0.8302	0.6455	StackingC(combo1)	weka
29.197	0.8294	0.6526	RandomCommittee(ExtraTree)	weka
29.225	0.8247	0.634	svmRadialCost	r
30.876	0.8284	0.6456	CascadeGeneralization(combo1)	weka
31.163	0.8229	0.6438	avNNet	r
31.702	0.8273	0.6294	ForestPA	weka
31.742	0.8228	0.6414	SMO	weka
32.781	0.8262	0.6309	RandomSubSpace(ExtraTree)	weka
33.876	0.8276	0.6494	RandomCommittee(RandomTree)	weka
34.949	0.8271	0.6532	MultiBoostAB(SimpleCart)	weka
35.556	0.8174	0.6122	LibSVM	weka
35.91	0.821	0.6269	StackingC(combo2)	weka
36.073	0.8226	0.6343	LogitBoost(DecisionStump)	weka
36.663	0.8146	0.6299	nnet	r

Continued on next page

Table 4.2 – continued from previous page

Rank	Accuracy	Kappa	Classifier	Software
36.685	0.8226	0.6312	C5.0	r
37.118	0.816	0.6254	mlpWeightDecay	r
37.225	0.8241	0.6287	RandomSubSpace(RandomTree)	weka
37.713	0.8219	0.6174	RandomSubSpace(REPTree)	weka
38.438	0.8224	0.6354	Grading(combo1)[ZeroR]	weka
38.753	0.8211	0.6348	Grading(combo1)[IBk]	weka
39.478	0.8071	0.6205	pcaNNet	r
39.893	0.8229	0.6465	xgbLinear	r
41.298	0.8149	0.635	mlp	r
41.337	0.8224	0.6484	AdaBoostM1(SimpleCart)	weka
41.787	0.7984	0.6018	rda	r
42.197	0.7946	0.5852	glmnet	r
42.511	0.8186	0.6379	vote(combo1)	weka
43.275	0.8115	0.5899	cforest	r
43.77	0.8171	0.6296	MultiScheme(combo1)	weka
45.185	0.8028	0.5865	svmLinear	r
45.219	0.8013	0.5954	LibLINEAR	weka
46.242	0.7902	0.614	fda	r
47.393	0.8158	0.6297	treebag	r
47.882	0.8091	0.6221	FT	weka
48.242	0.7978	0.58	SimpleLogistic	weka
48.478	0.8049	0.6032	knn	r
48.758	0.7952	0.5811	sda	r
48.854	0.8129	0.6238	Vote(combo2)	weka
49.034	0.8032	0.5992	IBk	weka
50.163	0.7962	0.5925	Logistic	weka
50.23	0.8056	0.6245	MultilayerPerceptron	weka
50.73	0.8105	0.6262	LADTree	weka
51.893	0.7916	0.579	pda	r
52.326	0.7906	0.5793	LDA(W)	weka
53.629	0.7856	0.5701	mlpML	r
53.86	0.7947	0.5788	A1DE	weka
54.287	0.7999	0.5945	C5.0Rules	r
55.034	0.7717	0.5605	lda(R)	r
56.865	0.7944	0.5868	FURIA	weka
57.652	0.7956	0.5892	J48graft	weka
58.888	0.7885	0.568	rpart	r
59.258	0.7936	0.5849	C5.0Tree	r
60.247	0.788	0.5614	SimpleCart	weka
60.511	0.7792	0.5689	NaiveBayes	weka
61.287	0.7909	0.5786	NBTree	weka
61.994	0.7908	0.602	Elman	weka
62.461	0.7886	0.5799	JRip	weka

Continued on next page

Table 4.2 – continued from previous page

Rank	Accuracy	Kappa	Classifier	Software
64.371	0.7839	0.5711	CDT	weka
64.494	0.7835	0.561	BFTree	weka
64.629	0.7889	0.5863	PART	weka
64.792	0.7878	0.5832	J48	weka
65.466	0.7205	0.4585	RFlda	r
65.888	0.7605	0.5413	QDA	weka
66.045	0.7846	0.5816	REPTree	weka
66.32	0.7668	0.5503	NaiveBayesUpdateable	weka
66.522	0.7878	0.5806	NNge	weka
66.624	0.7789	0.5578	ctree	r
67.663	0.7746	0.5468	lvq	r
67.736	0.7468	0.5253	nb	r
68.23	0.7777	0.6287	MODLEM	weka
69.522	0.771	0.5664	IB1	weka
69.809	0.7428	0.5133	naive_bayes	r
69.815	0.7495	0.4788	slda	r
71.848	0.7595	0.5205	DecisionTable	weka
72.517	0.6214	0.3661	bayesglm	r
72.938	0.707	0.5414	rlda	r
74.933	0.7717	0.5674	Ridor	weka
76.82	0.6001	0.3332	rocc	r
77.096	0.7189	0.4996	HoeffdingTree	weka
77.657	0.7647	0.5526	RandomTree	weka
78.86	0.7374	0.5541	J48Consolidated	weka
79.208	0.7561	0.5406	ExtraTree	weka
79.826	0.6829	0.4696	CSimca	r
80.848	0.6822	0.3812	OneR	weka
84.73	0.6467	0.3042	DecisionStump	weka
85.528	0.6461	0.3029	ConjunctiveRule	weka
88.551	0.6599	0.4359	VFI	weka
88.657	0.6533	0.4082	OLM	weka
90.421	0.5941	0.2321	HyperPipes	weka
92.045	0.5257	-0.0011	PCT	weka
92.045	0.5257	-0.0011	ZeroR	weka
95.112	0.6283	0.2897	Vfdr	weka

Note
combo1=RandomTree, LDA, REPTree, NaiveBayes, IB1
combo2=RandomTree, ExtraTree, LDA, libLINEAR

Table shows the Friedman rank, average accuracy and average kappa for each algorithm tested. Ordered by Friedman rank.

4.2 Dataset Results

On the 89 datasets we had 46 different classification algorithms with the best accuracy, which means Wolpert's "No Free Lunch Theorem" is still true [5](the "No Free Lunch Theorem" states that there is no single classification algorithm that is the most accurate on all tasks). Among the 46 different classification algorithms, the RandomCommittee(ExtraTree) algorithm have the best score on the most datasets with eight, catboost, svmPoly and rda have seven, followed by AutoWEKA, parRF, wsrf and xgbTree with six. The avNNet, bagging(ExtraTree), SMO and svmRadial algorithms are best on five datasets each. Surprisingly, some of the weaker classification algorithms such as roc curves(rocc) was the best on three datasets and OneR was best on two, rocc was the best on acute-inflammation, pittsburgh-bridges-T-or-D and statlog-heart, OneR was the best on voting and echocardiogram.

It is worth noting that the best algorithm on congressional voting is OneR, which means that the best model to predict how Americans vote is very simple. The AutoWEKA algorithm also used OneR on congressional-voting, with the same result. The rocc algorithm is best on two datasets, statlog-heart and Pittsburgh-bridges-T-or-D, and the HyperPipes algorithm is the best on the ilpd-indian-liver dataset, which shows that even if a method seems to be weak overall(rocc is not even top 50 according to accuracy and HyperPipes is the third weakest according to accuracy) it can be the best method on a dataset.

When multiple algorithms are best on a dataset they tend to have something in common. Two algorithms are best on congressional-voting, both use OneR. Three are best on echocardiogram, all of them are "simple" models, perhaps the more intricate models simply overcomplicate. Three are best on the iris dataset, all of them are discriminant analysis. Two are best on the parkinsons dataset, both use IB1. The exceptions are the acute-inflammation, acute-nephritis, mushroom and tic-tac-toe datasets, but they have too many algorithms achieving a perfect score to require a specific approach to predict all instances correctly.

The ZeroR and PCT algorithms have a shared lowest accuracy on 26 of the datasets, they are never the best algorithms.

Table 4.3: Best result per dataset

Dataset	**Accuracy**	**Algorithm**
abalone	0.688	RFlda
acute-inflammation	1.000	52 got this score
acute-nephritis	1.000	41 got this score
arrhythmia	0.764	wsrf
balance-scale	0.999	svmPoly
bank	0.901	parRF

Continued on next page

Table 4.3 – continued from previous page

Dataset	**Accuracy**	**Algorithm**
blood	0.801	FT
breast-cancer	0.763	RandomSubSpace(ExtraTree)
breast-cancer-wisc —||—	0.973 —||—	NaiveBayesUpdateable NaiveBayes
breast-cancer-wisc-diag	0.980	svmPoly
breast-cancer-wisc-prog	0.822	gbm
breast-tissue	0.734	RandomCommittee(ExtraTree)
car	0.999	catboost
cardiotocography-10clases	0.891	xgbTree
cardiotocography-3clases	0.954	xgbLinear
chess-krvkp	0.996	C5.0
congressional-voting —||—	0.630 —||—	OneR AutoWEKA
conn-bench-sonar-mines-rocks	0.887	RandomCommittee(ExtraTree)
contrac	0.570	AutoWEKA
credit-approval	0.874	parRF
cylinder-bands	0.816	RandomSubSpace(RandomTree)
dermatology	0.983	wsrf
echocardiogram —||— —||—	0.855 —||— —||—	ConjunctiveRule OneR DecisionStump
ecoli	0.878	parRF
energy-y1	0.974	Bagging(ExtraTree)
energy-y2	0.939	catboost
fertility	0.889	wsrf
flags	0.699	LightGBM
glass	0.804	catboost
haberman-survival	0.757	LogitBoost(DecisionStump)
heart-cleveland	0.620	nnet
heart-hungarian	0.841	rlda
heart-switzerland	0.456	avNNet
heart-va	0.365	RandomCommittee(ExtraTree)
hepatitis	0.852	nb
ilpd-indian-liver	0.728	HyperPipes
ionosphere	0.955	svmRadial
iris —||— —||—	0.980 —||— —||—	pda sda LDA(W)
led-display	0.741	rda
libras	0.889	svmRadial
low-res-spect	0.919	SimpleLogistic

Continued on next page

Table 4.3 – continued from previous page

Dataset	**Accuracy**	**Algorithm**
lymphography	0.876	sda
magic	0.888	catboost
mammographic	0.838	FT
molec-biol-promoter	0.913	fda
molec-biol-splice	0.966	xgbTree
mushroom	1.000	35 got this score
musk-1	0.954	svmRadial
musk-2	0.999	MultilayerPerceptron
nursery	1.000	AdaBoostM1(SimpleCart)
oocytes_merluccius_nucleus_4d	0.858	SMO
oocytes_merluccius_states_2f	0.930	xgbLinear
oocytes_trisopterus_nucleus_2f	0.859	avNNet
oocytes_trisopterus_states_5b	0.944	avNNet
ozone	0.971	slda
page-blocks	0.976	RandomForest
parkinsons —||—	0.959 —||—	MultiScheme(combo1) IB1
pima	0.776	svmRadial
pittsburg-bridges-MATERIAL	0.875	naive_bayes
pittsburg-bridges-REL-L	0.744	RandomSubSpace(ExtraTree)
pittsburg-bridges-T-OR-D	0.894	rocc
pittsburg-bridges-TYPE	0.659	xgbTree
planning	0.723	Bagging(ExtraTree)
primary-tumor	0.510	xgbTree
ringnorm	0.987	rda
seeds	0.971	rda
semeion	0.963	svmPoly
spambase	0.959	catboost
statlog-australian-credit	0.696	fda
statlog-german-credit	0.772	wsrf
statlog-heart	0.857	rocc
statlog-image	0.986	AdaBoostM1(SimpleCart)
statlog-vehicle	0.853	rda

Continued on next page

Table 4.3 – continued from previous page

Dataset	**Accuracy**	**Algorithm**
steel-plates	0.806	AutoWEKA
synthetic-control	0.994	svmRadial
teaching	0.700	Vote(combo2)
tic-tac-toe	1.000	9 got this score
titanic	0.790	LADTree
twonorm	0.979	svmPoly
vertebral-column-2clases	0.863	mlpWeightDecay
vertebral-column-3clases	0.869	nnet
wall-following	0.999	LogitBoost(DecisionStump)
waveform	0.871	AutoWEKA
waveform-noise	0.869	SimpleLogistic
wine	0.997	rda
wine-quality-red	0.714	RandomCommittee(ExtraTree)
wine-quality-white	0.708	RandomCommittee(ExtraTree)
yeast	0.630	parRF
zoo	0.981	svmPoly

Note The Table contains the best algorithm(s) on each dataset and the accuracy achieved.

4.3 Probability of achieving 95% of the maximum accuracy

The catboost algorithm achieved an accuracy score of at least 95% of the best achieved accuracy on 82 out of 89 datasets. The four following algorithms xgbTree, parRf, rf and LightGBM all follow between 87.6% and 89.8%.

Algorithm	**x/89**	**% chance**
catboost	82/89	92.1%
xgbTree	79/89	88.7%
parRF	80/89	89.8%
rf	79/89	88.7%
LightGBM	78/89	87.6%
gbm	76/89	85.4%
AutoWEKA	75/89	84.2%
RandomForest	75/89	84.2%
wsrf	75/89	84.2%
svmPoly	72/89	80.9%
Bagging(ExtraTree)	72/89	80.9%
svmRadial	71/89	79.8%
Stacking(combo1)	73/89	82.0%
RandomCommittee(ExtraTree)	70/89	78.7%
CascadeGeneralization(combo1)	70/89	78.7%
RandomCommittee(RandomTree)	68/89	76.4%
ForestPA	70/89	78.7%
MultiBoostAB(SimpleCart)	70/89	78.7%
RandomSubSpace(ExtraTree)	72/89	80.9%
svmRadialCost	69/89	77.5%

Table 4.4: Probability of achieving 95% of the maximum Accuracy, or more.

4.4 catboost vs Maximum Accuracy

The maximum accuracy(considered the best average result per dataset across all classification algorithms) is 85.49%. The catboost algorithm got an accuracy of 83.96%, which is only 1.5 percentage point worse. The catboost algorithm is rarely far off from the maximum accuracy, in 68/89 datasets it is within 2.5 percentage point behind the maximum accuracy, in 82/89 datasets it has a score which is above 95% of the best achieved accuracy and in all datasets it has a score within 90% of the maximum accuracy. The catboost algorithm is above 5 percentage points behind on only three datasets, which is statlog-vehicle (8.1%), balance-scale(7.8%) and teaching(6.5%). The three aforementioned weak spots are easily seen on Figure 4.1, teaching can be seen as the gap approximately on position(15,60), statlog-vehicle can be seen as the gap approximately on position(37,75) and the gap approximately on position(83,92) is balance-scale.

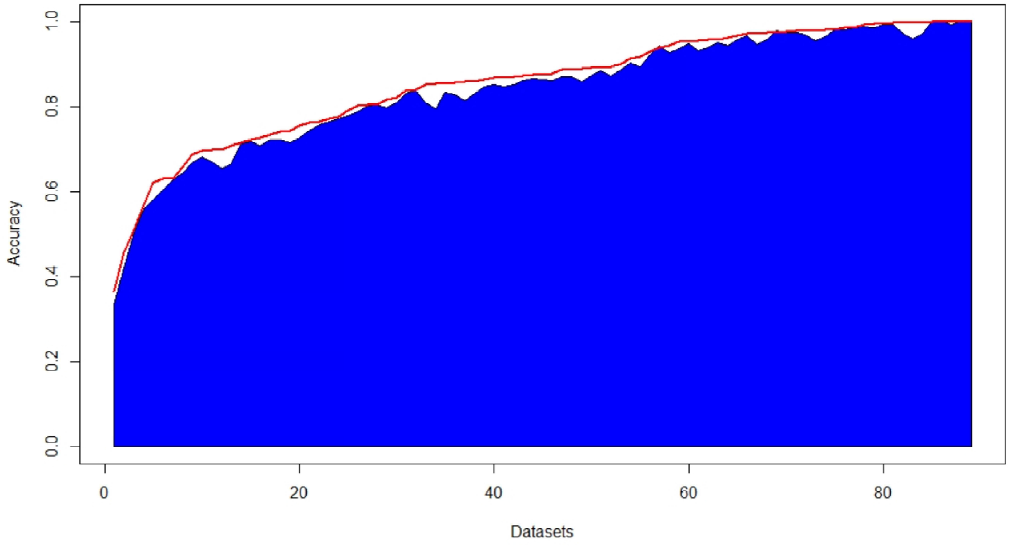


Figure 4.1: Figure shows how close catboost(blue) is to the maximum accuracy(red) on the datasets.

4.5 Variation In Results

The most stable algorithm among the top 20 is the WEKA algorithm bagging(ExtraTree), with a difference between the average of the lowest percentage correct on a 10 fold cross-validation per dataset and the average of the highest percentage correct on a 10 fold cross-validation per dataset of 2.96 percentage points. The least stable among the top 20 is another WEKA algorithm, the CascadeGeneralization(combo1). The xgbTree algorithm got the highest maximum with 85.70, slightly above catboost's 85.68. The catboost algorithm got a higher minimum with 82.07 vs xgbTree's 81.71. The two RandomForest implementations in caret are very similar, minimum 82.05 for parRF and 81.99 for rf, and maximum 85.15 for parRF and 85.12 for rf. The variation in results is surprisingly low, the highest among the top 20 being only 4.86 percentage points.

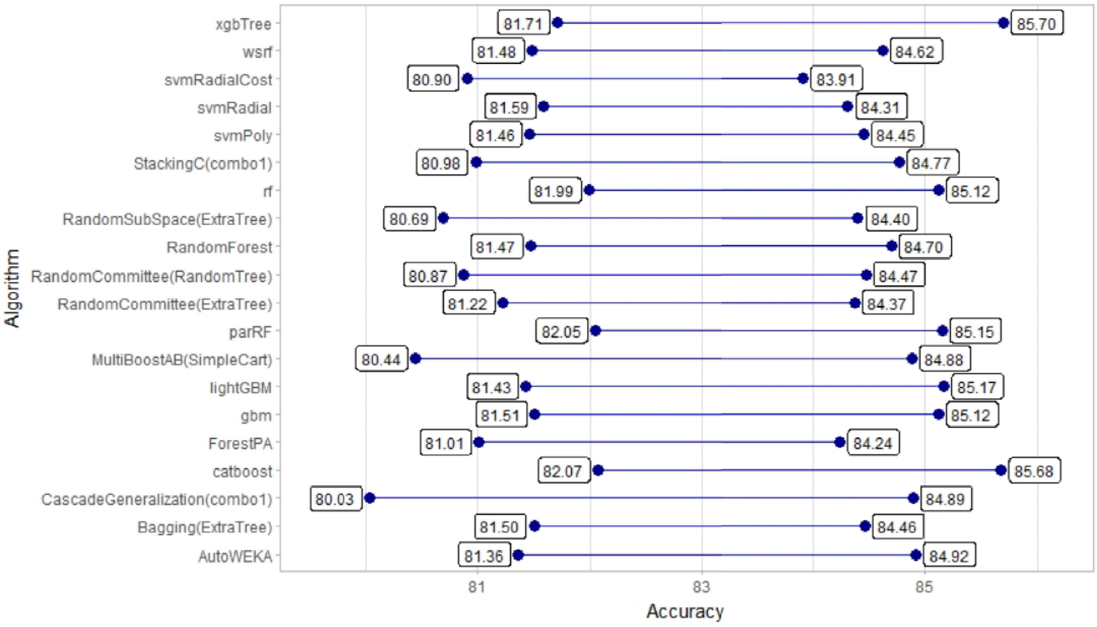


Figure 4.2: Figure shows the variation in results the top 20 algorithms got on the 100x10 fold cross-validation. Left dumbbell is the combined minimum on each dataset, right dumbbell is the combined maximum on each dataset.

Chapter 5

Discussion

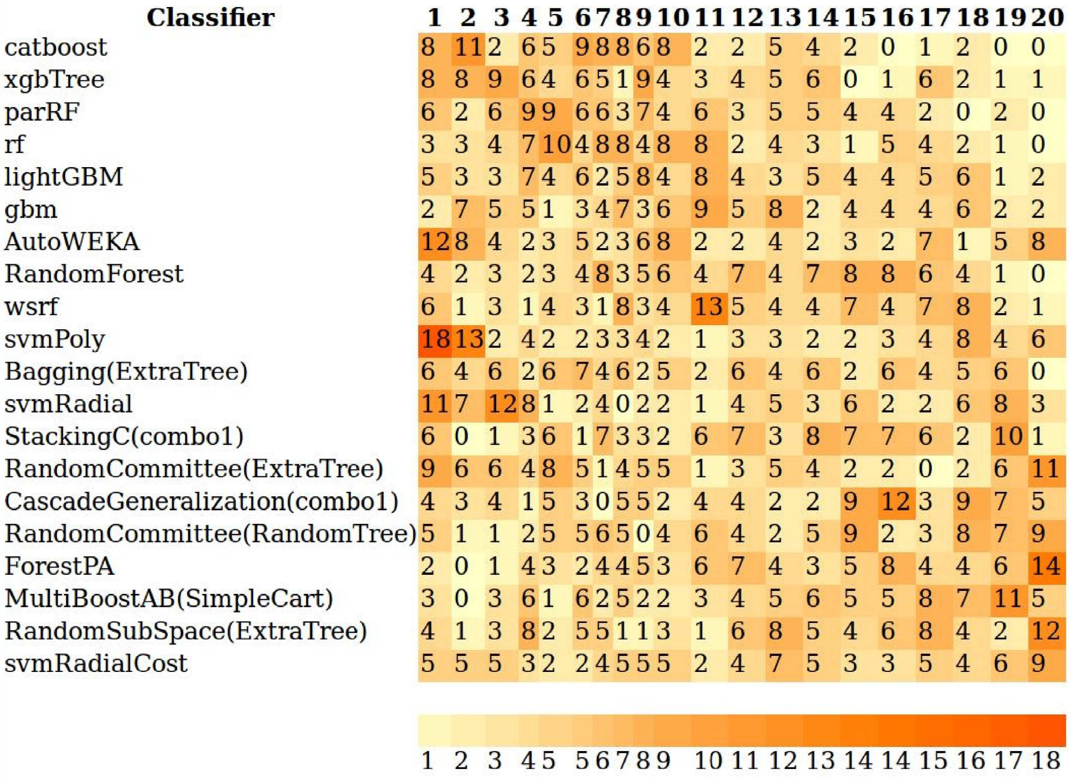
Algorithm	**avg Accuracy %**
catboost	83.96%
XgbTree	83.78%
parRF	83.72%
rf	83.70%
LightGBM	83.44%
gbm	83.41%
AutoWEKA	83.28%
RandomForest	83.21%
wsrf	83.14%
svmPoly	83.14%
Bagging(ExtraTree)	83.07%
svmRadial	83.04%
StackingC(combo1)	83.02%
RandomCommittee(ExtraTree)	82.94%
Cascadegeneralization(combo1)	82.84%
RandomCommittee(RandomTree)	82.76%
ForestPA	82.73%
MultiBoostAB(SimpleCart)	82.71%
RandomSubSpace(ExtraTree)	82.62%
svmRadialCost	82.47%

Note Table shows the top 20 classification algorithms.

Table 5.1: 20 best algorithms according to accuracy.

The top 20 algorithms consists of five boosting ensembles(catboost, xgbTree, LightGBM, gbm and MultiBoostAB(SimpleCart)), nine tree ensembles(parRF, rf, RandomForest, wsrf, Bagging(ExtraTree), RandomCommittee(ExtraTree), RandomCommittee(RandomTree), ForestPA and RandomSubSpace(ExtraTree)), three support vector machines(svmPoly, svmRadial and svmRadialCost), two combined classifiers(StackingC(combo1) and CascadeGenerlization(combo1)) and an automatic learning algorithm(AutoWEKA). Of the top 20 classification algorithms, only four are not ensembles, and three of those four are support vector machines, while the last one, AutoWEKA might use ensembles. In other words the only family of classifiers that can match ensembles are Support Vector Machines and AutoWEKA. Having AutoWEKA among the ten best algorithms is interesting, it is the best of WEKA's algorithms. Perhaps with longer tuning time it could have scored even better.

Perhaps more interesting than the average accuracy is the distribution among the top 20 classification algorithms, see Figure 5.



Classifier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
catboost	8	11	2	6	5	9	8	8	6	8	2	2	5	4	2	0	1	2	0	0
xgbTree	8	8	9	6	4	6	5	1	9	4	3	4	5	6	0	1	6	2	1	1
parRF	6	2	6	9	9	6	6	3	7	4	6	3	5	5	4	4	2	0	2	0
rf	3	3	4	7	10	4	8	8	4	8	8	2	4	3	1	5	4	2	1	0
lightGBM	5	3	3	7	4	6	2	5	8	4	8	4	3	5	4	4	5	6	1	2
gbm	2	7	5	5	1	3	4	7	3	6	9	5	8	2	4	4	4	6	2	2
AutoWEKA	12	8	4	2	3	5	2	3	6	8	2	2	4	2	3	2	7	1	5	8
RandomForest	4	2	3	2	3	4	8	3	5	6	4	7	4	7	8	8	6	4	1	0
wsrf	6	1	3	1	4	3	1	8	3	4	13	5	4	4	7	4	7	8	2	1
svmPoly	18	13	2	4	2	2	3	3	4	2	1	3	3	2	2	3	4	8	4	6
Bagging(ExtraTree)	6	4	6	2	6	7	4	6	2	5	2	6	4	6	2	6	4	5	6	0
svmRadial	11	7	12	8	1	2	4	0	2	2	1	4	5	3	6	2	2	6	8	3
StackingC(combo1)	6	0	1	3	6	1	7	3	3	2	6	7	3	8	7	7	6	2	10	1
RandomCommittee(ExtraTree)	9	6	6	4	8	5	1	4	5	5	1	3	5	4	2	2	0	2	6	11
CascadeGeneralization(combo1)	4	3	4	1	5	3	0	5	5	2	4	4	2	2	9	12	3	9	7	5
RandomCommittee(RandomTree)	5	1	1	2	5	5	6	5	0	4	6	4	2	5	9	2	3	8	7	9
ForestPA	2	0	1	4	3	2	4	4	5	3	6	7	4	3	5	8	4	4	6	14
MultiBoostAB(SimpleCart)	3	0	3	6	1	6	2	5	2	2	3	4	5	6	5	5	8	7	11	5
RandomSubSpace(ExtraTree)	4	1	3	8	2	5	5	1	1	3	1	6	8	5	4	6	8	4	2	12
svmRadialCost	5	5	5	3	2	2	4	5	5	5	2	4	7	5	3	3	5	4	6	9

Figure 5.1: A heatmap which shows the distribution among the top 20 classification algorithms. The horizontal columns show how the algorithm is spread across the 20 positions, while the vertical shows the position's spread across algorithms. E.g. catboost is the best algorithm 8 times, second best 11 times and third best 2 times. xgbTree is the best algorithm 8 times, second best 8 times and third best 9 times.

Among the top 20 classification algorithms, svmPoly is the best 18 times, second best 13 times and third best two times. This explains svmPoly's high Friedman Rank(fourth best), while it is only the tenth best according to accuracy. We can see the AutoWEKA algorithm is the best 12 times, svmRadial is the best 11 times, and the RandomCommittee(ExtraTree) is the best nine times. The best algorithm, catboost, is the best only eight times, and second best 11 times and third best two times. However, the catboost algorithm rarely occupy the bottom ten spots among the top 20, the catboost algorithm is in the bottom half of top 20 only 18/89 times. A similar trend is visible for the other top algorithms according to accuracy, xgbTree(29/89), parRF(31/89), rf(32/89) and LightGBM(42/89).

5.1 Paired T-tests

We ran paired T-tests on the top ten classification algorithms. A P value below 0.05 indicates there is statistically significant differences between the results of the two classification algorithms. The catboost algorithm is statistically significantly better than all methods except xgbTree and parRF, however it is very close to be significantly better than parRF. The only method which is significantly better than parRF, rf, svmPoly and AutoWEKA is catboost, which indicates that there is no practical difference between xgbTree, parRF, rf, AutoWEKA and svmPoly.

	catboost	xgb	parRF	rf	LGBM	gbm	AW	RF	wsrf
xgbTree	0.17								
parRF	0.05	0.7							
rf	**0.03**	0.6	0.32						
LGBM	**0.00**	**0.00**	0.06	0.08					
gbm	**0.00**	**0.00**	0.10	0.12	0.86				
AW	**0.04**	0.14	0.19	0.22	0.66	0.71			
RF	**0.00**	**0.00**	**0.00**	**0.00**	0.21	0.35	0.82		
wsrf	**0.00**	**0.00**	**0.00**	**0.00**	**0.03**	0.18	0.71	0.67	
svmPoly	**0.04**	0.11	0.17	0.20	0.48	0.47	0.68	0.88	0.98

Note xgb=xgbTree, LGBM=LightGBM, RF=RandomForest, AW=AutoWEKA

Table shows the P value of the paired t tests among the top ten classification algorithms. The statistically significant differences are marked with bold font.

Table 5.2: Paired T Tests

5.2 Effects of bagging

In all cases bagging increases the accuracy compared to the base classifier. RandomForest is 6.74 percentage points better than RandomTree, Bagging(ExtraTree) is 7.46 percentage points better than ExtraTree. The improvement is lower for the ctree algorithm, Cforest is 3.26 percentage points better than ctree. The improvement is even lower for the rpart algorithm, treebag is 2.73 percentage points better than rpart. The difference between nnet and avNNet is only 0.83 percentage points, which shows us what Breiman showed in 1996, stable Classification Algorithms do not benefit much from Bagging [36].

Algorithm	**Accuracy**	**Bagged**	**Accuracy**
RandomTree	76.47	RandomForest	**83.21**
ExtraTree	75.61	Bagging(ExtraTree)	**83.07**
rpart	78.85	treebag	**81.58**
nnet	81.46	avNNet	**82.29**
ctree	77.89	cforest	**81.15**

Note Table shows the accuracy of base-classifiers and the accuracy of their bagged versions. In all cases the bagged versions are better.

Table 5.3: Bagged classification algorithms vs base algorithms

5.3 Comparison to Similar Research

There are two studies which are very similar to this experiment, the Manuel Fernández-Delgado, Eva Cernadas, Senén Barro study *Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?* [14] and the Jacques Wainer study *Comparison of 14 different families of classification algorithms on 115 binary datasets* [15]. Our results are somewhat compatible with their results. Off course, we do not use the same datasets as Manuel Fernández-Delgado et.al. and Jacques Wainer, so comparing % accuracy and kappa numbers is useless, and we do not test the same amount of classification algorithms so the Friedman Ranks will also not be comparable, however we can still look at how the classification algorithms did compared to other classification algorithms. Both studies had random forest as the best classification algorithm, four of our top nine algorithms are random forest variants. When Manuel Fernández-Delgado, Eva Cernadas, Senén Barro did their study the top three boosting variants were not released on the market(gbm was but could only handle binary problems at the time), but Jacques Wainer did use the gbm algorithm(Wainer changed the all the datasets to binary problems), which had a slightly lower rank than the rf algorithm, but not significantly lower, and according to Jacques Wainer gbm, rf and svmRadial would perform at similar levels. We have rf, svmRadial and gbm in the fifth, sixth and seventh place according to the Friedman Rank.

The Delgado et. al. study had svmPoly as the best support vector machine, Wainer had svmRadial and we again got the best results with svmPoly. This could be explained by different tuning parameter values, perhaps Wainer found better values for svmRadial than we and Manuel Fernández-Delgado et.al. did or perhaps we and Delgado et.al. found better values for svmPoly than Wainer did.

We have more WEKA algorithms among our top 20(9) than Delgado et.al.(4), this can be explained by Delgado et.al. testing C and Matlab methods too, but a more plausible explanation is that Delgado et.al. did their study before the MultiSearch tuning package arrived on the market, which also explains why SMO barely got into top 50 in their study and is top 25 in ours.

When looking at the top ten caret algorithms from the Delgado study and comparing them to ours: As we can see in Table 5.4 the algorithms are not in the exact same order, some have switched one rank in the order, but no algorithm is more than one spot away from the other research's spot.

Delgado et.al.	**Ours**
parRF	parRF
rf	svmPoly
svmPoly	rf
svmRadialCost	svmRadial
svmRadial	svmRadialCost
C5.0	avNNet
avNNet	C5.0
nnet	nnet
pcaNNet	mlp
mlp	pcaNNet

Note Table shows the order among the best caret algorithms from the Delgado study, and the order of the same algorithms in our experiment.

Table 5.4: Result Comparison Delgado et.al. vs ours.

Delgado et.al. had six Random Forest variants, five Support Vector Machine variants, five Artificial Neural Networks, three Boosted Trees and a bagged libSVM among their top 20. We did not split the algorithms into the same families that Delgado et.al. did but it is easy to see what is what in their system. We have five Boosted Trees(4 out of top 6), six Random Forest variants, with another three which are earlier variants(a RandomSubSpace and two RandomCommittees), three Support Vector Machine variants, two combined classifier models and an automatic method(AutoWEKA). Several of our top 20 were not involved in the Delgado et.al study, some because they were not available such as gbm, LightGBM, xgbTree, catboost. They did try RandomSubSpace, but they used REPTrees as base classifier. They tried RandomCommittee(RandomTree), which they got on 30th place according to Friedman Rank, perhaps they did not have the tuning options we have now. We have no Artificial Neural Networks in our top 20, but the Artificial neural Networks have approximately the same ranks between themselves as in the Delgado study, which indicates that they have not improved, nor have they worsened, perhaps those interested in creating new artificial neural networks concentrate on deep learning instead.

Our weakest classification algorithms were ZeroR and PCT with 52.5%. Manuel Fernández-Delgado et.al had MetaCost, ZeroR, MultiScheme, CSC and Vote. Their MetaCost variant used ZeroR "bagging" without bootstrap aggregation, MultiScheme is an ensemble which selected one of several ZeroR classifiers using cross validation on the training set, CSC combines several ZeroR where each pattern is weighted depending of the cost assigned to each error type and Vote used sever ZeroR base classifiers combined using the average rule. Perhaps they did this because ZeroR is often set as the default baseclassifier in WEKA.

5.4 Comparison Within Family

As the scatter plot in Figure 5.2 show us, the classification algorithms in the Boosting Ensembles, Tree Ensembles, Combined Classifiers and Support Vector Machines families do not differ much from each other. The Classification Tree family has the highest spread, but even there most algorithms gather around 78-79% accuracy. The two bottom algorithms are ZeroR and PCT. The Misc classifiers, bayes classification methods, rule-based models and classification trees are very diverse in their performance.

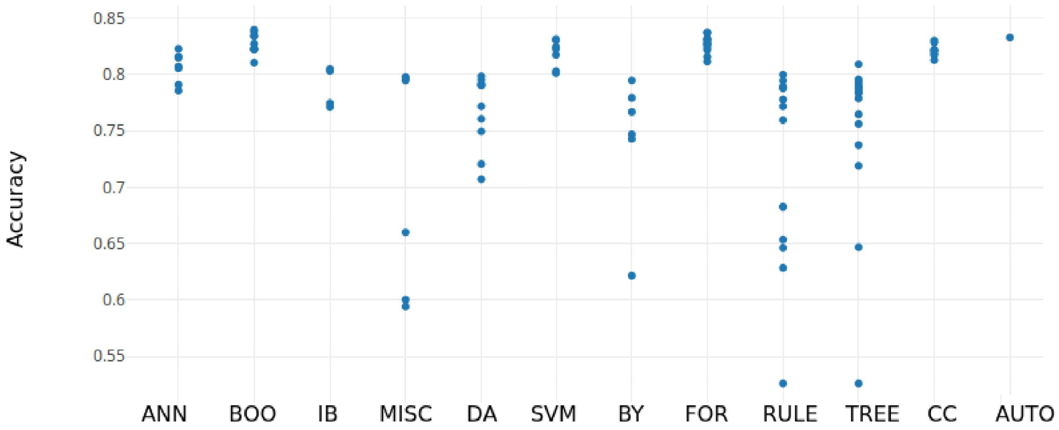
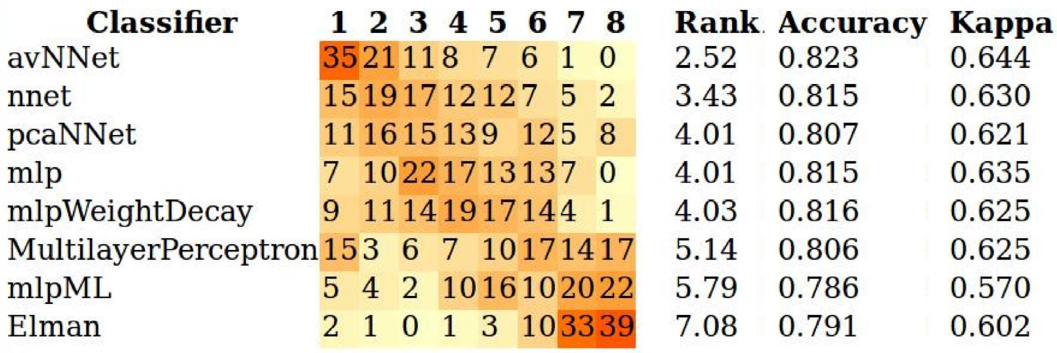


Figure 5.2: Figure shows a scatter plot with the accuracy as y values and the algorithms' family as x values. ANN=artificial neural networks, BOO=boosting ensembles, IB=instanced based algorithms, DA=discriminant analysis, BY=bayes classification methods, for=tree ensembles and CC=combined classifiers.

5.4.1 Artificial Neural Networks

The Delgado study has five Artificial Neural Network implementations among their top 20. We have none. The avNNet algorithm is their best ANN from caret too, it being slightly better than nnet, which is slightly better than pcaNNet, followed by mlp. Our ANN results differ just slightly, avNNet is the best, nnet second best, mlpWeightDecay third best, nnet fourth best and pcaNNet fifth best, although it is the third best according to the Friedman Rank. Delgado et.al. had poor results with mlpWeightDecay, 76.9% vs mlp's 80.3%. Delgado et.al. mention they use "automatic parameter tuning" in caret which indicates



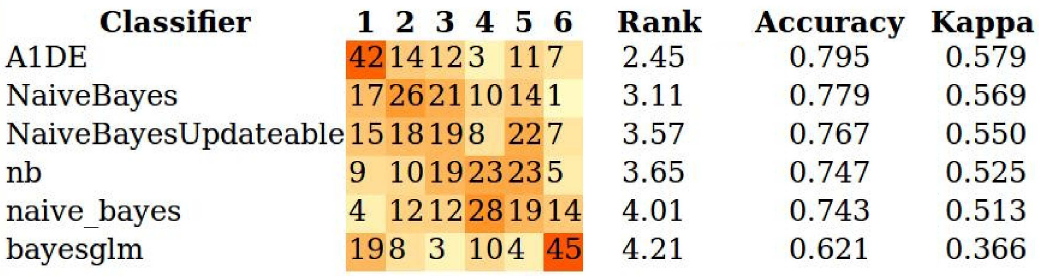
Classifier	1	2	3	4	5	6	7	8	Rank	Accuracy	Kappa
avNNet	35	21	11	8	7	6	1	0	2.52	0.823	0.644
nnet	15	19	17	12	12	7	5	2	3.43	0.815	0.630
pcaNNet	11	16	15	13	9	12	5	8	4.01	0.807	0.621
mlp	7	10	22	17	13	13	7	0	4.01	0.815	0.635
mlpWeightDecay	9	11	14	19	17	14	4	1	4.03	0.816	0.625
MultilayerPerceptron	15	3	6	7	10	17	14	17	5.14	0.806	0.625
mlpML	5	4	2	10	16	10	20	22	5.79	0.786	0.570
Elman	2	1	0	1	3	10	33	39	7.08	0.791	0.602

Figure 5.3: Figure shows a heatmap for Artificial Neural Networks, and the Friedman Rank, Accuracy and Kappa for the Artificial Neural Networks.

they use the random search(tuneLength=#), perhaps we can blame it some bad luck with the random parameter values.

The pcaNNest algorithm did not manage to create a model for the arrhythmia dataset, yet it is third best according to the Friedman Rank(the Friedman Rank of an algorithm increases a lot when every other algorithm is better than it). The worst ANN according to accuracy is the mlpML algorithm. Poor tuning values are very likely to blame, with three layers it can be difficult to set up. The Artificial Neural Networks performance are very similar, nnet, mlp and mlpWeightDecay are within 0.1 percentage point, the pcaNNet algorithm has a Friedman Rank close to mlp and mlpWeightDecay, and the WEKA variant of MultiLayer Perceptron has almost the same accuracy as pcaNNet.

5.4.2 Bayes Classification Methods

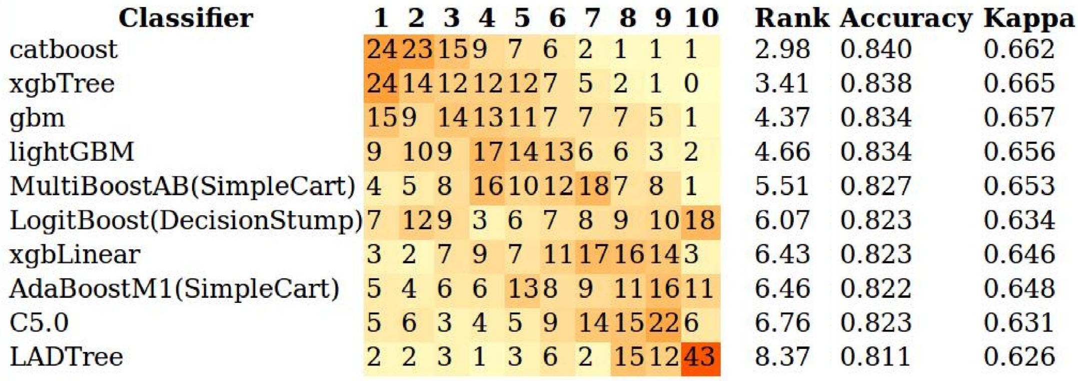


Classifier	1	2	3	4	5	6	Rank	Accuracy	Kappa
A1DE	42	14	12	3	11	7	2.45	0.795	0.579
NaiveBayes	17	26	21	10	14	1	3.11	0.779	0.569
NaiveBayesUpdateable	15	18	19	8	22	7	3.57	0.767	0.550
nb	9	10	19	23	23	5	3.65	0.747	0.525
naive_bayes	4	12	12	28	19	14	4.01	0.743	0.513
bayesglm	19	8	3	10	4	45	4.21	0.621	0.366

Figure 5.4: Figure shows a heatmap for Bayesian Classification Methods, and the Friedman Rank, Accuracy and Kappa for the Bayesian Classification Methods.

Among the Bayesian Classification Methods the A1DE is the superior classification algorithm across all metrics, followed by NaiveBayes and NaiveBayesUpdateable. A1DE is the best algorithm among them more often than any of the other two combined. The three WEKA methods are better than the three caret methods, the most inferior of which is bayes generalized linear model, which is 17.4 percentage points weaker than A1DE. None of the Bayesian Classification Methods reached 80% accuracy. Not much to say about these methods compared to Delgado et.al [14]'s study as we only have two methods in common, NaiveBayes and NaiveBayesUpdateable, where NaiveBayes has the higher Friedman Rank in both cases.

5.4.3 Boosting Ensembles

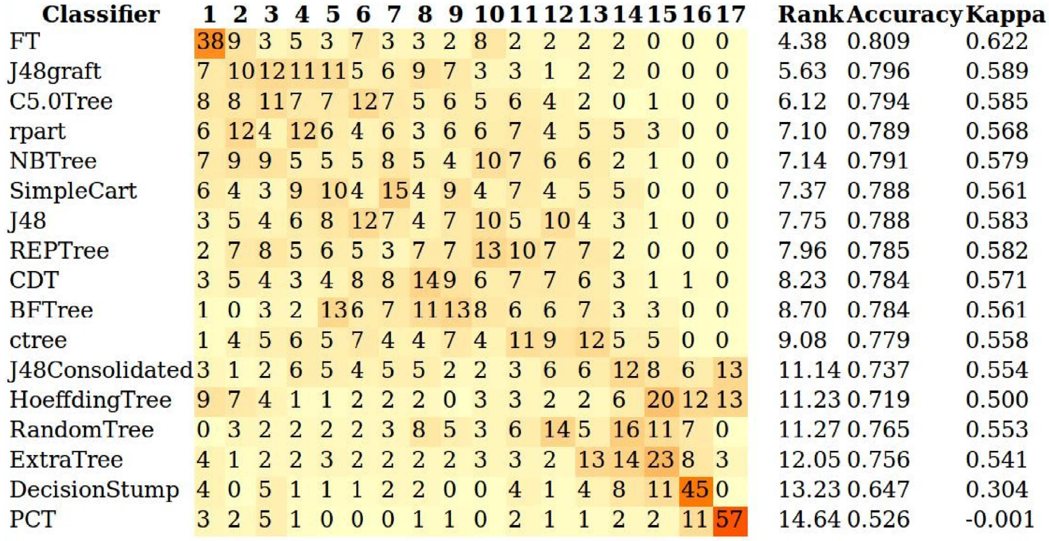


Classifier	1	2	3	4	5	6	7	8	9	10	Rank	Accuracy	Kappa
catboost	24	23	15	9	7	6	2	1	1	1	2.98	0.840	0.662
xgbTree	24	14	12	12	12	7	5	2	1	0	3.41	0.838	0.665
gbm	15	9	14	13	11	7	7	7	5	1	4.37	0.834	0.657
lightGBM	9	10	9	17	14	13	6	6	3	2	4.66	0.834	0.656
MultiBoostAB(SimpleCart)	4	5	8	16	10	12	18	7	8	1	5.51	0.827	0.653
LogitBoost(DecisionStump)	7	12	9	3	6	7	8	9	10	18	6.07	0.823	0.634
xgbLinear	3	2	7	9	7	11	17	16	14	3	6.43	0.823	0.646
AdaBoostM1(SimpleCart)	5	4	6	6	13	8	9	11	16	11	6.46	0.822	0.648
C5.0	5	6	3	4	5	9	14	15	22	6	6.76	0.823	0.631
LADTree	2	2	3	1	3	6	2	15	12	43	8.37	0.811	0.626

Figure 5.5: Figure shows a heatmap for the Boosting Ensembles, and the Friedman Rank, Accuracy and Kappa for the Boosting Ensembles

Among the Boosting Ensembles the catboost algorithm is the best on all metrics except Kappa, where xgbTree is superior. The catboost algorithm is the best or the second best among the Boosting Ensembles 47/89 times, xgbTree 38/89 times, LightGBM 19/89 times and gbm 24/89 times. There is a gap between catboost, xgbTree, LightGBM and gbm down to MultiBoostAB(SimpleCart), and another gap to LogitBoost, xgbLinear, C5.0 and AdaBoostM1(SimpleCart), all of which have an average accuracy of 82.3% or 82.2%. The worst Boosting Ensemble is LADTree, which is still a fairly good algorithm with an average accuracy of 81.1%. We cannot compare our results to Delgado et.al's results as C5.0 and LogitBoost(DecisionStump) are the only Boosted Trees we both tested, and they both have an accuracy of 82.3(82.26 rounded up). We chose to use SimpleCart as baseclassifier on AdaBoostM1 and MultiBoostAB because they did not seem to boost DecisionStump at all.

5.4.4 Classification Trees

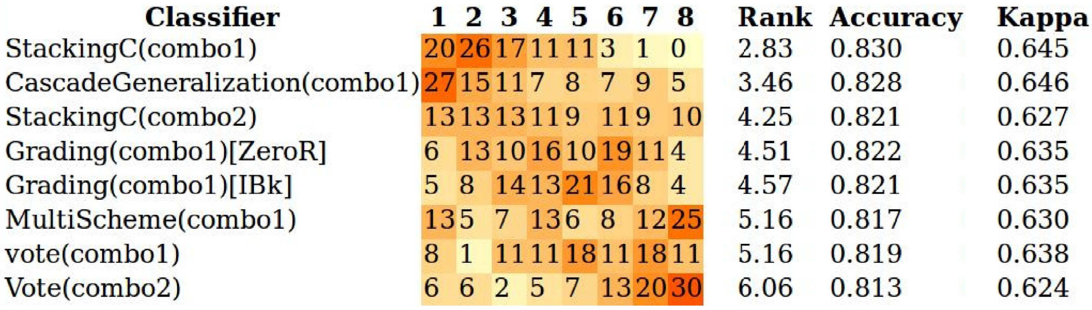


Classifier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	Rank	Accuracy	Kappa
FT	38	9	3	5	3	7	3	3	2	8	2	2	2	2	0	0	0	4.38	0.809	0.622
J48graft	7	10	12	11	11	5	6	9	7	3	3	1	2	2	0	0	0	5.63	0.796	0.589
C5.0Tree	8	8	11	7	7	12	7	5	6	5	6	4	2	0	1	0	0	6.12	0.794	0.585
rpart	6	12	4	12	6	4	6	3	6	6	7	4	5	5	3	0	0	7.10	0.789	0.568
NBTree	7	9	9	5	5	5	8	5	4	10	7	6	6	2	1	0	0	7.14	0.791	0.579
SimpleCart	6	4	3	9	10	4	15	4	9	4	7	4	5	5	0	0	0	7.37	0.788	0.561
J48	3	5	4	6	8	12	7	4	7	10	5	10	4	3	1	0	0	7.75	0.788	0.583
REPTree	2	7	8	5	6	5	3	7	7	13	10	7	7	2	0	0	0	7.96	0.785	0.582
CDT	3	5	4	3	4	8	8	14	9	6	7	7	6	3	1	1	0	8.23	0.784	0.571
BFTree	1	0	3	2	13	6	7	11	13	8	6	6	7	3	3	0	0	8.70	0.784	0.561
ctree	1	4	5	6	5	7	4	4	7	4	11	9	12	5	5	0	0	9.08	0.779	0.558
J48Consolidated	3	1	2	6	5	4	5	5	2	2	3	6	6	12	8	6	13	11.14	0.737	0.554
HoeffdingTree	9	7	4	1	1	2	2	2	0	3	3	2	2	6	20	12	13	11.23	0.719	0.500
RandomTree	0	3	2	2	2	2	3	8	5	3	6	14	5	16	11	7	0	11.27	0.765	0.553
ExtraTree	4	1	2	2	3	2	2	2	2	3	3	2	13	14	23	8	3	12.05	0.756	0.541
DecisionStump	4	0	5	1	1	1	2	2	0	0	4	1	4	8	11	45	0	13.23	0.647	0.304
PCT	3	2	5	1	0	0	0	1	1	0	2	1	1	2	2	11	57	14.64	0.526	-0.001

Figure 5.6: Figure shows a heatmap for the Classification Trees, and the Friedman Rank, Accuracy and Kappa for the Classification Trees.

The FT classification algorithm is the best Classification Tree across all metrics, it is the best algorithm more often than the following eight classification trees combined. FT is not that much better than the rest in terms of accuracy, which means it often is just slightly better than the rest. The PCT algorithm is the worst by far, which is not surprising considering it only guesses on the %majority of classes. Our results have a lot in common with The Delgado et.al. study, among the Decision Trees we both use C5.0Tree is the best, DecisionStump is the weakest followed by RandomTree and ctree. The rpart algorithm did become second among the eight trees we tested that Delgado et.al also tested, it is the fourth best for them, otherwise there are only minor differences. All the classification trees got an accuracy below 81.0%.

5.4.5 Combined Classifiers



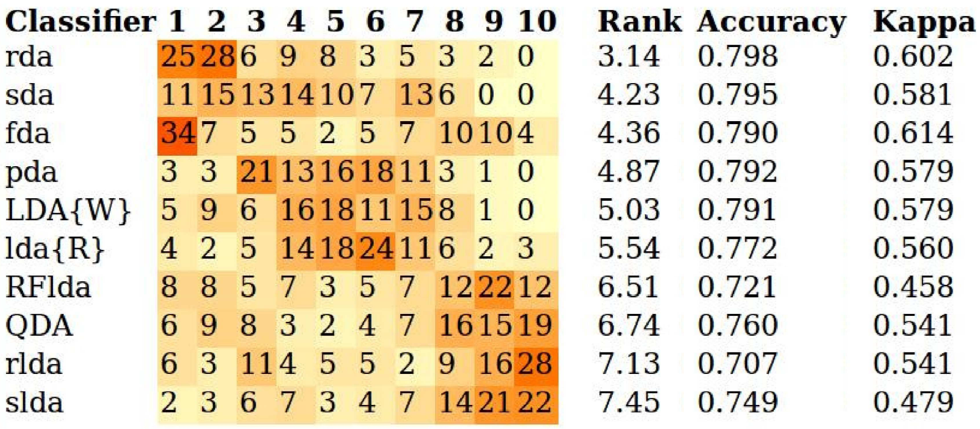
Classifier	1	2	3	4	5	6	7	8	Rank	Accuracy	Kappa
StackingC(combo1)	20	26	17	11	11	3	1	0	2.83	0.830	0.645
CascadeGeneralization(combo1)	27	15	11	7	8	7	9	5	3.46	0.828	0.646
StackingC(combo2)	13	13	13	11	9	11	9	10	4.25	0.821	0.627
Grading(combo1)[ZeroR]	6	13	10	16	10	19	11	4	4.51	0.822	0.635
Grading(combo1)[IBk]	5	8	14	13	21	16	8	4	4.57	0.821	0.635
MultiScheme(combo1)	13	5	7	13	6	8	12	25	5.16	0.817	0.630
vote(combo1)	8	1	11	11	18	11	18	11	5.16	0.819	0.638
Vote(combo2)	6	6	2	5	7	13	20	30	6.06	0.813	0.624

Figure 5.7: Figure shows a heatmap for the Combined Classifiers, and the Friedman Rank, Accuracy and Kappa for the Combined Classifiers.

When we set the combined classifiers up we chose base classifiers that, in our experience, were quick and had few tuning parameters. We tuned them scarcely, either with only 2 values for each parameter or none at all, to save time. Despite this, StackingC and CascadeGeneralization both achieved a score within the top 20, proving once again that an ensemble of weaker learners(RT(76.4), LDA(79,1), REPT(78,5), NB(77,9) and IB1(77,1)) can beat stronger learners. Amongst the combined classifiers, combo1 was unsurpringly the best combination, as it is more diverse than combo2, although, in its defence, vote(combo2) achieved the highest score on the teaching dataset.

The stackingc algorithm got the highest accuracy and friedman rank. The CascadeGeneralization(combo1) achieved the highest kappa score. For grading, ZeroR proved to be a better meta classifier than IBk, which Seewald & Furnkranz [65] used in their paper. MultiScheme was, with the base classifiers we chose, a weaker learner than Vote, but that does not necessarily mean that Vote is a better learner than MultiScheme, it all depends on the base classifiers. Overall, the combined classifiers performed quite well, with the lowest accuracy score being 81.3%, which is the highest minimum across the families.

5.4.6 Discriminant Analysis

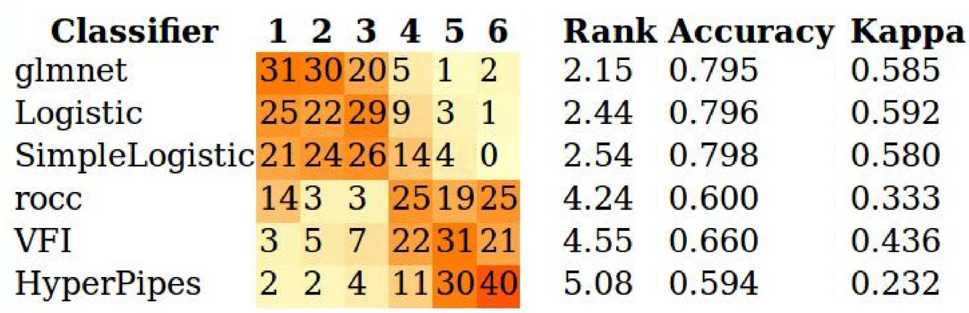


Classifier	1	2	3	4	5	6	7	8	9	10	Rank	Accuracy	Kappa
rda	25	28	6	9	8	3	5	3	2	0	3.14	0.798	0.602
sda	11	15	13	14	10	7	13	6	0	0	4.23	0.795	0.581
fda	34	7	5	5	2	5	7	10	10	4	4.36	0.790	0.614
pda	3	3	21	13	16	18	11	3	1	0	4.87	0.792	0.579
LDA{W}	5	9	6	16	18	11	15	8	1	0	5.03	0.791	0.579
lda{R}	4	2	5	14	18	24	11	6	2	3	5.54	0.772	0.560
RFlda	8	8	5	7	3	5	7	12	22	12	6.51	0.721	0.458
QDA	6	9	8	3	2	4	7	16	15	19	6.74	0.760	0.541
rlda	6	3	11	4	5	5	2	9	16	28	7.13	0.707	0.541
slda	2	3	6	7	3	4	7	14	21	22	7.45	0.749	0.479

Figure 5.8: Figure shows a heatmap for the Discriminant Analysis algorithms, and the Friedman Rank, Accuracy and Kappa for the Discriminant Analysis algorithms.

The rda algorithm is the best Discriminant Analysis algorithm we tested. It is best or second best among the Discriminant Analysis algorithms on 53/89 datasets, and it is the best on all metrics. The Delgado et.al. study had fda as their best. We both have slda as the weakest of the discriminant analysis algorithms. There is one major upset from their results, sda was their second weakest among the six we have both tested, for us it is the second best. Their best and our third best, fda, is the best Discriminant Analysis algorithm 34/89 times, far more often than any other algorithm. WEKA's LDA is almost two percentage point ahead of caret's lda, which got errors on arrhythmia and zoo, and thus was given an accuracy of 0 on those datasets. With a normal score on those datasets caret's lda would have been much closer.

5.4.7 Misc Classification Algorithms

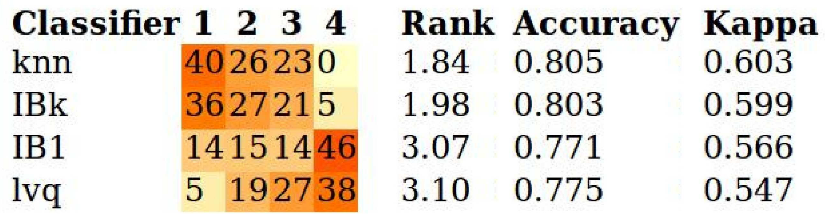


Classifier	1	2	3	4	5	6	Rank	Accuracy	Kappa
glmnet	31	30	20	5	1	2	2.15	0.795	0.585
Logistic	25	22	29	9	3	1	2.44	0.796	0.592
SimpleLogistic	21	24	26	14	4	0	2.54	0.798	0.580
rocc	14	3	3	25	19	25	4.24	0.600	0.333
VFI	3	5	7	22	31	21	4.55	0.660	0.436
HyperPipes	2	2	4	11	30	40	5.08	0.594	0.232

Figure 5.9: Figure shows a heatmap for the Misc Classification Algorithms, and the Friedman Rank, Accuracy and Kappa for the Misc Classification Algorithms.

The top three misc classification algorithms, glmnet, Logistic and SimpleLogistic have a huge gap down to the weaker three, rocc, VFI and HyperPipes. The glmnet algorithm is best according to Friedman Rank, with logistic second and SimpleLogistic third. When it comes to Accuracy it is the opposite, with SimpleLogistic best before Logisitc and glmnet. All of them are similar, only 0.03 percentage points difference between them. The rocc and HyperPipes algorithm are among the weakest we tested overall, but they still manage to be the best on some datasets. Delgado et.al. got comparable results, with VFI and HyperPipes the two weakest and glmnet SimpleLogistic and Logistic fairly even.

5.4.8 Instance Based Algorithms

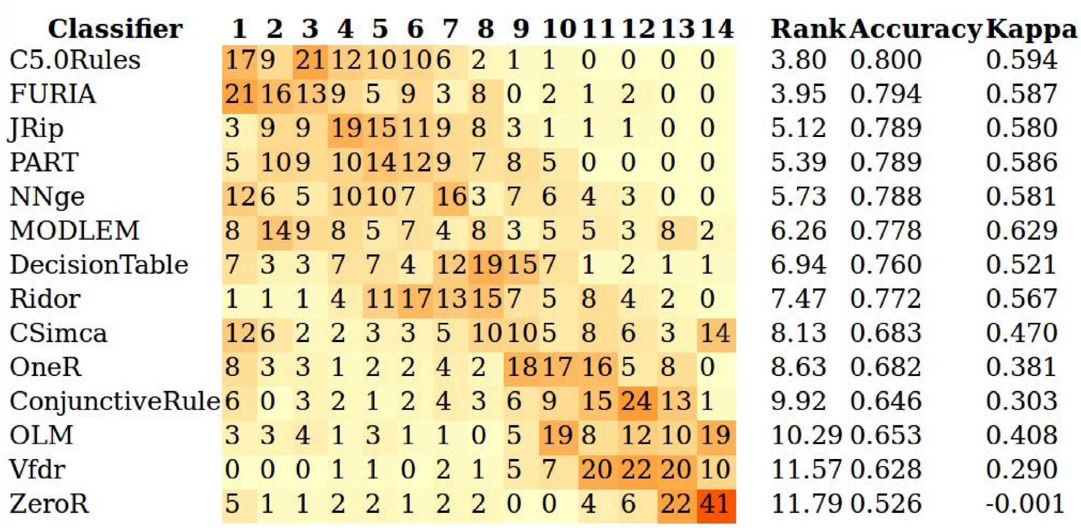


Classifier	1	2	3	4	Rank	Accuracy	Kappa
knn	40	26	23	0	1.84	0.805	0.603
IBk	36	27	21	5	1.98	0.803	0.599
IB1	14	15	14	46	3.07	0.771	0.566
lvq	5	19	27	38	3.10	0.775	0.547

Figure 5.10: Figure shows a heatmap for the Instance Based Algorithms, and the Friedman Rank, Accuracy and Kappa for the Instance Based Algorithms.

The knn and IBk algorithms are significantly better than IB1 and lvq, with three percentage points higher accuracies. The Delgado et.al. [14] results were similar to ours, knn being the best and IBk better than IB1. The nearest neighbor algorithms have comparable results to rule-based models, classification trees, bayes classification methods and discriminant analysis algorithms if we look only at their top 4.

5.4.9 Rule-based Models

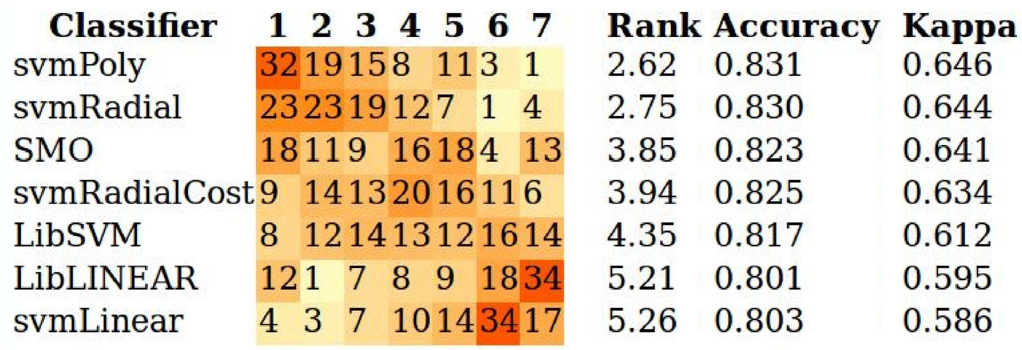


Classifier	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Rank	Accuracy	Kappa
C5.0Rules	17	9	21	12	10	10	6	2	1	1	0	0	0	0	3.80	0.800	0.594
FURIA	21	16	13	9	5	9	3	8	0	2	1	2	0	0	3.95	0.794	0.587
JRip	3	9	9	19	15	11	9	8	3	1	1	1	0	0	5.12	0.789	0.580
PART	5	10	9	10	14	12	9	7	8	5	0	0	0	0	5.39	0.789	0.586
NNge	12	6	5	10	10	7	16	3	7	6	4	3	0	0	5.73	0.788	0.581
MODLEM	8	14	9	8	5	7	4	8	3	5	5	3	8	2	6.26	0.778	0.629
DecisionTable	7	3	3	7	7	4	12	19	15	7	1	2	1	1	6.94	0.760	0.521
Ridor	1	1	1	4	11	17	13	15	7	5	8	4	2	0	7.47	0.772	0.567
CSimca	12	6	2	2	3	3	5	10	10	5	8	6	3	14	8.13	0.683	0.470
OneR	8	3	3	1	2	2	4	2	18	17	16	5	8	0	8.63	0.682	0.381
ConjunctiveRule	6	0	3	2	1	2	4	3	6	9	15	24	13	1	9.92	0.646	0.303
OLM	3	3	4	1	3	1	1	0	5	19	8	12	10	19	10.29	0.653	0.408
Vfdr	0	0	0	1	1	0	2	1	5	7	20	22	20	10	11.57	0.628	0.290
ZeroR	5	1	1	2	2	1	2	2	0	0	4	6	22	41	11.79	0.526	-0.001

Figure 5.11: Figure shows a heatmap for the Rule-based Models, and the Friedman Rank, Accuracy and Kappa for the Rule-based Models.

The best rule-based model is C5.0Rules, with FURIA in close second. JRip, PART and NNge are almost equal in accuracy. The weakest rule-based model is off course ZeroR. Our results are again similar to the Delgado et.al. [14] results, C5.0Rules is the best, JRip and PART next and ZeroR at the other end. Rule-based models have a high spread in accuracy, from 52.5% to 80.0%, only the classification trees have a higher spread.

5.4.10 Support Vector Machines

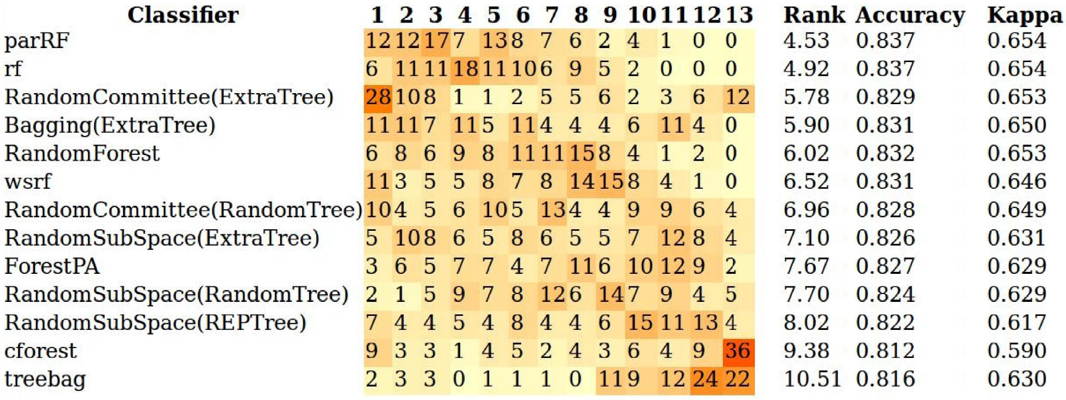


Classifier	1	2	3	4	5	6	7	Rank	Accuracy	Kappa
svmPoly	32	19	15	8	11	3	1	2.62	0.831	0.646
svmRadial	23	23	19	12	7	1	4	2.75	0.830	0.644
SMO	18	11	9	16	18	4	13	3.85	0.823	0.641
svmRadialCost	9	14	13	20	16	11	6	3.94	0.825	0.634
LibSVM	8	12	14	13	12	16	14	4.35	0.817	0.612
LibLINEAR	12	1	7	8	9	18	34	5.21	0.801	0.595
svmLinear	4	3	7	10	14	34	17	5.26	0.803	0.586

Figure 5.12: Figure shows a heatmap for the Support Vector Machines, and the Friedman Rank, Accuracy and Kappa for the Support Vector Machines.

The Support Vector Machines generally perform very well, all of them have an accuracy above 80.0%. The best SVM according to all metrics is svmPoly, it is slightly ahead of svmRadial in Friedman Rank, accuracy and kappa. SMO and svmRadial are the third and fourth best, with decent accuracies of 82.3% and 82.5%. The linear SVMs are atleast 1.4 percentage points less accurate than the nonlinear SVMs. We use six SVM algorithms that Delgado et.al. [14] used, among those they had the best results with svmPoly, followed by svmRadialCost and svmRadial. Our results differ, we have svmPoly as the best followed by svmRadial and then SMO, their SMO was outside top 50. Probably caused by improved tuning packages for WEKA, which was limited at the time of their research.

5.4.11 Tree Ensembles



Classifier	1	2	3	4	5	6	7	8	9	10	11	12	13	Rank	Accuracy	Kappa
parRF	12	12	17	7	13	8	7	6	2	4	1	0	0	4.53	0.837	0.654
rf	6	11	11	18	11	10	6	9	5	2	0	0	0	4.92	0.837	0.654
RandomCommittee(ExtraTree)	28	10	8	1	1	2	5	5	6	2	3	6	12	5.78	0.829	0.653
Bagging(ExtraTree)	11	11	7	11	5	11	4	4	4	6	11	4	0	5.90	0.831	0.650
RandomForest	6	8	6	9	8	11	11	15	8	4	1	2	0	6.02	0.832	0.653
wsrf	11	3	5	5	8	7	8	14	15	8	4	1	0	6.52	0.831	0.646
RandomCommittee(RandomTree)	10	4	5	6	10	5	13	4	4	9	9	6	4	6.96	0.828	0.649
RandomSubSpace(ExtraTree)	5	10	8	6	5	8	6	5	5	7	12	8	4	7.10	0.826	0.631
ForestPA	3	6	5	7	7	4	7	11	6	10	12	9	2	7.67	0.827	0.629
RandomSubSpace(RandomTree)	2	1	5	9	7	8	12	6	14	7	9	4	5	7.70	0.824	0.629
RandomSubSpace(REPTree)	7	4	4	5	4	8	4	4	6	15	11	13	4	8.02	0.822	0.617
cforest	9	3	3	1	4	5	2	4	3	6	4	9	36	9.38	0.812	0.590
treebag	2	3	3	0	1	1	1	0	11	9	12	24	22	10.51	0.816	0.630

Figure 5.13: Figure shows a heatmap for the Tree Ensembles, and the Friedman Rank, Accuracy and Kappa for the Tree Ensembles.

Among the tree ensembles the random forest methods from R, parRF and rf, are the top two. They have equal kappa and accuracy(parRF rounded down, rf rounded up). The RandomCommittee(ExtraTree) algorithm is the best more often than rf and parRF combined with 28 vs 6+12. It is worth noting that the tree ensembles are very even, and all above 81.2%, which is the result of the weakest algorithm cforest. 81.2% is higher than all the decision trees, which makes all tree ensembles better than all decision trees. Our results are comparable to the the Manuel Fernández-Delgado study, parRF the best followed by rf, however unlike them, our RandomForest(WEKA) results are much better than our cforest results. Overall, the tree ensembles performed very well, with the lowest accuracy score being 81.2% and have two of the top four algorithms overall.

5.5 What Could Have Been Done Differently

There are plenty of classification algorithms we could have tested. The most interesting in our eyes, which we would have tested given more time, are xgbDart(a variation of xgbTree), xgboost with histogram algorithm, ORFlog(forest of Oblique Trees), RotationForest, Decorate(claimed to be consistently more accurate than bagging and RandomForest [66] [67]) and Stacking with better base classifiers. The caret package's evolutionary tree algorithm(evtree) would have been fun to test as well.

Obviously, how we set up the tuning could have been done better. All classification algorithms get a single tunegrid for 89 datasets, which would never happen in practice. Every method should have a specific tunegrid for each dataset, with thorough testing to determine the boundaries of the parameters. The algorithms with few tuning parameters have a small advantage because we can set up a wide search on those few parameters, e.g. 100 values for lambda for the glmnet algorithm. Table 5.5 shows how much the boosting ensembles gain from proper tuning on datasets. The xgbTree algorithm is 1.4 percentage point and 1.35 percentage point better than the random forest algorithms after tuning, but loses almost all the advantage when ran on the other 99 10 fold cross-validations. The xgbTree algorithm is also much better than catboost on the tuning run, 0.76 percentage point ahead.

Algorithm	**Tuning%**	**Overall %**
catboost	84.70	83.96
xgbTree	85.46	83.78
parRF	84.11	83.72
rf	84.06	83.7

Note Table shows the avg accuracy on the tuning run and overall for the four best algorithms.

Table 5.5: Tuning vs overall accuracy

The tuning parameter values could have been different, for catboost we used what was recommend by yandex [68], and added 2 to depth, 0.01 to learning_rate, 150, 200, 250 to iterations, 2 and 1e-6 to l2_leaf_regulations and 0.9 and 0.8 to rsm. There are probably better values to use, if not generally then specifically on a single dataset. The LightGBM algorithm was not tuned properly, as we were running out of time and reduced the tuning grid to finish in time. We tuned all the values except num_leaves first, and later tuned num_leaves. This is off course not the proper way to do it, but when in a hurry one gets desperate. In reflection, we realize it should have been started earlier, as we infer it could reach better results. The MultiLayerPerceptron for WEKA had it's original tunegrid reduced to save time. In hindsight some other methods could have been tuned better as well.

If we were to redo this experiment we would have reduced the repeated cross-validation to maybe 25 repeats. The variation in results was low so it seems like 100 repeats is unnecessary.

5.6 Limits on this Research

While we tried to create a general test experiment, with many datasets, repeated cross-validations and many algorithms, our results do not give any absolute foresight into how a classification algorithm will perform on a random dataset. This experiment's results only show how the 104 algorithms perform on these 89 datasets. Different tuning parameters, different datasets and different seeds will produce different results. It is possible an algorithm such as LightGBM, AutoWEKA or some combined classifier could be tuned better and reach an average accuracy of 84% or higher on these dataset.

5.7 Answering the Research Questions

- **RQ 1: What are the best classification algorithms?** The best classification algorithms are catboost, xgbTree and parRF. The catboost algorithm is statistically significantly better than all other algorithms except xgbTree and parRF. The catboost algorithm achieved the best Friedman Rank, and it achieved the best accuracy. The xgbTree algorithm achieved the best kappa.
- **RQ 1.1: What do the better classification algorithms have in common, are they of the same family, or do they share other characteristics?** There is a clear trend that certain families are much better than others. Among the top 6 classification algorithms according to accuracy, the two best are boosting ensembles, and the fifth and sixth best are also boosting ensembles. Nine of the top twenty classification algorithms according to accuracy are tree ensembles, two are combined classifiers and three are support vector machines. Furthermore, all the boosting ensembles have an accuracy above 81.0%, all the tree ensembles have an accuracy above 81.0%, all the combined classifiers have an accuracy above 81.0% and all the Support Vector Machines have an accuracy above 81.0%, except the two linear support vector machines. There is a clear separation between these families and the rule-based models, decision trees, misc classifiers, instanced based classifiers, discriminant analysis classifiers and bayesian methods. The only other family that can reach above 81% are the artificial neural networks. This is easily seen in Figure 5.2. This does not mean that the decision trees, rule-based models etc. have no purpose, they can become strong ensembles, either through boosting, bagging or stacking.

- **RQ 1.2: Does Wolpert's No Free Lunch Theorem still hold true?** Absolutely. We have 46 different algorithms with the best result on 89 different datasets, multiple datasets have more than one best algorithm. The RandomCommittee(ExtraTree) algorithm is the best on eight datasets, catboost, rda and svmPoly are the best on seven and AutoWEKA, parRF, wsrf and xgbTree are the best on six datasets. The rocc algorithm, which is the fourth weakest algorithm according to accuracy, is the best algorithm on three datasets(acute-inflammation, pittsburgh-bridges-T-or-D and statlog-heart). The Manuel Fernández-Delgado, Eva Cernadas, Senén Barro study [14] argued that in practice, parRF was very near to the best attainable accuracy for almost all the data sets, and if that is where the bar is placed, catboost will jump over easily as it is much closer to the maximum attainable accuracy(1.6 percentage points away from maximum vs 4.9 percentage points from maximum).

- **RQ 1.3: How does Auto-WEKA perform compared to the other classification algorithms?** AutoWEKA got the eight highest accuracy with 83.28%, slightly above RandomForest(83.20%), making it the best classification algorithm from WEKA. AutoWEKA does make some weird choices now and then, like bagging(IBk) or Vote with ZeroR as one of the base classifiers. Given more time perhaps with an expanded list of classifiers and parameters to its arsenal it might have grabbed a spot at the podium. It would have been interesting to test a similar AUTOML version for caret, with access to the stronger algorithms like catboost, xgbTree and parRF. There is an AutoML algorithm from the h2o package, which also contains multiple strong algorithms such as xgboost, Generalized Linear Modeling, distributed Random Forest, GBM and Deep learning. Perhaps worth looking into for another study.

- **RQ 1.4: For the bagging ensembles, what are the effects of bagging compared to the base-classifiers?** The effects of bagging are what is to be expected. Breiman claimed in 1996 [36] that unstable classifiers with high accuracy works best for bagging. The RandomTree and ExtraTree are 6.74 and 7.46 percentage points better when used with bagging, while the more stable nnet is only 0.83 percentage point better when used with bagging.

- **RQ 1.5: Do our results differ from the results from similar research?** Compared to the Manuel Fernández-Delgado, Eva Cernadas, Senén Barro study [14] we have a few minor differences, but overall we reach almost the same results when we compare the classification algorithms both groups used. Their best, parRF and rf are also our best if we look at only the algorithms both groups used. Their top 10 algorithms from caret have almost the same ranks between them as they do in our test, some have switched positions(for them parRF best, rf second best svmPoly third best, for us parRF best svmPoly second best and rf third best(Friedman Rank)). We have better results with some of the WEKA methods, which we think are caused by new WEKA improvements coming after 2014. The bagged vs base classifier results are in agreement with what is expected of bagging [36].

Chapter 6

Conclusion

We tested 108 classification algorithms from caret and WEKA. The classification algorithms were tested on 10 fold cross-validation repeated 100 times, on 89 datasets. The algorithms were tuned once per dataset, with a manual tunegrid. We used three metrics, accuracy, kappa and friedman rank.

The five best algorithms according to accuracy were catboost(83.96), xgbTree(83.78), parRF(83.72), rf(83.7) and LightGBM(83.44). The five best according to kappa were xgbTree(66.54), catboost(66.2), gbm(65.71), rf(65.43) and parRF(65.36), all of which can be considered substantial agreement according to Cohen(61-80 =substantial agreement) [63]. The five best according to friedman rank were catboost(17.938), xgbTree(20.747), parRF(20.966), svmPoly(21.157) and rf(21.421). The worst classification algorithms were ZeroR and PCT with Accuracies of 52.5%, and Kappa of -0.001. The best algorithm, catboost, got an average accuracy of 83.96%, only 1.5 percentage point behind the maximum accuracy(the average of the best average result for each dataset across all algorithms) of 85.5%. The catboost algorithm overcomes 90% of the maximum accuracy on every single dataset, and overcomes 95% of the maximum accuracy in 92% of the datasets. The catboost algorithm achieved on average 98.15% of the maximum accuracy over all the datasets.

There is a hard point between two groups of families of classification algorithms, in our case 81.0% accuracy. All of the boosting ensembles, all the tree ensembles, all the combined classifiers, and all the Support Vector Machines except the linear SVMs have an accuracy above 81.0%. All the bayesian classifiers, all the decision trees, all the discriminant analysis algorithms, all the rule-based models, all the misc classification algorithms and all the instance-based models have their algorithms with a score below 81.0% accuracy. The only family of classifiers which have some above and some below 81.0% are the artificial neural networks. Most of the best classification algorithms are from the caret package, with the top 7 coming from caret, and 8/10 best using caret, all tuned with manual tunegrids, WEKA is not far behind however, with 9 of the top 20 on

Friedman Rank, and 10 of the top 20 on accuracy.

Chapter 7

Bibliography

[1] Tom M. Mitchell
Machine Learning(1997)
McGraw Hill. p.2.
ISBN 0-07-042807-7

[2] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. & Lang, K.(1989)
Phoneme recognition using time-delay neural networks
IEEE Transactions on Acoustics, Speech and Signal Processing, 37(3)
328-339.

[3] Cooper, G. et al. (1997).
An evaluation of machine-learning methods for predicting pneumonia mortality
Artificial Intelligence in Medicine, vol 9, no.2
pp 107-138.

[4] K. Tzidiris, Th. Kalampokas, G.A. Papakostas
Airfare Prices in Prediction Using Machine Learning Techniques
2017 25th European Signal processing Conference(EUSIPCO)

[5] David H. Wolpert
The lack of a priori distinctions between learning algorithms
Neural Computations, 9:1341-1390, 1996

[6] The Home for Data Science & Machine Learning
Kaggle.com

[7] Eric Bauer & Ron Kohavi
An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants
Machine Learning 36, 105-139 1999
Kluwer Academic Publishers

[8] Tjen-Sien Lim, Wei-Yin Loh & Yu-Shan Shih
A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-Three Old and New Classification Algorithms
Machine Learning, 40, 203-228 2000
2000 Kluwer Academic Publishers

[9] Olcay Taner Yildiz & Ethem Alpaydin
Linear Discriminant Trees
International Journal of Pattern Recognition and Artifical Intelligence
Vol 19. No. 3 (2005)
pp. 323-353

[10] xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Quing Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand & Dan Steinberg
Top 10 algorithms in data mining
Springer-Verlag London Limited 2007

[11] Gouda I. Salama, M. B. Abdelhalim and Magdy Abd-elghany Zeid
Experimental Comparison of Classifiers for Breast Cancer Diagnosis
2012

[12] K. Wisaeng
A Comparison of Different Classification Techniques for Bank Direct Marketing
International Journal of Soft Computing and Engineering
ISSN:2231-2307, Volume 3 Issue 4 (2013)

[13] Diego Raphael Amancio, Cesar Henrique Comin, Dalcimar Casanova, Gonzalo Travieso, Odemir Martinez Bruno, Francisco Aparecido Rodrigues and Luciano da Fontoura Costa
A Systematic Comparison of Supervised Classifiers
PLoS ONE 9(4):e94137. doi:10.1371/journal.pone.0094137 (2014)

[14] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro
Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?
Journal of Machine Learning Research 15(2014) 3133-3181

[15] Jacques Wainer
Comparison of 14 different families of classification algorithms on 115 binary datasets

[16] Edwin Raczko & Bogdan Zagajewski
Comparison of support vector machine, random forest and neural network classifiers for tree species classification on airborne hyperspectral APEX images,

European Journal of Remote Sensing, 50:1, 144-154, DOI:10.1080/22797254.2017.1299557

[17] Anthony Goldbloom,
in an interview on
*https://www.datasciencecentral.com/profiles/blogs/
has-deep-learning-made-traditional-machine-learning-irrelevant*

[18] LightGBM's documentation.
https://LightGBM.readthedocs.io/en/latest/index.html

[19] Yandex
https://tech.yandex.com/catboost/

[20] Ethem Alpaydin
Introduction to Machine Learning
2nd edition 2010.
The MIT Press Cambridge, Massachusetts London England
pp. 233-256

[21] Brian Ripley, William Venables
Package 'nnet', *Software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models*
Version 7.3-12
Publication date 2016-02-02

[22] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown
Auto-WEKA 2.0: Autonatic model selection and hyperparameter optimization in WEKA
Journal of Machine Learning Research 17(2016)

[23] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
third edition 2011.
Morgan Kaufmann Publishers inc. San Francisco, CA.
pp.350-355.

[24] Andrew Gelman, Yu-Sung Su, Masanao Yajima, Jennifer Hill, Maria Grazia Pittau, Jouni Kerman, Tian Zheng, Vincent Dori
Package 'arm', *Data Analysis Using Regression and Multilevel/Hierarchical Models*
Version: 1.9-3 Publication date: 2016-11-21

[25] Michal Majka
Package 'naivebayes', *High Performance Implementation of the Naive Bayes Algorithm*
Version: 0.9.2
Publication date: 2018-01-03.

[26] Ethem Alpaydin
Introduction to Machine Learning
2nd edition 2010.
The MIT Press Cambridge, Massachusetts London England
pp. 186-192

[27] Max Kuhn, Weston Steve, Culp Mark, Coulter Nathan, Ross Quinlan, RuleQuest Research
Package 'C50', *C5.0 decision trees and rule-based models for pattern recognition that extend the work of Quinlan*(1993, ISBN:1-55860-238-0).
Version: 0.1.1
Publication date 2017-11-20

[28] Torsen Hothorn, Kurt Hornik, Carolin Strobl, Achim Zeileis
Package 'party', *A Laboratory for Recursive Partitioning*
Version: 1.2-4
Publication date:2017-12-13

[29] Terry Therneau, Beth Atkinson, Brian Ripley
Package 'rpart', *Recursive pertitioning for classification regression and survival trees. An implementation of most of the functionality of the 1984 book by Breiman, Friedman, Olshen and Stone*
version: 4.1-13
Publication date 2018-02-23

[30] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
third edition 2011
Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 600.

[31] Ethem Alpaydin
Introduction to Machine Learning
2nd edition 2010. The MIT Press Cambridge, Massachusetts London England
pp. 209-218.

[32] Antonio Pedro Duarte Silva
Package 'HiDimDA, *High Dimensional Discriminant Analysis*
Version: 0.2-4
Publication date:2014-10-18

[33] Stephen Marsland
Machine Learning: An Algorithmic Perspective
1st Chapman & Hall/CRC 2009
pp. 153-155.

[34] Ian H. Witten, Eibe Frank, Mark A. Hall
Data Mining, Practical Machine Learning Tools and Techniques

3rd edition 2011.
Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 351-352.

[35] Stephen Marsland
Machine Learning: An Algorithmic Perspective
1st Chapman & Hall/CRC 2009
pp. 160-164.

[36] Leo Breiman
Bagging Predictors
Machine Learning 24,(1996)
Kluwer Academic Publishers, Boston.
pp 123-140.

[37] Stephen Marsland
Machine Learning: An Algorithmic Perspective
1st Chapman & Hall/CRC 2009
pp. 154-160.

[38] Leo Breiman
Machine Learning(2001) 1st Kluwer Academic Publishers 2001
pp.5-32.

[39] Ian H Witten, Eibe Frank, Mark A Hall
Data Mining, Practical Machine Learning Tools and Techniques 3rd edition 2011.
Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 369 - 370.

[40] Joäo Gama, Pavel Brazdil
Cascade Generalization
LIACC , FEP, University of Porto, Rua Campo Alegre, 823 4150 Porto, Portugal
Machine Learning, 41, 315343, 2000
c 2000 Kluwer Academic Publishers. Manufactured in The Netherlands.

[41] Andy Liaw
Package 'randomForest', *Breiman and Cutler's Random Forest for Classification and Regression*
Version 4.6-12
Publication date: 2015-10-06

[42] Jaak Simm, Ildefons Magrans de Abril
Package 'extratrees', *Extremely Randomized Trees(ExtraTrees)Method for Classification and Regression*
Version: 1.0.5
Publication date: 2014-12-27

[43] Greg Ridgeway
Package 'gbm', *Generalized Boosted Regression Models*
Version:2.1.3 Publication date:2017-03-21

[44] *http://topepo.github.io/caret/train-models-by-tag.html*

[45] Yoav Freund and Robert E. Schapire
Experiments with a new boosting algorithm Int. Conf. on Machine Learning
Morgan Kaufmann, 1996.
pp 148156.

[46] Klaus Schliep, Klaus Hechenbichler, Antoine Lizee
Package'kknn', *Weighted k-Nearest Neighbors*
Version: 1.3.1
Publication date: 2016-03-26

[47] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
third edition 2011. Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 355-364.

[48] Tom M. Mitchell
Machine Learning(1997)
McGraw Hill. pp. 274-282.
ISBN 0-07-042807-7 third edition 2011.

[49] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 61-83.

[50] Vanden Branden K, Hubert M
Robust classification in high dimensions based on the SIMCA method. 2006
Chemometrics and Intellegent Laboratory Systems 79:10–21 MODLEM
Tzu-Liang (Bill) Tseng, Chun-Che Huang, Kym Fraser and Hsien-Wei Ting
Rough set based rule induction in decision making using credible classification and preference from medical application perspective
Computer Methods and Programs in Biomedicine January 2016

[51] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
third edition 2011. Morgan Kaufmann Publishers inc. San Francisco, CA.
pp. 408-413.

[52] *https://www.rdocumentation.org/packages/e1071/versions/1.6-8/topics/svm*

[53] Chih-Chung Chang, Chih-Jen Lin(2001)
LIBSVM: A library for support vector machines
http://www.csie.ntu.edu.tw/ cjlin/libsvm/

[54] John Platt
Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines Technical Report MSR-TR-98-14 April 21, 1998

[55] Jerome Friedman, Trevor Hastie, Rob Tibshirani, Noah Simon, Balasubramanian Narasimhan, Junyang Qian
Package 'glmnet',*Lasso and Elasticnet Generalized Linear Models*
Version: 2.0-16
Publication date: 2018-03-12

[56] Martin Lauss
Package 'rocc', *ROC based classification*
Version:1.2
Publication date:2015-02-20

[57] Drew Conway, John Myles White
Machine Learning for hackers
OReilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.(2012)
pp.1-3

[58] R website *https://www.r-project.org/about.html*

[59] https://www.cs.waikato.ac.nz/ml/WEKA/

[60] Max Kuhn
Building predictive models in R using the caret package 2008.
Journal of Statistical Software, 28(5):
pp 126

[61] Jiawei Han, Micheline Kamber, Jian Pei
Data Mining Concepts and Techniques
third edition 2011. Morgan Kaufmann Publishers inc. San Francisco, CA. pp. 370-371.

[62] Ron Kohavi
A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection.
Proceedings of the 14th international joint conference on Artificial intelligence.
p.1137-1143, August 20-25 1995, Montreal, Quebec, Canada.

[63] Mary L. McHugh
Interrater reliability: the kappa statistic
Biochem Med. v22(3): 2012 Oct 15.

[64] Nuria Macià and Ester Bernadó-Mansilla.
Towards UCI+: a mindful repository design.
Information Sciences, 261(10):237262, 2014.

[65] Alexander K. Seewald & Johannes Furnkranz
An Evolution of Grading Classifiers
Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis
Springer Verlag ISBN:3-540-42581-0 pp 115-124

[66] Prem Melville and Raymond J. Mooney
Creating Diversity In Ensembles Using Artificial Data Journal of Information Fusion: Special Issue on Diversity in Multi Classifier Systems Vol. 6, 1 (2004), pp. 99-111.

[67] Prem Melville and Raymond J.Mooney
Constructing Diverse Classifier Ensembles Using Artificial Training Examples Proceedings of the 18th International Joint Conference on Artificial Intelligence 2003
publisher = Morgan Kaufmann Publishers Inc. San Francisco, CA, USA
pp 505–510

[68] Tuning values recommended by Yandex
https://tech.yandex.com/catboost/doc/dg/concepts/r-usages-examples-docpage/

Appendix A

Tuning Parameter Values

A.1 R

- **avNNet**
  1. size 1, 2, 3,..., 8, 9
  2. decay 0, 0.1, 0.2, 0.3, 0.01, 0.02, 0.03, 0.001, 0.002, 0.003, 0.0001
  3. bag true/false
- **bayesglm** no parameters
- **C5.0**
  1. winnow True/False
  2. trials 1, 2, 3,..., 19, 20
  3. model tree, rules
- **C5.0Rules** no parameters
- **C5.0Tree** no params
- **catboost**
  1. depth 2,4,6,8
  2. learning rate 0.01, 0.1
  3. iterations 100, 150, 200, 250
  4. l2_leaf_reg 1e-3, 1e-6, 2
  5. rsm 0.9, 0.95, 0.8
  6. border_count 255
- **cforest**
  1. mtry 1, 2, 3,...,$\sqrt{atributes}+1$

- **CSimca** no parameters
- **ctree**
  1. mincriterion 0.1, 0.2, 0.3, 0.4, 0.5, 0.01, 0.02, 0.03, 0.04, 0.05, 0.001, 0.002, 0.003, 0.004, 0.005
- **elm**
  1. nhid 1, 2, 3, ... ,19 , 20
  2. actfun ”purelin”, ”sin”, ”radbas”, ”tansig”, ”sig”, ”hardlim”, ”hardlims”, ”satlims”,”tribas”, ”poslin”
- **fda**
  1. degree 1, 1.5, 2, 2.5, 3
  2. nprune 1, 2, 3,..., 19, 20
- **gbm**
  1. shrinkage 0.1, 0.01, 0.03
  2. n.trees 50, 100, 150, 200, 250
  3. interaction depth 1, 2, 3, 4, 5
  4. n.minobsinnode 10, 20
- **glmnet**
  1. alpha 0, 1
  2. lambda sequence 0.0001->1, length=100
- **knn**
  1. k 1, 2, 3,..., 9, 10
- **lda** no parameters
- **LightGBM**
  1. num_iterations 150, 300
  2. learning_rate 0.1, 0.01
  3. num_leaves 8, 16, 32, 64
  4. min_gain_to_split 0.2, 0.4
  5. feature_fraction 0.75, 0.8, 0.95
  6. min_sum_hessian_in_leaf 0.05, 0.1
  7. min_data_in_leaf 1,3
  8. bagging_fraction 0.8, 0.9
  9. lambda_l2 0.2, 0.4

- **lvq**
  1. size 5, 10, 15, 20, 25, 30, 35, 40, 45, 50
  2. k 1, 2, 3,..., 9, 10
- **mlp**
  1. size 2, 3, 4,..., 18, 19, 20
- **mlpML**
  1. layer1 1, 3, 5, 7, 9
  2. layer2 1, 3, 5, 7, 9
  3. layer3 1, 3, 5, 7, 9
- **mlpWeightDecay**
  1. size 1,2,3,...,8,9
  2. decay 0, 0.1, 0.2, 0.3, 0.01, 0.02, 0.03, 0.001, 0.002, 0.003, 0.0001
- **naive_bayes**
  1. laplace 0, 0.5, 1
  2. usekernel TRUE, FALSE
  3. adjust 0,0.5,1
- **nb**
  1. usekernel TRUE, FALSE
  2. fL 0, 1
  3. adjust 0, 1
- **nnet**
  1. size 1, 2, 3,...,8, 9
  2. decay 0, 0.1, 0.2, 0.3, 0.01, 0.02, 0.03, 0.001, 0.002, 0.003, 0.0001
- **parRF**
  1. mtry 1, 2, 3,...,$\sqrt{atributes} + 1$
- **pcaNNet**
  1. size 1, 2, 3,..., 8, 9
  2. decay 0, 0.1, 0.2, 0.3, 0.01, 0.02, 0.03, 0.001, 0.002, 0.003, 0.0001
- **pda**
  1. lambda 0.001, 0.002, 0.003, 0.01, 0.02, 0.03, 0.1, 0.2, 0.3

- **rFerns**
  1. depth 1, 2, 3,..., 15, 16
- **rda**
  1. gamma 0,0.25,0.5,0.75,1
  2. lambda 0, 0.25, 0.5, 0.75, 1
- **rf**
  1. mtry $1, 2, 3, ..., \sqrt{atributes} + 1$
- **RFlda**
  1. q 1, 2, 3, 4, 5
- **rlda**
  1. estimator "Thomaz-Kitani-Gillies", "Schafer-Strimmer", "Moore-Penrose Pseudo-Inverse"
- **rocc**
  1. xgenes 1, 2, 3, 4,..., 97, 98, 99, 100
- **rpart**
  1. cp 0.1, 0.2, 0.3, 0.4, 0.5, 0.01, 0.02, 0.03, 0.04, 0.05, 0.001, 0.002, 0.003, 0.004, 0.005
- **sda**
  1. diagonal TRUE, FALSE
  2. lambda 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1
- **slda** no parameters
- **svmLinear**
  1. C 1, 1.5, 2, 2.5, 3, 3.5, 4
- **svmPoly**
  1. degree 1, 1.5, 2, 2.5, 3
  2. scale 0.01, 0.001, 0.1, 1, 10
  3. C 0.25, 0.5, 1, 2, 3, 4
- **svmRadial**
  1. C 1, 1.5, 2, 2.5, 3, 3.5, 4
  2. sigma 0.1, 0.01, 0.02, 0.03, 0.001, 0.002, 0.003, 0.15, 0.015, 0.025

- **svmRadialCost**
    1. C 1, 1.5, 2, 2.5, 3, 3.5, 4
- **treebag** no parameters
- **wsrf**
    1. mtry 1, 2, 3...,$\sqrt{atributes} + 1$
- **xgbLinear**
    1. eta 0.1, 0.01
    2. alpha 0, 0.1, 0.01, 0.001
    3. lambda 0,0.1, 0.01, 0.001
    4. nrounds 50, 100, 150, 250
- **xgbtree**
    1. eta 0.1, 0.2, 0.3, 0.02, 0.4, 0.03
    2. max_depth 1, 2, 3, 4, 5, 6, 7, 8
    3. gamma 0
    4. colsample_bytree 0.5, 0.6, 0.7, 0.8
    5. min_child_weight 1, 2
    6. subsample 0.5, 0.625, 0.75, 0.875
    7. nrounds 50, 100, 150, 200, 250

A.2 WEKA

- **A1DE**
    1. frequencyLimit 1,2,3....10
    2. subsumptionResolution true, false
    3. weightedAODE true, false
- **AdaBoostM1**
    1. numIterations 10,25,40,75,100
    2. useResampling true, false
    3. classifier SimpleCart
    4. classifier.useOneSE true, false
    5. classifier.numFoldsPruning 3,4,5
- **AutoWEKAClassifier**

1. timeLimit 45
2. parallelRuns 8

- **bagging(ExtraTree)**

  1. classifier ExtraTree
  2. numIterations 500 750 1000
  3. **representCopiesUsingWeights** true, false
  4. numExecutionSlots 8

- **BFTree**

  1. minNumObj 2,3,4,5
  2. numFoldsPruning 2,3,4...12
  3. pruningStrategy 0,1,2
  4. useGini true, false
  5. useOneSE true, false

- **CascadeGeneralization(combo1)**

  1. useLogOdds false
  2. metaClassifier.minNumInstances 5,10,15
  3. numFolds 10
  4. classifiers[1].minNum 1.0
  5. classifiers[1].minVarianceProp 0.1,0.01
  6. classifiers[3].ridge 1.0E-7, 1.0E-9
  7. classifiers[4].minNum 2,5,8
  8. classifiers[4].minVarianceProp 0.01,0.01
  9. classifiers[4].numFolds 3,4,5,6
  10. classifiers [IB1, RandomTree, NaiveBayes, LDA, REPTree]

- **CDT**

  1. SValue 1.0,1.0...5.0
  2. minVarianceProp 0.0005,0.0010....0.0025
  3. noPruning true, false

- **ConjunctiveRule**

  1. folds 3,4,5
  2. minNo 2,3,4,5
  3. numAntds -1,1,2,3

- **DecisionStump** No parameters
- **DecisionTable**
   1. evaluationMeasure acc,rmse,mae,auc
   2. useIBk true,false
   3. search BestFirst, GreedyStepWise
   4. search.direction 0,1,2
   5. search.searchTermination 5
   6. search
   7. search.conservativeForwardSelection true, false
- **Elman**
   1. trainingTime 600,1200,1800 item hiddenLayers 2,3,5,8
   2. learningRate 0.5,0.4,0.3,0.2,0.1
   3. momentum 0.01,0.05, 0.001,0.005,0.01,0.05,0.1
   4. nominalToBinaryFilter false
   5. normalizeAttributes false
   6. normalizeNumericClass false
   7. decay false
- **ExtraTree** No parameters
- **ForestPA**
   1. numberOfTrees 100
   2. simpleCartMinimumRecords 2,3,4,5
   3. simpleCartPruningFolds 2,3,4,5
- **FT**
   1. numBoostingIterations 5,10,15,20,25
   2. modelType 0,1,2
   3. minNumInstances 5,7,9...25
   4. weightTrimBeta 0,0.01,0.02
   5. useAIC true,false
- **FURIA**
   1. folds 2,3,4,5
   2. minNo 2,3,4,5
   3. optimizations 2,4...10

4. TNorm 0,1
5. uncovAction 0,1,2

- **Grading(combo1)[IBk]**

  1. numFolds 10
  2. classifiers[0].minNum 1.0
  3. classifiers[0].minVarianceProp 0.1,0.01
  4. classifiers[1].ridge 1.0E-7, 1.0E-9
  5. classifiers[2].minNum 2,5,8
  6. classifiers[2].minVarianceProp 0.1,0.01
  7. classifiers[2].numFolds 3,4,5,6
  8. metaClassifierIBk
  9. metaClassifier.KNN 10
  10. metaClassifier.distanceWeighting Weight by 1-distance
  11. classifiers [RandomTree, LDA, REPTree, NaiveBayes, IB1]

- **Grading(combo1)[ZeroR]**

  1. numFolds 10
  2. classifiers[0].minNum 1.0
  3. classifiers[0].minVarianceProp 0.1,0.01
  4. classifiers[1].ridge 1.0E-7, 1.0E-9
  5. classifiers[2].minNum 2,5,8
  6. classifiers[2].minVarianceProp 0.1,0.01
  7. classifiers[2].numFolds 3,4,5,6
  8. classifiers [RandomTree, LDA, REPTree, NaiveBayes, IB1]

- **HoeffdingTree**

  1. leafPredictionStrategy 0,1,2
  2. splitCriterion 0,1,2
  3. splitConfidence 1.0E-8,5.0E-8,1.0E-7,5.0E-7,1.0E-6
  4. hoeffdingTieThreshold 0.03, 0.035, 0.04...0.1
  5. minimumFractionOfWeightInfoGain 0.01,0.015,0.02...0.05
  6. gracePeriod 50,100,150...500
  7. naiveBayesPredictionThreshold 0,10,50,100,200

- **HoeffdingTree** No parameters
- **IB1** No parameters

- **IBk**
  1. KNN 1,25,50,100
  2. crossValidate true, false
  3. distanceWeighting No distance weighting, Weight by 1/distance, Weight by 1-distance
  4. nearestNeighbourSearchAlgorithm BallTree,CoverTree,KDTree
  5. nearestNeighbourSearchAlgorithm.ballTreeConstructor BottomUpConstructor, MiddleOutConstructor, TopDownConstructor
  6. nearestNeighbourSearchAlgorithm.nodeSplitter MedianOfWidestDimension,MidPointOfWidest
  7. nearestNeighbourSearchAlgorithm.distanceFunction EuclideanDistance
  8. nearestNeighbourSearchAlgorithm.skipIdentical true, false

- **J48**
  1. confidence 0.15,0.20...0.55
  2. minNumObj 2,3,4,5
  3. reducedErrorPruning true
  4. useLaplace true,false
  5. useMDLcorrection true,false

- **J48Consolidate**
  1. RMbagSizePercent -2,-1
  2. RMnewDistrMinClass 50
  3. RMnumberSamples 75,85,90
  4. RMnumberSamplesHowToSet -C
  5. RMreplacement false
  6. collapseTree true,false
  7. confidenceFactor 0.10,0.13...0.25
  8. doNotMakeSplitPointActualValue true,false
  9. minNumObj 2,4,6
  10. subtreeRaising true,false
  11. useLaplace true,false
  12. useMDLcorrection true,false

- **J48Graft**
  1. confidenceFactor 0.15,0.17...0.25
  2. minNumObj 2,3,4,5
  3. subtreeRaising true

4. relabeltrue,false
5. useLaplace true,false

- **JRip**

  1. folds 2,3,4,5
  2. minNo 2,3,4,5
  3. optimizations 2,4...10
  4. usePruningtrue,false

- **LadTree**

  1. numOfBoostingIterations 10,25,50,100

- **LDA**

  1. ridge 1.0E-5,5.0E-5,1.0E-6,5.0E-6,1.0E-7

- **libLINEAR**

  1. SVMType 0,1...7
  2. eps 0.005,0.01...0.05
  3. cost 1,2,3,4

- **libSVM**

  1. cacheSize 400
  2. SVMType 0,1
  3. cost 0.5,1...2.5
  4. degree 2,2.5...5
  5. kernelType 0,1,2,3
  6. nu 0.3,0.4...0.7
  7. probabilityEstimates true, false
  8. shrinking true,false

- **LogitBoost**

  1. numIterations 100,200...500
  2. ZMax 2,2.5...5
  3. shrinkage 0.05,0.1,0.2
  4. useResamplingtrue,false
  5. useEstimatedPriors true,false

- **MultiBoostAB**

   1. useResampling true,false
   2. numSubCmtys 3,5,10
   3. numIterations 25,50,100
   4. classifierSimpleCart
   5. classifier.useOneSE true,false
   6. classifier.numFoldsPruning 3,4,5

- **MultiLayerPerceptron**
   1. learningRate 0.2,0.3,0.5
   2. momentum 0.1,0.15...0.3
   3. trainingTime 500,750,1000
   4. hiddenLayersa,5,10,25
   5. decay false

- **MultiScheme(combo1)**
   1. numFolds 0,5,10
   2. classifiers[0].minNum 1.0
   3. classifiers[0].minVarianceProp 0.1,0.01
   4. classifiers[1].ridge 1.0E-7,1.0E-9
   5. classifiers[2].minNum 2,5,8
   6. classifiers[2].minVarianceProp 0.1,0.01
   7. classifiers[2].numFolds 3, 4, 5, 6
   8. classifiers [RandomTree, LDA, REPTree, NaiveBayes, IB1]

- **NaiveBayes**
   1. useKernelEstimator true, false
   2. useSupervisedDiscretizatiom true, false

- **NaiveBayesUpdateable**
   1. useKernelEstimator true, false
   2. useSupervisedDiscretizatiom true, false

- **NBTree** No parameters

- **NNge**
   1. numAttemptsOfGeneOption 5,9...25
   2. numFoldersMIOption 5,9...25

- **OLM**

    1. resolutionMode 0,1,2,3
    2. classificationMode 1,0

- **OneR**
    1. minBucketSize 5,9...25

- **PART**
    1. binarySplits true, false
    2. confidenceFactor 0.15,0.2...0.5
    3. minNumObj 2,3,4,5
    4. reducedErrorPruning true, false
    5. unpruned true, false
    6. useMDLcorrection true, false

- **PCT**
    1. minNumInstances 5,15,25,50,100
    2. useAIC true, false

- **QDA**
    1. ridge 1.0E-7,5.0E-7,1.0E-6,5.0E-6,1.0E-5

- **RandomCommittee(ExtraTree)**
    1. numIterations 500,1000,2500

- **RandomCommittee(RandomTree)**
    1. numIterations 500,1000,2500
    2. classifier.minNum 1,2,3,5
    3. classifier.minVarianceProp 0.0001, 0.0005, 0.001, 0.005, 0.01

- **RandomForest**
    1. numIterations 500,1000,2500
    2. classifier.minNum 1,2...5
    3. classifier.minVarianceProp 0.0001, 0.0005, 0.001, 0.005, 0.01

- **RandomSubSpace(ExtraTree)**
    1. numIterations 100
    2. classifier ExtraTree

- **RandomSubSpace(RandomTree)**

1. numIterations 100
2. classifier RandomTree
3. classifier.minNum 1,2
4. classifier.minVarianceProp 0.0005, 0.001...0.005

- **RandomSubSpace(REPTree)**
    1. numIterations 100
    2. classifier REPTree
    3. classifier.minNum 2,3,4,5
    4. classifier.minVarianceProp 0.01, 0.001, 0.005, 0.0001, 0.0005
    5. classifier.numFolds 2,4...10
    6. classifier.spreadInitialCount
    7. classifier-noPruning true
- **RandomTree**
    1. breakTiesRandomly true, false
    2. minNum 1, 2
    3. minVarianceProp 0.0005, 0.001...0.0025
- **REPTree**
    1. minNum 2, 3, 4, 5
    2. minVarianceProp 0.01, 0.001, 0.0001, 0.0001
    3. numFolds 4,5...10
    4. spreadInitialCount true, false
    5. noPruning true
- **Ridor**
    1. folds 3, 4, 5
    2. shuffle 1, 2 ... 5
    3. minNo 2,3...5
    4. wholeDataErr true, false
- **SimpleCart**
    1. minNumObj 2, 3, 4
    2. numFoldsPruning 3, 4, 5
    3. useOneSE true, false
- **SimpleLogistic**

   1. maxBoostingIterations 100, 300 ... 900
   2. heuristicStop 10, 28 ... 100
   3. weightTrimBeta 0, 0.5 ... 2.0
   4. useAIC true, false
   5. errorOnProbabilities true, false

- **SMO**
   1. c 1.0, 2.5, 5.0, 10.0
   2. calibrator.ridge 1.0E-7, 5.0E-7, 1.0E-8, 5.0E-8, 1.0E-9
   3. calibrator.useConjugateGradientDescent true, false
   4. kernel.cacheSize 500287
   5. kernel PolyKernel, Puk, RBFKernel,
   6. kernel.exponent 2.0
   7. kernel.useLowerOrdertrue, false
   8. kernel.omega 1.0, 2.5, 5.0, 10.0
   9. kernel.sigma 1, 2.5, 5, 10
   10. kernel.gamma 0.03, 0.02, 0.01, 0.008, 0.005
   11. filterType No normalization/standardization

- **StackingC(combo1)**
   1. metaClassifier.attributeSelectionMethod 0, 1, 2
   2. metaClassifier.eliminateColinearAttributes true, false
   3. metaClassifier.ridge 1.0E-5, 1.0E-7
   4. classifiers[0].minNum 1
   5. classifiers[0].minVarianceProp 0.1, 0.01
   6. classifiers[1].ridge 1.0E-7, 1.0E-9
   7. classifiers[2].minNum 2,5,8
   8. classifiers[2].minVarianceProp 0.1, 0.01
   9. classifiers[2].numFolds 3, 4, 5, 6
   10. classifiers [RandomTree, LDA, REPTree, NaiveBayes, IB1]

- **StackingC(combo2)**
   1. metaClassifier.attributeSelectionMethod 0, 1, 2
   2. metaClassifier.eliminateColinearAttributes true, false
   3. metaClassifier.ridge 1.0E-5, 1.0E-7
   4. classifiers[0].minNum 1, 2

5. classifiers[0].minVarianceProp 0.002, 0.004
6. classifiers[1].ridge 1.0E-5, 1.0E-7
7. classifiers[3].SVMType 0, 1, 2, 5, 6, 7
8. classifiers[3].eps 0.02, 0.04
9. classifiers[3].epsilonParameter 0.2, 0.4
10. classifiers[3].cost 2, 4
11. classifiers [RandomTree, ExtraTree, LDA, libLINEAR]

- **Vfdr**

  1. gracePeriod 10, 15 ... 50
  2. nbWeightTreshold 5, 7 ... 25
  3. predictionStrategy 0, 1
  4. hoeffdingConfidence 1.0E-8, 5.0E-7, 1.0E-7, 5.0E-6, 1.0E-6, 5.0E-5, 1.0E-5

- **VFI**

  1. bias 0.2, 0.3 ... 1.2
  2. weightByConfidence true, false

- **Vote(combo1)**

  1. classifiers[0].minNum 1
  2. classifiers[0].minVarianceProp 0.1, 0.01
  3. classifiers[1].ridge 1.0E-7, 1.0E-9
  4. classifiers[2].minNum 2, 5, 8
  5. classifiers[2].minVarianceProp" 0.1, 0.01
  6. classifiers[2].numFolds 3, 4, 5, 6
  7. classifiers

- **Vote(combo2)**

  1. classifiers[0].minNum 1
  2. classifiers[0].minVarianceProp 0.1, 0.01
  3. classifiers[1].ridge 1.0E-7, 1.0E-9
  4. classifiers[2].minNum 2, 5, 8
  5. classifiers[2].minVarianceProp" 0.1, 0.01
  6. classifiers[2].numFolds 3, 4, 5, 6 item
  7. [RandomTree, LDA, REPTree, NaiveBayes, IB1]

- **Vote(combo2)**

1. classifiers[0].minNum 1
2. classifiers[0].minVarianceProp 0.1, 0.01
3. classifiers[2].ridge 1.0E-5, 1.0E-7
4. classifiers[3].SVMType 0, 1, 2, 5, 6, 7
5. classifiers[3].eps 0.02, 0.04
6. classifiers[3].epsilonParameter 0.2, 0.4
7. classifiers[3].cost 2, 4
8. classifiers [RandomTree, ExtraTree, LDA, libLINEAR]

Appendix B

R code

B.1 R script for tuning

The tuning script can be used by calling the tunec function, it needs the number of tuning parameters, the method(algorithm) and a tunegrid for this algorithm. In this example it uses svmPoly, to change to RFlda call the function, e.g. tunec(1,f,train_control,”RFlda”,expand.grid(.q=c(1:5))).

```
# more libraries may be required, depending on the classification algorithm.
library(caret)
library(RWeka)
library(doParallel)
#where the files are
setwd("D:/Datasets")
#find all .arff files, recursive
files<-list.files(pattern=".arff$", recursive=TRUE)
#ten fold cross-validation
train_control<-trainControl(method="cv", number=10)

tunec<-function(tune,data,cv,mt,tg)
{
#let’s use 8 threads
cl <- makeCluster(8)
registerDoParallel(cl)
print(sprintf("starting dataset %s.",data))

#same seed for all methods...
set.seed(765)
dataset<-read.arff(sprintf("%s",data))
model<-train(dataset[,1:(ncol(dataset)-1)],dataset[,ncol(dataset)],
trControl=cv,method=mt,tuneGrid=tg)
stopCluster(cl)
#let’s order on accuracy, best first
res<-model$results[order(-model$results$Accuracy),]

#before attempting this, create a folder, here called svmPoly,
#and a file params.txt with the line "method,Software,Type",
  #in this case svmPoly,R,SVM
write(sprintf("’%s’,’degree=%s, scale=%s, C=%s’",shortname(data),
model$bestTune$degree,model$bestTune$scale,model$bestTune$C),
sprintf("C:\\Users\\tom\\Desktop\\%s\\params.txt",mt),append=TRUE)

write(sprintf("%s,%s",head(res[1+tune],1),head(res[2+tune],1)),
sprintf("C:\\Users\\tom\\Desktop\\%s\\%s.txt",mt,shortname(data)),append=TRUE)
print(res)
print(sprintf("finished dataset %s.",data))
}
#datafiles are in their own folder, e.g. abalone/abalone.arff
# function removes all infront of / and the ending, e.g.
#shortname("abalone/abalone.arff")="abalone"
shortname<-function(instr)
{
return(gsub(".arff","",gsub("^.*\\/","",instr)))
}
#for each .arff file
for(f in files)
{
tunec(3,f,train_control,"svmPoly",expand.grid(.degree=c(1,1.5,2,2.5,3),
.scale=c(0.01,0.001,0.1,1,10),.C=c(0.25,0.5,1,2,3,4)))

}
```

B.2 R script for repeated cross-validation

```
library(caret)
library(RWeka)
library(doParallel)
setwd("D:/Datasets")
#this time we repeat the 10 fold cross-validation 99 times.
tr<-trainControl(method="repeatedcv",number=10,repeats=99)
#this script has to adapted to each method, the less parameters, the less work.
repcv<-function(data,mt,cv,d,s,c)
{
#8 threads again
cl <- makeCluster(8)
registerDoParallel(cl)
print(sprintf("starting %s",data))
set.seed(765)
dataset<-read.arff(sprintf("%s",data))
mod<-train(dataset[,1:(ncol(dataset)-1)],dataset[,ncol(dataset)],
trControl=cv,method=mt,tuneGrid=expand.grid(.degree=d,.scale=s,.C=c))
stopCluster(cl)
il<-sprintf("C:\\Users\\tom\\Desktop\\%s\\%s.txt",mt,shortname(data))
sink(il,append=TRUE)
ctr=1
ct=1
k=0
a=0
#we have 990 values for kappa and accuracy(99x10 folds)
#we add them 10 times and divide by 10
while(ctr<991)
{
k=k+mod$resample[ctr,2]
a=a+mod$resample[ctr,1]
if(ct==10)
{
cat(sprintf("\n%s,%s",toString(a/10),toString(k/10)))
ct=0
k=0
a=0
}
ctr=ctr+1
ct=ct+1
}
sink()
print(sprintf("end %s",data))
}
shortname<-function(instr)
{
return(gsub(".arff","",gsub("^.*\\/","",instr)))
}
finishstuff<-function(tr)
{
#one line per dataset
repcv("acute-inflammation/acute-inflammation.arff","svmPoly",tr,1,0.1,0.25)
repcv("acute-nephritis/acute-nephritis.arff","svmPoly",tr,1,1,0.25)
}
finishstuff(tr)
```

Appendix C

Java code

The java code is too large to print. See the usb. The following is the modifications we made to the different packages.
Core WEKA package
Modified normalize method for weka.core.Utils

```
public static void normalize(double[] doubles, double sum)
{
if (Double.isNaN(sum))
{
//throw new IllegalArgumentException("Can’t normalize array. Sum is NaN.");
sum = 0;
}
if (sum == 0)
{
// Maybe this should just be a return.
//throw new IllegalArgumentException("Can’t normalize array. Sum is zero.");
  return;
}
for(int i = 0; i < doubles.length; i++)
 {
doubles[i] /= sum;
}
}
```

Fuzzy Unordered Rule Induction Package
Modification of FURIAs rulesetForOneClass method, in the optimization stage

```
if (Double.isNaN(repDL) || Double.isInfinite(repDL))
{
repDL = Double.MIN_VALUE;
//throw new Exception("Should never happen: repDL" +
"in optmz. stage NaN or " + "infinite!");
}
```

We added this method to the Vfdr class.

```
//Get method for the HoeffdingTieThreshold
public double getHoeffdingTieThreshold()
{
return m_hoeffdingTieThreshold;
}
```

Vfdr Package
Modified version of Vfdrs setOptions method

```
@Override public void setOptions(String[] options) throws Exception
{
String gracePeriod = Utils.getOption('G', options);
if (gracePeriod.length() != 0)
{
     setGracePeriod(Integer.parseInt(gracePeriod));
}
String nbThreshold = Utils.getOption('N', options);
if (nbThreshold.length() != 0)
{
setNBWeightThreshold(Double.parseDouble(nbThreshold));
}
String predictionStrat = Utils.getOption('R', options);
if (predictionStrat.length() != 0)
{
setPredictionStrategy(Integer.parseInt(predictionStrat));
}
//Previously mapped to the hoeffdingConfidence
String hoeffdingTieThreshold = Utils.getOption('T', options);
if (hoeffdingTieThreshold.length() != 0)
{
 setHoeffdingTieThreshold(Double.parseDouble(hoeffdingTieThreshold));
}
//The C option was not previously parsed
String hoeffdingConfidence = Utils.getOption('C', options);
if (hoeffdingConfidence.length() != 0)
{
setHoeffdingConfidence(Double.parseDouble(hoeffdingConfidence));
}
setOrderedSet(Utils.getFlag('O', options));
super.setOptions(options);
Utils.checkForRemainingOptions(options);
}
```

MODLEM Package
The modified setOptions method for MODLEM.

```
public void setOptions(String[] options)throws Exception
{
String value = Utils.getOption("RT", options);
if (value.toLowerCase().equals("c") || value.equals("0"))
{
setRulesType(new SelectedTag(0, RULES_TYPES));
}
else if (value.toLowerCase().equals("l") || value.equals("1"))
{
setRulesType(new SelectedTag(1, RULES_TYPES));
}
else if (value.toLowerCase().equals("u") || value.equals("2"))
{
setRulesType(new SelectedTag(2, RULES_TYPES));
}
else
{
//throw new IllegalArgumentException("Invalid rules approximation type");
}
value = Utils.getOption("CM", options);
if (value.toLowerCase().equals("e") || value.equals("0"))
{       setConditionsMeasure(new SelectedTag(0, CONDITIONS_MEASURES));
}
else if (value.toLowerCase().equals("l") || value.equals("1"))
{
setConditionsMeasure(new SelectedTag(1, CONDITIONS_MEASURES));
}
else
{
//throw new IllegalArgumentException("Invalid conditions measure");
}     value = Utils.getOption("CS", options);
if (value.toLowerCase().equals("n"))
{
setClassificationStrategy(new SelectedTag(1, CLASSIFICATION_STRATEGIES));
}
else if (value.toLowerCase().equals("s") || value.equals("0"))
{
setClassificationStrategy(new SelectedTag(0, CLASSIFICATION_STRATEGIES));
}     else if (value.toLowerCase().equals("d") || value.equals("2"))
{
 setClassificationStrategy(new SelectedTag(2, CLASSIFICATION_STRATEGIES));
}
else if (value.toLowerCase().equals("m") || value.equals("8"))
{
setClassificationStrategy(new SelectedTag(8, CLASSIFICATION_STRATEGIES));
}
else if (value.toLowerCase().equals("c") || value.equals("6"))
{
setClassificationStrategy(new SelectedTag(6, CLASSIFICATION_STRATEGIES));
}
else if (value.toLowerCase().equals("l") || value.equals("3"))
{
setClassificationStrategy(new SelectedTag(3, CLASSIFICATION_STRATEGIES));
}
else if (value.toLowerCase().equals("1") || value.equals("1"))
{
setClassificationStrategy(new SelectedTag(1, CLASSIFICATION_STRATEGIES));
}
else
{
//throw new IllegalArgumentException("Invalid classification strategy");
}
value = Utils.getOption("AS", options);
if (value.toLowerCase().equals("f") || value.toLowerCase().equals("0"))
{
setMatchingType(new SelectedTag(0, MATCHING_TYPES));
}
else if (value.toLowerCase().equals("m") || value.toLowerCase().equals("1"))
{
setMatchingType(new SelectedTag(1, MATCHING_TYPES));
}
else if (value.toLowerCase().equals("a") || value.toLowerCase().equals("2"))
{
setMatchingType(new SelectedTag(2, MATCHING_TYPES));
}
else
{
//throw new IllegalArgumentException("Invalid matching type");
}
super.setOptions(options);
}
```