CLASSIFIER OF ASTROPHYSICS DATA

KLASIFIKÁTOR ASTROFYZIKÁLNÍCH DAT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE VOJTĚCH RYLKO
AUTHOR

VEDOUCÍ PRÁCE Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2012

Abstrakt

Cílem této práce je implementace algoritmu pro dolování z dat pro použití v astrofyzice.

V práci jsou představeny základní pojmy a principy dolování z dat. Zejména jeho obecná definice, rozlišení mezi klasifikací a regresí a vyhodnocování přesnosti modelu. Text se zabývá převážně učením s učitelem.

Blíže představeny jsou algoritmy založené na rozhodovacích stromech. Je definován rozhodovací strom jako model a uveden obecný algoritmus pro tvorbu rozhodovacích stromů z dat. Jsou diskutována různá kritéria dělení v uzlech (zejména založená na etropii), kritéria pro ukončení růstu a ořezávání stromů. Pro ilustraci jsou uvedeny vybrané algoritmy – ID3, CART, RainForest a BOAT.

Na dříve uvedených informacích je založena kapitola o souborech rozhodovacích stromů. Zabývá se základními způsoby jejich kombinací *(bagging* a *arcing)*. Detailněji je popsán obecný algoritmus náhodných lesů a RandomForest^{TM} jako příklad jeho praktické realizace.

Na základě srovnání algoritmů a provedených experimentů v literatuře jsou k implementaci vybrány náhodné lesy. Implementovaný algoritmus je detailněji popsán – k dělení uzlů používá Gini entropie a průměrnou kvadratickou chybu, ignoruje chybějící hodnoty a pro kombinaci výstupů jednotlivých stromů používá většinové hlasování / průměr. Jako formát vstupních a výstupních dat je zvolena podmnožina ARFF formátu. Architektura implementace je ilustrována UML diagramy s popisujícím komentářem. Jednotlivé aspekty implementace jsou stručně popsány – implementačním jazykem je C++11, je využívána knihovna Boost (zejména chytré ukazatele, serializace, nastavení parametrů a konfigurační soubory, . . .) společně s dalšími volně dostupnými knihovnami (google-glog pro logování, googletest pro jednotkové testování, . . .). Grafického výstupu je dosaženo tiskem modelu náhodného lesu do XML souboru a jeho transformací skriptem do jazyka DOT.

Pro ověření validity a vlastností implementace a jejího srovnání s jinými implementacemi náhodných stromů (Waffles, RF-ACE a R – balíček randomForest) jsou navrženy, popsány a provedeny exprimenty: klasifikace astronomických těles na základě barevných indexů, regrese rudého posuvu na základě barevných indexů, osm klasifikačních a pět regresních experimentů na datech z UCI repository. Průběh experimentů je plně automatizován skripty (Bash, Python a R) a je měřena doba učení modelů. Z výsledků experimentů vyplývá, že autorova implementace si vedla výborně při klasifikaci a průměrně při regresi; z časového hlediska měla problémy při datech s mnoha instancemi.

Výsledkem práce je zdokumentovaná, snadno rozšiřitelná implementace náhodných lesů v jazyce C++ s grafickým znázorněním modelu, mnoha možnostmi nastavení a experimentálně ověřenou funkčností. Diskuze o dalším možném pokračování projektu se zabývá zejména odstraněním problemů s časovou náročností a přidáním nových funkcionalit.

Abstract

This bachelor thesis describes selection, design and implementation of a data mining algoritm for astrophysical usage. The implementation of the random decision forests algorithm in C++ is evaluated on two astrophysical and some general experiments. Experiments are both classification and regression with time measuring. For comparison another three implementations are evaluated. The resulting implementation shows good results mainly in classification.

Klíčová slova

Dolování z dat, strojové učení, rozhodovací stromy, náhodné lesy, astrofyzika, astroinformatika, C++.

Keywords

Data mining, machine learning, decision trees, random decision forests, astrophysics, astroinformatics, C++.

Citation

Vojtěch Rylko: Classifier of astrophysics data, bachelor's thesis, Brno, FIT BUT, 2012

Classifier of astrophysics data

Declaration

I declare that this thesis is my own work that has been created under the supervision of Doc. Ing. Jaroslav Zendulka, CSc., and in consultation with Dr. Petr Škoda, Ph.D. All sources and literature that I have used during elaboration of the thesis are correctly cited with complete reference to the corresponding sources.

....
Vojtěch Rylko
May 9, 2012

Acknowledgements

I would like to thank to Doc. Ing. Jaroslav Zendulka, CSc., for his lead and valuable suggestions. I would like to also thank to Dr. Petr Škoda, Ph.D, and Ing. Jaroslav Vážný for their help with astrophysical topics.

Contents

Chapter 1

Introductions

"But the cleverest algorithms are no substitute for human intelligence and knowledge of the data in the problem." (Leo Breiman and Adele Cutler)

Astronomy has became a data rich science. The evolution of instruments and detectors caused exponential growth of astronomical data. For example the Large Synoptic Survey Telescope will produce data flow of about 20-30 TB per night [23]. Need for effective and useful exploration of the data leads to birth of a new discipline – Astroinformatics. Astroinformatics applies data mining techniques on massive astronomical data sets.

The main goal of this thesis is to select and implement data mining algorithm based on decision trees suitable for astrophysical usage. Output of the thesis will be functional program which will be able to perform classification and regression and which will meet the requirements given by external consultant.

The thesis is divided into seven chapters. Chapter 2 introduces main concepts and terms. Chapter 3 describes data mining methods based on decision trees. Chapter 4 studies models based on ensembles of decision trees. The random decision forests algorithm is described in detail in subsection 4.4.1. In chapter 5 we presents design and implementation of the random decision forests algorithm. The program is console application implemented in C++11. It uses Boost library and some other free libraries. Trained models are de/serialized in XML format. The trained model can be visualized. Chapter 6 demonstrates two astrophysical and some validation experiments. It also compare my implementation with related implementations. Last chapter (7) discusses achieved results and opportunities for further development.

Chapter 2

Data mining and classification

Data Mining (DM) is part of the more general process called Knowledge Discovery, Knowledge Discovery from Data, Knowledge Discovery in Data or KDD (some authors call KDD often Data Mining; this may be confusing) [18, 2]. KDD has evolved from the intersection of such fields as databases, machine learning, pattern recognition, data visualization and others [13]. We define KDD [14] as "non-trivial extraction of implicit, previously unknown and potentially useful information from data". This interactive and iterative process consists of following steps [13]:

1. Learning the application domain
2. Creating a target dataset
3. Data cleaning and preprocessing
4. Data reduction and projection
5. Choosing the function of data mining
6. Choosing the data mining algorithm(s)
7. Data mining
8. Interpretation
9. Using discovered knowledge

Data Mining involves fitting *models to* or determining *patterns from* observed data. Most data mining algorithms consist of three components: the model, the preference criterion and the search algorithm. [13]

The model has some *function* and the representational form (we focus on use of tree-like structures as models) and it contains parameters that are to be determined from data. The model should reflect useful knowledge.

The preference criterion is a basis for preference of one model or parameters over another on the given data. It is usually measure in form of goodness-of-fit of the model on the data including some term to avoid overfitting.

The search algorithm is used to find particular models and parameters.

Then data mining algorithm is usually instantiation of the model, the preference criterion and the search algorithm. (For example model based on decision tree, with classification

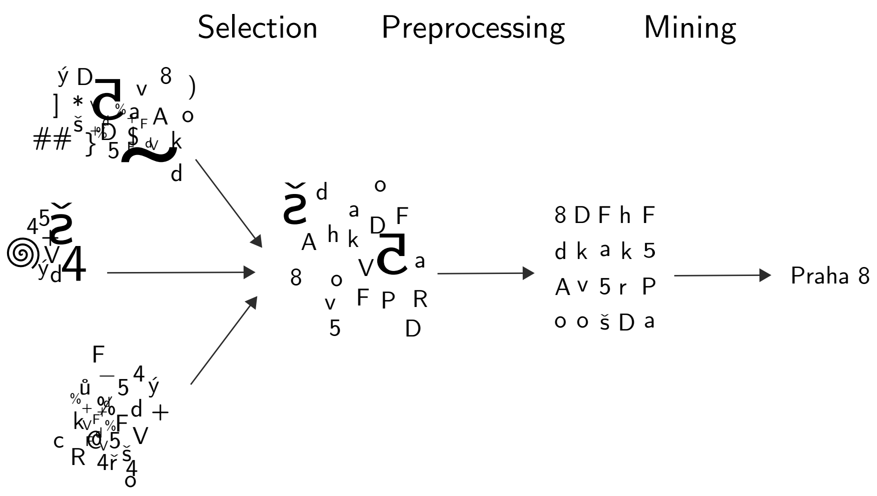


Figure 2.1: Simple illustration of the knowledge discovery process

function, with model preference based on generalization error, determined by greedy search using a heuristic function.)

In literature this three components are often mixed up in a description of a particular algorithm.

2.1 Model functions

Model function specify the kind of knowledge to be mined. The common functions in data mining practice are classification, regression, clustering, outlier analysis, characterization, discrimination, association analysis, ... We are interested in classification and regression.

2.2 Classification and regression

Classification and regression (or numerical prediction) are one of the most common tasks in DM. Classification predicts (or maps) a data item into one of several predefined categorical[1] labels (or classes). Regression predicts a numerical value.

Both learning method are *supervised.* This mean that the class label (or predicted attribute for regression) for each training tuple *is provided* (in contrast with *unsupervised learning* where is not). We call such data *labeled data.* Table 2.1 shows example of such labeled data.

Let $X_1, \ldots, X_m, Y$ be a random variables where X_i is attribute variable and has domain $\mathrm{Dom}(X_i)$. The number of attribute variables is m. The random variable Y has domain $\mathrm{Dom}(Y) = \{1, ..., k\}$ for classification problem or $\mathrm{Dom}(Y) = \mathbb{R}$ for regression problem. We call Y the class label or predicted attribute.

A classifier $\mathcal{C}$ is a function $\mathcal{C} : \mathrm{Dom}(X_1) \times \ldots \times \mathrm{Dom}(X_m) \to \mathrm{Dom}(Y)$.

[1]Categorical values are discrete and unordered.

sepal length	sepal width	petal length	petal width	class of iris plant
5.1	3.5	1.4	0.2	Iris-setosa
4.9	3.0	1.4	0.2	Iris-setosa
5.7	2.8	4.1	1.3	Iris-versicolor
6.3	3.3	6.0	2.5	Iris-virginica
5.8	2.7	5.1	1.9	Iris-virginica
...	...	...	...	...
5.9	3.0	5.1	1.8	Iris-virginica

Table 2.1: Iris classification data set [3]

2.3 Model representation

There exists many models include decision trees and rules, neural networks, Bayesian networks, support vector machines, hidden Markov model and others. Model representation determines both the flexibility of model in describing the data and the interpretability of the model by humans. We are interested in decision trees.

2.4 Attributes

Attributes as well as class labels may be *categorical, ordinal or numerical.* Some preprocessing of input data is often recommended. It may include for example discretization of continuous attributes, outlier detection, cleaning, feature normalization (scaling), filling missing values etc. Good visualization can help with this task.

Some algorithms do not support numerical values then discretization must be performed. Or they do not support missing values than missing values must be filled. And so on.

2.5 Handling missing values

Missing values complicate both training and classification phase. But simple deleting instances with missing attributes may be a waste. One approach is to replace missing value by most frequent (for categorical) or average (for numerical) value of attribute (which may introduce some *noise* into data) [4]. Second approach is when calculating the splitting criteria for attribute X_i, than simply ignore all instances with missing value for attribute X_i. On the other hand, the splitting criteria should be reduced proportionally as nothing has been learned from these instances. There exists also another more sophisticated approaches to handling missing values.

2.6 Measuring the performance

The generalization error is probability to misclassify of unknown instance X and it is rarely known (because underlying distribution D of the labeled instance space is known only in synthetic cases). *Classification accuracy* is one minus generalization error. *The training error* is defined as percentage of correctly classified instances of *training set.* Training error is typically more optimistic than true generalization error.

Holdout and cross validation and it's variations are techniques used to empirically estimate generalization error. *Holdout method* randomly split given dataset into training and test sets (usually two-thirds of data is considered for training set) and error on test set is considered to be final estimation.

Another commonly used method is *k-fold cross-validation* which randomly divides dataset into k mutually exclusive subsets of approximately equal size. Then each subset is taken as test set and remaining subsets are taken as training data. For each that taken subset is computed training error and k training errors can be averaged (or otherwise combined) to produce final estimation. Special cases of this method are *2-fold cross-validation* (where $k = 2$) and *leave-one-out cross-validation* where k is equal to number of observations in the given dataset so it's very computational expensive (because training process is repeated many times).

Bootstrapping technique random samples full data *with replacement*. Instead of repeatedly analyzing subsets of data, it repeatedly analyze subsamples of data.

Not only predictive performance is important measure of classifier's quality, but also stability, discriminatory power, simplicity of model. [32]

2.7 Mining astrophysical data

Astrophysical classification and prediction tasks do not differ from general predictive problems much. Astrophysical experiments obviously consist of large collection of training data and great deal of unlabeled data. Numerical attributes are more common than categorical. Missing values are usually not present. The number of attributes are rarely very high.

We present experiments (6.1.1 and 6.1.2) which try to demonstrate common astrophysical data mining task. One for regression and one for classification problem.

We need to clarify that particular real-world experiment may differ greatly from the above characterization.

Chapter 3

Decision trees

Decision trees (DT) are class of predictive data mining (DM) methods which use decision tree as predictive model. They can be used either for classification or regression problems. Because *learning* such model is complex task many algorithms and frameworks were introduced. They differ in their approaches to solving particular problems, but underlying concept is most often the same – splitting the set of examples on *a split attribute(s)* in top-down manner – such algorithm is called Top-Down Induction of Decision Trees (TDIDT). The TDIDT has been known since the mid-1960s. [4]

The decision tree is a flowchart-like tree structure. Each *internal node* denotes a test on an attribute(s), each *branch* represents an outcome of the test, and each *leaf node* hold a class label (or target value). The topmost node is *root* node. Example of decision tree is shown in figure 3.1. [18]

Decision trees are very popular, because their representation in tree form is intuitive an easy interpretable by human. The learning and classification steps of decision trees are simple and fast. The decision trees can handle high dimensional data. In general, decision trees have good accuracy. [18]

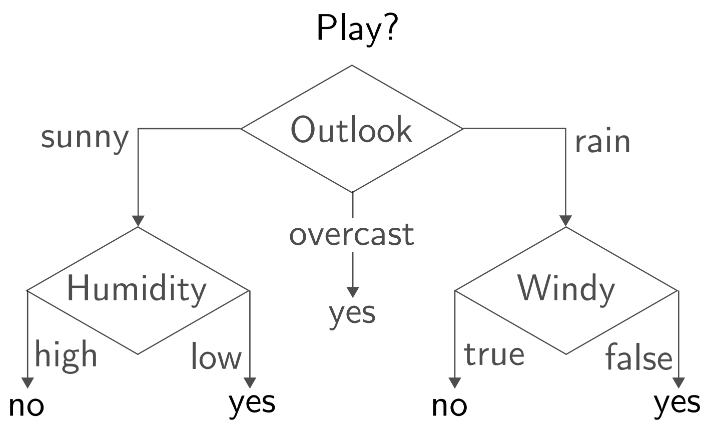


Figure 3.1: A simple decision tree

3.1 Decision tree learning

The task is build from given dataset predictive model which can predict class or numerical value. Such algorithm we call *tree inducer*. Because complexity of searching optimal tree

Algorithm 3.1 Growing decision tree

```
1. Split a given dataset on split condition into two or more subsets and create internal split node.
2. For each subset:
    (a) If stopping criterion triggered, then mark subset as leaf node and assign value to it
    (b) else apply point 1. on subset.
```

most algorithms search *reasonably good* decision tree. The result do not depend on particular order in which the examples are given [29].

3.2 Splitting criteria

Most of the algorithms works in greedy style trying maximize/minimize some *heuristic function* in each node.

With relation to size of training set decision tree with many nodes typically overfit data, which may lead to small training error. Overfitting mean that tree lack generalization – it describes training set but not underlying relationships. On other hand, decision tree with small capacity can underfit data, resulting to poor training error. [32]

3.2.1 Univariate splitting criteria

Univariate means that split is according to the value of a single attribute. The inducer searches for the best attribute upon which to split. There are several widely used functions for attribute selection. Impurity-based are Gini impurity, Information gain, Gain Ration, twoing criterion, etc. From statistical methods we mention just χ^2-test. Random selection of attribute is also possible. Common problem of selecting methods is *bias* towards attributes with larger domains. Some algorithms (Gain Ration, Distance Measure) perform *normalization* to deal with *bias*. [24]

Goodness of split criterion is not so important for predictive accuracy, but significantly influence the size of unpruned tree [26].

The entropy of the set D is

$$E(D) = -\sum_{i=1}^{K} p_i \log_b(p_i)$$

where p_i is the probability that an instance X_i belongs to class Y_i and it is estimated by $|Y_{i,D}|/|D|$. Unit of entropy is *bit* for $b = 2$.

The Gini impurity is $Gini(D) = 1 - \sum_{i=1}^{K} p_i^2$.

For the numerical target the sum of squares is used as measure of entropy: $S(D) = \sum_{i=1}^{K}(Y_i - \bar{y})^2$, where $\bar{y}$ is the predicted value (most commonly the average of the target values) and Y_i is the ith target value.

3.2.2 Multivariate splitting criteria

In *multivariate splitting criteria* more attributes may participate in a single node split. Multivariate splitting criteria are mostly based on linear combination of the attributes. Methods used for finding best *linear combination* can be performed using a greedy search, linear programming, linear discriminant analysis and others. This type of criteria may dramatically improve tree's performance, but is more complicated to find best multivariate split, hence this criteria are much less popular. [24]

3.3 Surrogate splits

For each split node is given an ordered set of surrogate splits, consisting of an attribute label and a rule. Surrogate splits maximize the "predictive association" with the primary split. The surrogate split is used when classify a new example with missing attribute. [11]

3.4 Stopping criterion

Using the stopping criterion is sometimes called pre-pruning as opposite to post-pruning. Stopping criterion determine when to *stop* growing decision tree. As stopping criterion we may use (and possibly combine):

- all instances in subset belong to single class Y,
- splitting criteria is not greater than a threshold,
- maximum tree depth was reached,
- size of subset is lower than threshold,
- ...

3.5 Discretization

Many tree inducers require all attributes to take categorical values. Thus continuous attributes must be *discretized.* The simplest approach is to just treat continuous values as categorical ones, but this is unlikely to be efficient.

The common method is to divide a continuous attribute into *intervals.* Ranges of each interval may be equal size, this we call *equal width intervals.* Another method, where the number of instances in each interval are equal, is known as *equal frequency intervals method.* Both methods share same problem – how many intervals to choose?

ChiMerge algorithm is statistical approach to data discretization. It discretize each attribute separately with use of χ^2 test to determine similarity of two intervals. If intervals are very similar, then they can be merged. ChiMerge works recursively in bottom-up manner, and uses class information – in that it is *supervised.*

The discretization may be global or local. Local discretization is performed at each node of the decision tree. Global discretization (e.g. ChiMerge) converts each continuous attribute to categorical one once. Global discretization may be applied on dataset only once and independently on data mining algorithm. [4]

3.6 Post-pruning

Because employing good stopping criterion is complicated, post-pruning method was suggested. Also it can compensate, to some extent, for the sub-optimality of greedy tree induction [28]. Loosely stopping criteria is used which let the tree to overfit the training set. Than over-fitted tree is pruned back into smaller tree by cutting branches that are not contributing to the generalization accuracy.

There are several pruning algorithms. Most of them traverse tree bottom-up or to-down and prune nodes if it improves a certain criteria.

Cost-complexity pruning In the first stage, a sequence of increasingly smaller trees are build on the training data (from the original tree before pruning to the root tree by replacing one or more of the sub-trees in the predecessor tree with suitable leaves). In the second stage one of this trees is chosen as the pruned tree, based on its accuracy on a *pruning set.* [28, 24]

Reduced error pruning This method does not build sequence of trees. Simple traverse over the internal nodes bottom-up and prune node if pruning node does not reduce accuracy. Pruning set is used to estimate accuracy.

Pessimistic pruning Pessimistic pruning avoids the need for a separate pruning set by using a statistical correlation test. Procedure traverse tree in top-down direction and because descendants of pruned nodes are removed from the pruning process, procedure is relatively fast.

Error-based pruning Error-based pruning is an evolution of pessimistic pruning and it's implemented in C4.5 algorithm.

Minimum description length (MDL) pruning The minimum description length can be used for evaluating the generalized accuracy of node. MDL-based pruning methods are more popular for large dataset because they scale well [25, 33].

3.7 Datasets

Two basics dataset are needed for DT learning and evaluating – *training* and *testing* dataset. Some post-pruning algorithms may require special *pruning* dataset.

3.8 Algorithms

In this section we briefly introduce basic algorithms ID3, CART and two newer and interesting scalable algorithms.

3.8.1 ID3

ID3 (Iterative Dichotomiser 3) algorithm is one of the best known examples of tree inducer. It employs top-down, greedy search through space of possible decision trees in divide-and-conquer manner. Simplified algorithm is described in algorithm 3.2.

ID3 starts with a training set of instances and their class labels. The training set is recursively partitioned into smaller subsets on split attribute. Each attribute is evaluated

Algorithm 3.2 Summary of the simplified ID3 algorithm for boolean-valued functions [27]

```
ID3(Instances, TargetAttribute, Attributes)
  • Create a Root node for the tree.
  • If all Instances are positive (or negative), return the single-node tree Root, with label + (or −)
  • If Attributes is empty, return the single-node tree Root, with label = most common value of TargetAttribute in Instances.
  • Otherwise
      ◦ A ←the attribute from Attributes with highest information gain
      ◦ Let $Instances_{vi}$ be the subset of examples that have values $v_i$ for A
      ◦ If $Instancess_{vi}$ is empty
          · Than below this new branch add a leaf node with label = most common value of TargetAttribute in Instances
          · Else below this new branch add the subtree
            ID3($Instances_{vi}$, $TargetAttribute$, Attributes − {A})
```

using a statistical test to determine which one should be tested in split condition. The information gain is used as statistical test.

C4.5, the successor of ID3, uses gain ratio as splitting criteria. C4.5 can handle missing values and continuous attributes. It performs error-based pruning after the growing phase.

3.8.2 CART

CART (Classification and Regression Trees) construct binary trees. The CART is able to solve regression tasks (in this case it looks for splits that minimize the prediction squared error). It uses surrogates for overcoming the problems caused by missing values and uses pruning. It uses Gini impurity as splitting criteria.

3.8.3 RainForest

RainForest is unifying framework for decision trees construction that separates the scalability aspects of algorithm from the central features that determine the *quality* of the tree [17]. Framework applied to split selection method results in the scalable version of the origin method without modifying the result of the method. Only univariate splits are supported.

Framework concentrate on the tree growing phase, since it is a very time consuming due to its data-intensive nature.

Method at each node of the tree maintains for each attribute *AVC-set* („Attribute-Value, Classlabel”), which describe the training tuples at the node – it holds aggregate information. The size of AVC-set at node N depends only on the number of distinct values of attribute and the number of classes in the subset at N. The set of all AVC-sets at some node is *AVC-group*. This sets typically should fit in memory (even if original training set can't). RainForest provides also techniques for handling the case when even AVC-group does not

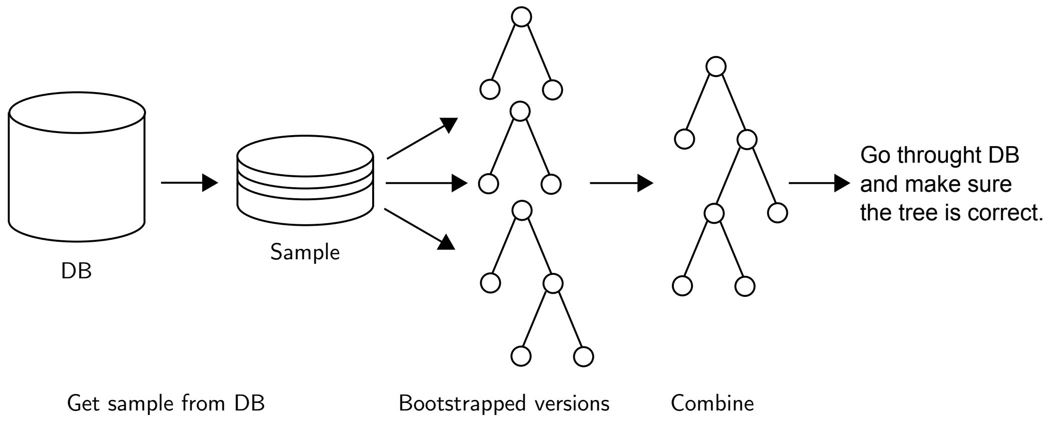


Figure 3.2: BOAT algorithm

fit in memory. [18]

3.8.4 BOAT

BOAT stands for *B*ootstrapped *O*ptimistic *A*lgorithm for *T*ree Construction. BOAT constructs several levels of the tree just with two pass over training dataset. BOAT was found to be two or three times faster than RainForest, while constructing exactly the same tree. An additional advantage of BOAT is that it can be used for incremental updates. The algorithm is not based on the use of any special data structures. The key idea is *optimistic* approach to tree construction in which we construct an initial tree using a small subset of the training data and refine it to arrive at the final tree. [16, 18]

Since training data D can not fit into memory, algorithm use subset $D^{'} \subset D$, which can fits into memory. Then algorithm use bootstrapping and compute trees. From generated trees are obtained *coarse splitting criteria.* The coarse splitting criteria reduces set of possible splitting criteria at every node. They are created by traversing trees in top-down manner and comparing splitting attribute X. If each X is not identical, node and subtree are removed. If X numerical, from splits points we can obtain a confidence interval. If X is categorical, than subsets induced by the splits must be identical in all subtrees. Otherwise, node and subtree are removed. Result of this procedure is tree T' made from „overlapping parts“ of bootstrapped trees. Resulting tree is evaluated on all data and if tree is not correct, the learning process is repeated.

Algorithm is illustrated on figure 3.2.

Chapter 4

Ensemble methods

Methods to make trees more stable and provide accurate predictions are ensemble methods such as bagging, arcing and boosting. First we introduce two components of test set error, *the bias* and *the variance.*

4.1 Bias and variance

The bias measures the accuracy or quality of the match, since the variance measures the precision or specificity of the match. Low bias means on average we accurately estimate underlying distribution F from training dataset D. A low variance means that the estimate of F does not change much as the training set varies. We can adjust the bias and variance of classifiers, but they are not independent [11]. Unstable classifiers can have low bias on a large range of datasets, but their problem is high variance [6].

4.2 Bagging predictors

Bagging predictors [6] (or *bootstrap aggregation*) is a method for generating multiple versions of a predictor and using these to get a aggregate predictor. This method can be used either for classification (majority voting) or regression (average) problems. Tests show that bagging can give substantial gains in accuracy, with critical factor in improvement of stability. On other hand method can degrade the performance of stable procedures and we lose tree's interpretable structure.

Parameters for method are 1) number of bootstrap replicates (Breiman suggest from 20 to 50), 2) size of the bootstrap learning set (Breiman suggest same size as size of training data set).

We can also get test set (or pruning set) by sampling *with replacement* from the training set.

4.3 Arcing classifiers

Arcing is acronym for *a*daptively *r*esample and *c*ombine. Basis of method is that the weights in the resampling are increased for those cases most often misclassified and the combining is done by weighted voting. Main effect of bagging and arcing is to reduce variance. Arcing seems to usually do better at this.[6]

Algorithm 4.1 Random decision forest for regression or classification [19]

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $D'_b$ of size $N$ from training data $D$.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating
        the following steps for each terminal node of the tree, until the stopping criterion
        is reached.
         i. Select $m$ variables at random from the $p$ variables.
        ii. Select the split condition among the $m$.
       iii. Split the binary node.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new instance X :

Regression $\hat{f}^B_{rf}(X) = \frac{1}{B}\sum_{b=1}^B T_b(X)$.

Classification $\hat{C}^B_{rf}(X) = majority\,vote\,\{\hat{C}_b(X)\}_1^B$, where $\hat{C}_b(\cdot)$ is the class prediction of bth tree.

4.4 Algorithms

4.4.1 Random decision forests

The term "random decision forests" (RDF) was introduced by Ho (1995) [21]. The essence of the method is to build multiple trees in randomly selected subspaces of the feature space.

Breiman (1996) [5] introduced bagging (see 4.2), a precursor to his version of random forests.

The Random Forests$^{\text{TM}}$[1] (RF$^{\text{TM}}$) as described by Breiman (2001) [7] is a combination of tree predictors. Each tree casts a unit vote for the most popular class at input X. The Random Forests do not overfit as more trees are added, but produce a limiting value of the generalization error. It is as good as Adaboost, it's relatively robust to outliners and noise, it's faster than boosting, it gives useful internal estimates of error, strength, correlation and variable importance and it's simple and easily parallized. Random Forests where implemented also on GPU [34, 35]. The method is most effective for problems involving high dimensional data because of the existence of more subspaces [21].

Breiman's RF uses bagging in tandem with random feature selection. Each tree is grown on the new training set using random feature selection while the new training set is drawn from the dataset with replacement. No pruning is done. This builds a large collection of *de-correlated* trees [19]. Breiman's RF have many another features (variable importance, proximities, interactions, . . .), but for our purposes the only core idea is important.

[1]Random Forests$^{\text{TM}}$ is a trademark of Leo Breiman and Adele Cutler and is licensed exclusively to Salford Systems for the commercial release of the software.

Random input selection

The simplest case is formed by selecting at random, at each node, a small group of m input variables to split on.

Using linear combinations of inputs

If there are only a few inputs, we may define more features by taking random linear combination of L inputs. At a given node, L variable are randomly selected and added together with multiplication coefficients uniformly generated from range $[-1, 1]$. F linear combinations are generated and then search for the best split is made over these.

For incommensurable input variables normalization must be performed. Normalization consist of subtracting means and dividing by standard deviations, each determined from training set.

For use categorical inputs in linear combination this inputs must be coded into dummy 0–1 variables. Variable with I values can be coded into $I - 1$ dummy variables. This make categorical variable $I-1$ times as probable as numeric variable to be selected in node splitting. When many of the variables are categorical, F must be increased.

Out-of-bag error estimate

Out-of-bag estimates of the generalization error, the strength and correlation are computed as follows. From training set D form bootstrap training sets D_b and train classifiers T_b. This classifiers forms *bagged predictor.* Then for each X, Y from training set, aggregate votes from classifiers for which D_k does *not* contain X, Y. This is called *out-of-bag classifier.* And from the error rate of the out-of-bag classifier we get the out-of-bag estimate for the generalization error.

This estimate is as accurate as using a test set of the same size as training sets and tend to overestimate the current error rate. This property removes the need for stand alone test set. [7]

Chapter 5

Design and implementation

5.1 Requirements

We introduce requirements developed with consultant Petr Škoda.

The implemented algorithm must use tree-like structure as a model and must solve classification and also regression problems. It should be time and memory efficient; the prediction accuracy is also very important. The comparison of different algorithms and the rationale for selection of particular algorithm must be provided.

The implementation should run on Unix-like systems. Rich settings configuration should be possible via command line parameters and/or configuration files. The program should have object oriented design, easily extendable and modifiable. It may provide graphical representation of trained model.

The important task is also the validation and test of the model on real astrophysical use cases.

5.2 Comparison of algorithms

Ensemble methods have theoretical advantages over single decision tree models [7, 19] confirmed by empirical experiments [5, 10, 9]. Empirical experiments show that the most interesting are boosted decision trees and random decision forests. Boosted decision trees perform better than random decision forests mainly on low dimensional data, but not significantly. On high dimensional data random decision forests outperform boosted decision trees [9] and are robust with respect to input parameters, even if strongly correlated [1]. Random decision forests are faster than boosting [7]. The method was successfully used in astronomical challenges [31, 1, 8]. And the method needs a minimum of human intervention – there are only few tunable parameters.

5.3 Algorithm description

Our random decision forest algorithm uses bagging for combination of binary decision trees.

Binary decision trees use n randomly selected attributes at each node for learning. As split criteria they use Gini impurity (for categorical target) and squared error (for numerical target).

For splitting on numerical attribute, data are sorted according this attribute and than for each possible split threshold splitting criteria is evaluated. For N distinct values there are $N-1$ possible splitting thresholds. For splitting on categorical attribute, each binary partition is evaluated – for N categories there are 2^N-2 possible partitions.

As stopping criterion are used minimum instances in node, maximal depth and minimal squared error (for a numerical target only).

Instances with unknown target value are ignored. When splitting in a node on an attribute, instances with the unknown value for this attribute are ignored.

For classification/prediction on the new data random decision forests use majority vote / average value. Particular decision tree may return *unknown value* – such value is ignored by RDF. The unknown value may be returned because of a test on a categorical attribute.

5.4 Data format

The ARFF[1] data format is well-known, text-based format. My implementation supports subset of the ARFF format. It supports *numeric* and *class* attributes, comments, relation name and missing values denoted by "?" character. It does not support the Sparse ARFF format, string and date attributes, and quotation marks and escape characters, because they are not widely supported by data mining programs.

Advantages of the ARFF format are portability, simplicity and header section with attributes declaration. Attributes declaration simplify parsing of data and allows better error checking. Disadvantages are size of the data file and missing information about the number of instances (this would be useful for memory allocation).

The example of the ARFF format is on the listing 5.1. It contains comments (start with "%" character), dataset's name "iris", four numerical attributes, one categorical attribute with three possible values and data section.

The output file is the same as input file but with appended a new attributes: with prediction and numerical attribute with a "certainty" value. The certainty (attribute "bcrdf_prob") is based on the ratio of trees which vote for the target class to the total number of trees or on the squared deviation of predictions of trees. The value 1 (for classification) means that every tree vote for the predicted class. The value 0 (for prediction) means that every tree predicted the same value.

5.5 Architecture

Program consist of two main logical parts – data-related (5.1) and data mining related (5.2). Data-related part consist of data and meta data representation and input/output handlers. Data mining-related part consists of particular algorithms and data structures for model representations.

MetaData consists of *Attributes*. *Attribute* may be *NumericalAttribute* or *CategoricalAttribute*. *CategoricalAttribute* ensures mapping of string representation of categories to internal numerical representation and back by *boost::bimap*.

Data represents dataset. It is implemented by *ExtendableData* for storing unknown number of instances, by *OneRowData* for storing data when classifying/predicting and *BaggedData*. *BaggedData* acts as proxy to another *Data* and contains indexes to sampled instances. Static method *getData* should return a best implementation of *Data* for the given *MetaData* and possible count of instances.

[1]The ARFF format description: http://weka.wikispaces.com/ARFF

Listing 5.1: Part of the Iris data in the ARFF format

```
% 1. Title: Iris Plants Database
%
% 2. Sources:
%      (a) Creator: R.A. Fisher
%      (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
%      (c) Date: July, 1988
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
 5.1,3.5,1.4,0.2,Iris-setosa
 4.9,3.0,1.4,0.2,Iris-setosa
 4.7,3.2,1.3,0.2,Iris-setosa
```

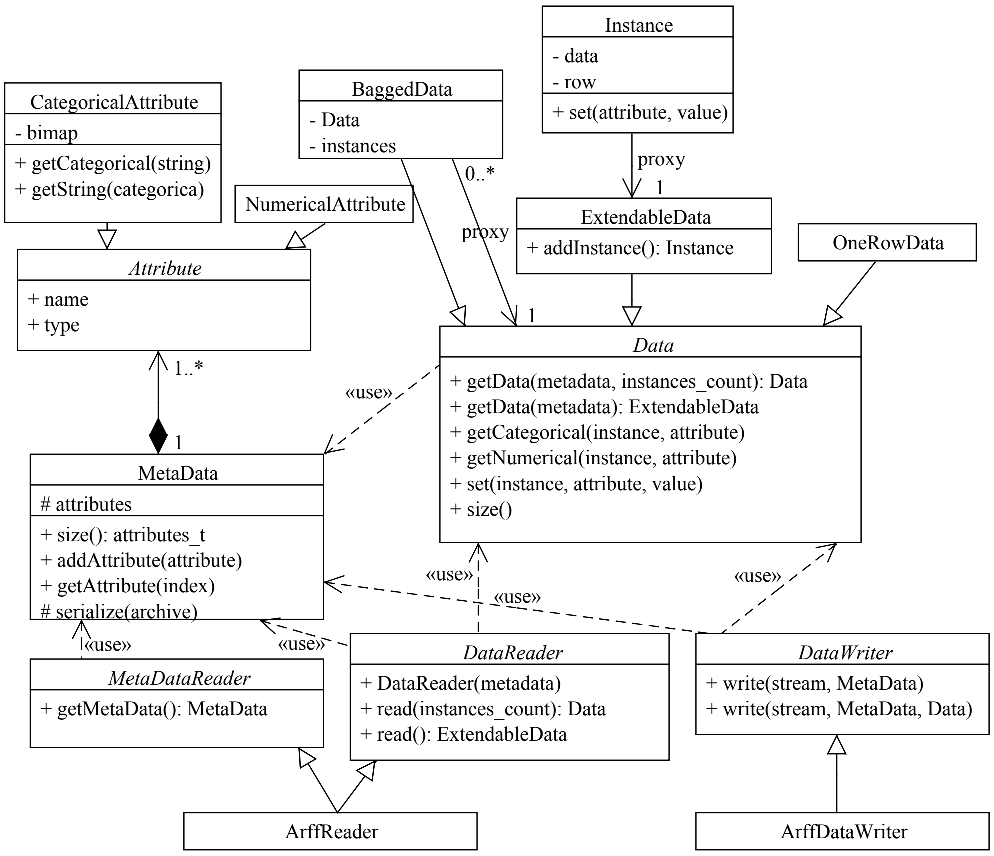


Figure 5.1: Class diagram of data-related classes

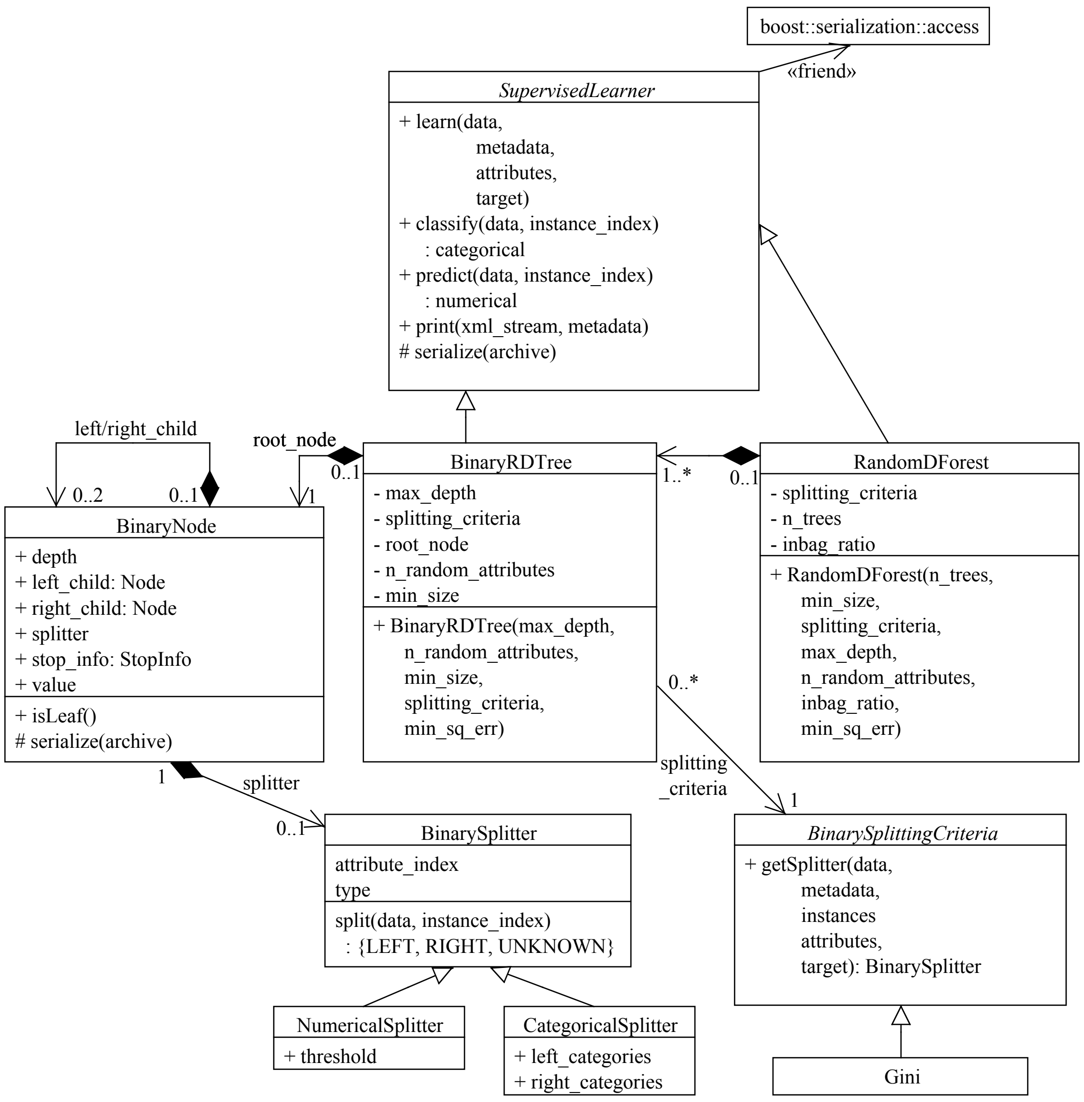


Figure 5.2: Class diagram of model-related classes

MetaDataReader and *DataReader* read input stream. They are implemented by *ArffReader*.

DataWriter write data and meta data to an output stream and is implemented by *ArffDataWriter*.

SupervisedLearner represents supervised data mining algorithms and models. It supports learning on given data and classifying/predicting on unseen data. It also supports serialization and printing of model in the XML format.

SupervisedLearner is implemented by *RandomDForest* and *BinaryRDTree*. *RandomDForest* implements random decision forests algorithm. It consists of *BinaryRDTrees*. It uses bagging (with *BaggedData*) for training particular trees. *BinaryRDTree* consists of binary nodes *(BinaryNode)*. It uses *BinarySplittingCriteria* for searching best splits. Splits are represented by *BinarySplitter*. Each internal *BinaryNode* contains one instance of *BinarySplitter*. Each leaf node contains target value and information about stopping criteria – *StopInfo* (for example "maximum depth").

5.6 Implementation

The program is implemented in *C++11* language. The object oriented paradigm allows easy functionality extending. It uses *Boost C++ Libraries*[2] (mainly program options, serialization, smart pointers and accumulators). For fast combinations computing for Gini impurity it uses *combinations.h [20]* by Howard Hinnant. *Glog* library handles logging (5.9) and for XML creation is also used external code (5.11).

Information about dependencies and compilation are in the *README* file.

5.7 Program flow

There are two main use cases of an experiment – *training* and *running*. The training phase include learning model on given data. In the running phase is such model used for predicting on new data. Running on the new data *without* target attribute we call *predicting* and running on the data *with* target attribute we call *testing*. When testing the accuracy can be evaluated.

Train Program reads meta data and data. Than it trains model specified by user. Resulting model and meta data are serialized into file. Model may be printed in XML into file.

Run Program deserialize model and meta data. Than it loads new meta data. New and old meta data are checked. Than it reads, classify and writes to output one instance at time. If target value is provided in new data than number of correctly classified instances or root mean squared error is reported.

5.8 Program options

For parsing command line parameters and configuration files is used *boost::program_ options*. Details about options and configuration file format are in appendix A.

[2] *Boost provides free peer-reviewed portable C++ source libraries.* http://www.boost.org/

5.9 Logging

For logging of various messages about progress, warnings, debug messages, etc is used *glog (google-glog)* [3]. The verbosity of the program is controlled by the environment variable *GLOG_v*; for the recommended value 2 the program parameters and learning progress is printed. Higher values are more verbose. Verbosity may be controlled also for each module separated – for example *"GLOG_vmodule=main=0,RandomDForest=5"* for very verbose messages about learning random decision forest *(RandomDForest=5)* and no messages about program parameters and program flow *(main=0).* For logging messages to standard error output *GLOG_logtostderr=1* must be set.

5.10 Serialization

Trained model *(SupervisedLearner)* and meta data *(MetaData)* are (de)serialized with *boost-::serialization.* Deserialization of meta data allows check of the new data for *compatibility* – data for classification/prediction must contain same attributes with same types, categorical attributes must have identical categories. Model and meta data are serialized into XML file which ensures portability.

5.11 Graphical output

Graphical representation of the model serves user's better understanding of the experiment's output. The RDF graphical output is used mainly for illustrative and educational purposes, because ensemble models are not easily human interpretable.

Model can be printed to file in XML format. It uses *Simple C++ class for XML writing*[4]. The class is slightly modified for providing indented output.

The *RandomDForest* XML output may be converted to DOT[5] language using script *./utils/rdf2dot.py.* The file in DOT format may be converted to various graphical outputs with for example Graphviz software[6].

Example of the graphical output is on figure 5.3. It is random decision forest trained on Iris data. The DOT file was converted to image with *dot* utility (part of the Graphviz). Another example is in the appendix B.

5.12 Testing

For unit testing is used *googletest – Google C++ Testing Framework*[7]. There are written various tests mainly for data reading. Script *./utils/run_tests.sh* provides convenient way to run all tests with error reporting.

[3]Logging library for C++. See http://code.google.com/p/google-glog/.

[4]Simple C++ class for XML writing. See http://www.codeproject.com/Articles/5588/Simple-C-class-for-XML-writing.

[5]DOT is a plain text graph description language. http://www.graphviz.org/content/dot-language

[6]*Graphviz is open source graph visualization software.* See http://www.graphviz.org/.

[7]Google C++ Testing Framework. See http://code.google.com/p/googletest/.

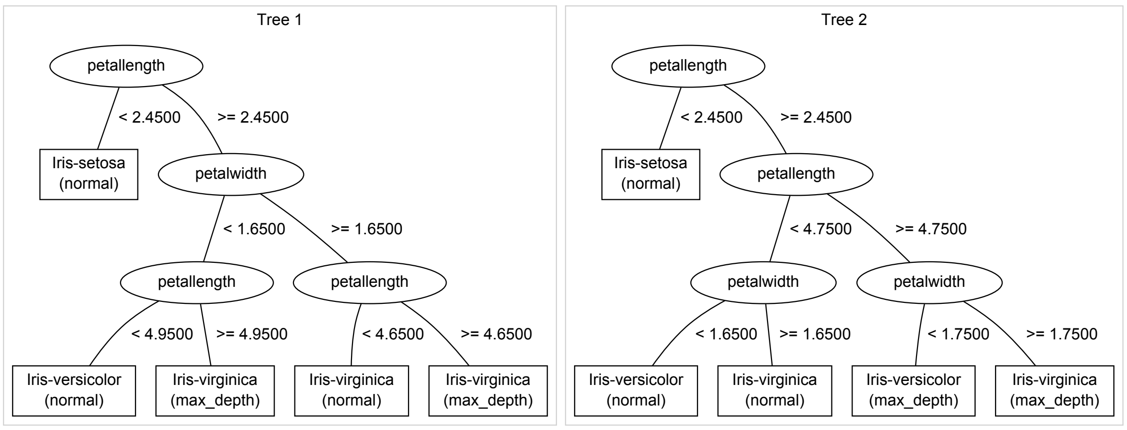


Figure 5.3: Graphical output for random decision trees trained on Iris data (configuration: two trees, maximum depth is two, two random attributes)

Chapter 6

Experiments

We describe and execute one classification and one regression astrophysical experiments. For validation of my implementation we run experiments on well-known datasets from UCI repository [3].

6.1 Data

We introduce two astrophysical datasets in detail and brief summary of validation datasets.

6.1.1 Astrophysical classification – Stars data

The Stars data consists of three continuous attributes – the color indexes[1] u-g, g-r, r-i and one target attribute. The target class attribute takes three values – STAR, GALAXY and QSO which stands for stars, galaxies and quasi-stellar objects.

Data was retrieved from SDSS (Sloan Digital Sky Survey) Data Release 7 (DR7) via CasJobs[2] using SQL query (listing 6.1). Sample of the output file from CasJobs in CSV format with headers is in listing 6.2. Data consists of 3 000 labeled instances. There are no missing values. Data are illustrated on figure 6.1 where you can see density estimates for each continuous attribute dependent on target class.

Graphical example of the random decision forest trained on the Stars data is in the appendix B.

6.1.2 Astrophysical regression – Redshift data

Redshift data was obtained from DAME Photometric redshift Estimation tutorial[3] (five thousands instances was randomly sampled from tutorial's dataset which contains 30 000 instances). Data contains four color indexes and target continuous attribute. Target attribute *zspec* stands for photometric redshift. Data consists of 5 000 labeled instances; there are no missing values. On figure 6.2a you can see many outliers and on figure 6.2b are histograms (for clarity without outliers).

[1]A color index is the difference between two magnitudes of the same star obtained with two different photometric filters. Magnitude is measure of the brightness of an object, measured in specific wavelength or wavelengths range.

[2]http://casjobs.sdss.org

[3]http://dame.dsf.unina.it/dame_photoz.html

Listing 6.1: Query to retrieve classification data from CAS

```
select top 1000
    u-g as u_g,
    g-r as g_r,
    r-i as r_i,
    dbo.fSpecClassN(s.specclass) as class
  from photoprimary p join specphotoall s on p.objid=s.objid
  where s.specclass = 1
  and u between 18 and 19
union all
select top 1000 u-g, g-r, r-i, dbo.fSpecClassN(s.specclass)
  from photoprimary p join specphotoall s on p.objid=s.objid
  where s.specclass = 2
  and u between 18 and 19
union all
select top 1000 u-g, g-r, r-i, dbo.fSpecClassN(s.specclass)
  from photoprimary p join specphotoall s on p.objid=s.objid
  where s.specclass = 3
  and u between 18 and 19
```

Listing 6.2: Sample of the output file

```
u_g,g_r,r_i,class
1.12368965148926,0.194650650024414,0.107603073120117,STAR
1.2844352722168,0.510932922363281,0.176275253295898,STAR
1.94010353088379,0.910341262817383,0.404125213623047,GALAXY
1.1341438293457,0.431035995483398,0.279729843139648,GALAXY
0.123207092285156,0.212028503417969,-0.0941390991210938,QSO
0.361385345458984,0.199728012084961,0.0848274230957031,QSO
```

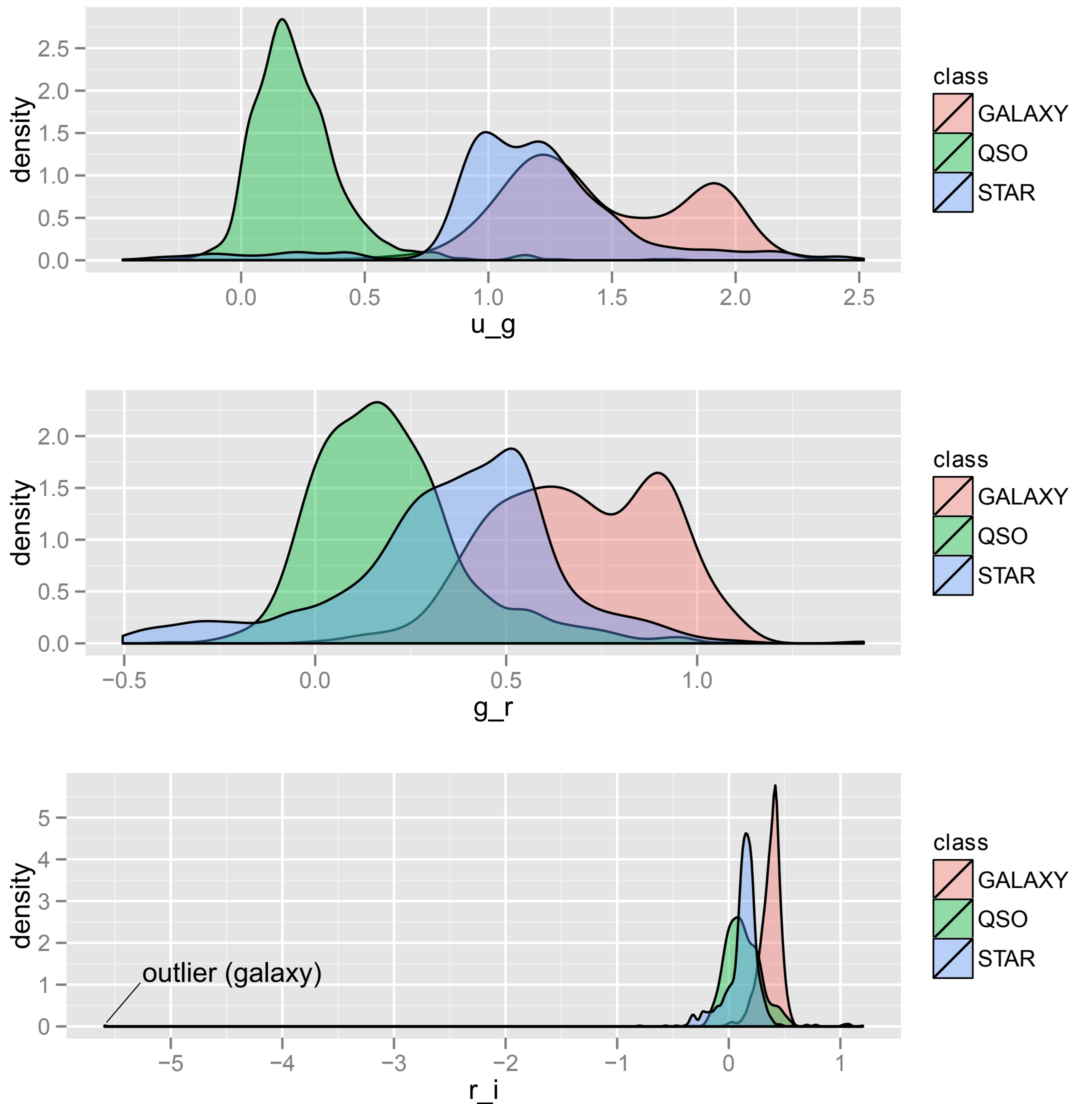
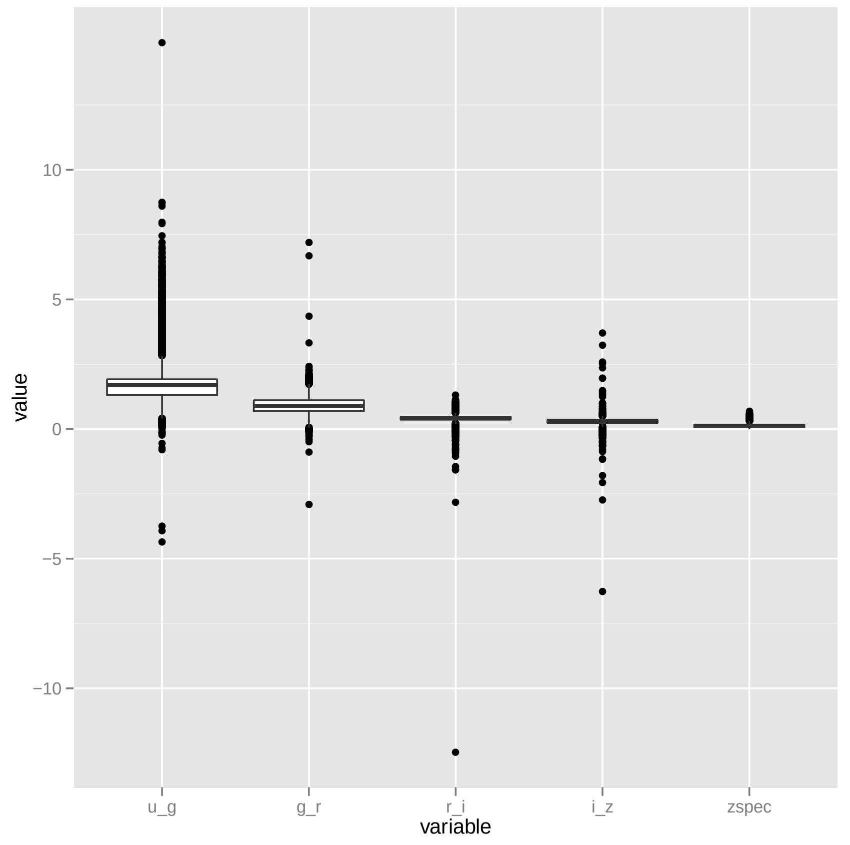
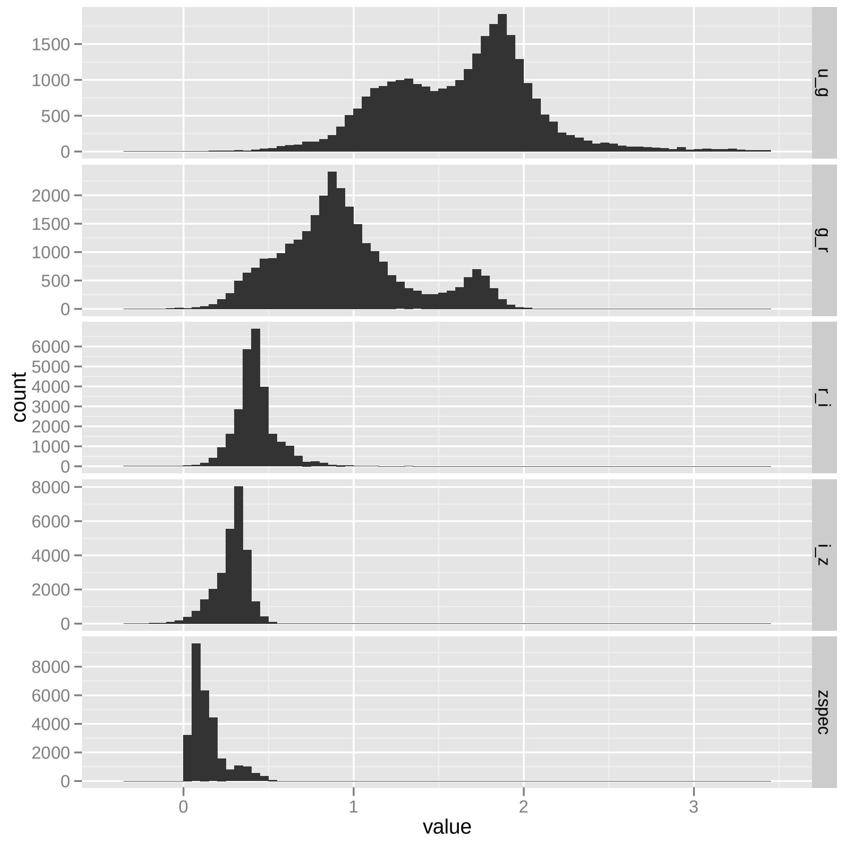


Figure 6.1: Stars data density estimates



(a) Boxplots identify outliers



(b) Histograms (without outliers)

Figure 6.2: Graphs for the Redshift data

dataset	instances	attributes	unknown values (%)	target
stars	3000	4	0.00	categorical
diabetes	768	8	0.00	categorical
glass	214	10	0.00	categorical
hepatitis	155	20	5.39	categorical
iris	150	5	0.00	categorical
letter	20000	17	0.00	categorical
vote	435	17	5.30	categorical
vowel	990	12	0.00	categorical
zoo	101	17	0.00	categorical
redshift	5000	5	0.00	numerical
auto93	93	22	0.68	numerical
autoMpg	398	8	0.19	numerical
bodyfat	252	15	0.00	numerical
fishcatch	158	8	6.88	numerical
housing	506	14	0.00	numerical

Table 6.1: Summary of datasets used in experiments

6.1.3 Validation datasets

Validation datasets from UCI Repository[3] were obtained from Weka Collections of datasets[4] (datasets from UCI Repository in ARFF format). The validation datasets consist of 8 datasets with categorical target and 5 datasets with numerical target. The summary of the validation datasets, the stars data and the redshift data is in table 6.1.

6.2 Experiments

For experiments holdout validation (2.6) was chosen (because it is easy to implement). The flow of the experiment is:

- Twenty times:
  - Shuffle labeled data and split them into training dataset (70%) and testing dataset (30%).
  - Train models on training dataset.
  - Run models on testing datasets.
  - Compute error.
- Average errors.

For splitting the data the *waffles_ transform* is used (the part of the Waffles [15]). For error computing and result creation various scripts in R [30] are used.

[4]http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html

dataset	bcrdf	R	rf-ace	waffles
	Classification error (%)			
stars	4.00	3.50	5.00	**3.40**
diabetes	25.20	**23.80**	24.70	24.50
glass	27.50	**26.60**	34.60	27.20
hepatitis	**9.30**	16.00	18.50	18.10
iris	5.00	**4.10**	6.90	**4.10**
letter	**3.80**	4.40	*92.00*	4.00
vote	**3.20**	4.00	4.30	4.30
vowel	4.40	4.90	6.90	**3.60**
zoo	**8.70**	18.20	34.20	14.20

Table 6.2: Classification experiments

6.2.1 Implementations

bcrdf The solution presented in this thesis.

R Package *randomForest* [22] (version 4.6-6) in R. The randomForest package provides an convenient R interface to the original Fortran programs by Breiman and Cutler (available at `http://www.stat.berkeley.edu/users/breiman/`).

rf-ace The RF-ACE [12] (version v1.0.4) is implemented in C++ and targets to fast implementation of random decision forests and gradient boosting trees. It is able to also perform feature selection. It is intensively developed since February 2011.

waffles The Waffles [15] (version waffles-2011-12-6) is implemented in C++ and targets to be "the world's most comprehensive collection of command-line tools for machine learning and data mining".

6.3 Results

Classification errors are in table 6.2. My implementation is comparably successful as R (package randomForest) and Waffles. RF-ACE is doing well except *letter* data where it fails.

Prediction errors are in table 6.3. R and Waffles have the best results. My implementation and RF-ACE is doing slightly worse.

Average running times are in table 6.4. All implementations are comparably fast on the small data. My implementation is very slow on the data with many instances.

dataset	bcrdf	R	rf-ace	waffles
	Root-mean-square deviation			
redshift	0.0011	**0.0009**	0.0026	0.0010
auto93	39.46	**35.43**	39.37	37.65
autoMpg	8.72	**8.56**	15.81	9.791
bodyfat	10.8	9.0	11.4	**6.7**
fishcatch	15639.3	8000.0	21437.3	**5878.8**
housing	33.2	**12.6**	17.7	13.8

Table 6.3: Regression experiments

	dataset	bcrdf	R	rf-ace	waffles
Classification	**stars**	1.34	0.89	0.37	0.52
	diabetes	0.36	0.71	0.18	0.23
	glass	0.13	0.70	0.06	0.07
	hepatitis	0.09	0.69	0.05	0.06
	iris	0.07	0.79	0.03	0.04
	letter	**52.84**	5.50	4.11	5.44
	vote	0.12	0.71	0.09	0.06
	vowel	0.94	0.78	0.46	0.38
	zoo	0.07	0.71	0.04	0.03
Regression	**redshift**	**162.04**	3.43	0.38	3.18
	auto93	0.12	0.69	0.04	0.05
	autoMpg	**3.56**	0.70	0.10	0.17
	bodyfat	0.59	0.70	0.09	0.13
	fishcatch	0.17	0.68	0.04	0.09
	housing	1.59	0.77	0.22	0.29

Table 6.4: Average running time of experiments

Chapter 7

Conclusion

This project aims to implement data mining algorithm for astrophysical usage.

In this thesis I described theoretical background and briefly introduced interesting algorithms. I choose the random decision forests and provided rationale for this selection. I do not known an implementation of RDF which declared efficient implementation, is well documented, easily extendable and provide graphical representation of trained model. So I designed and implemented the algorithm in C++ with usage of the Boost and other third party libraries. I described my implementation in detail and compared it with various related implementations of the random decision forests. For the comparison and for a validation of implementations I performed two astrophysical experiments and 13 experiments on datasets from UCI repository [3]. In experiments I measured accuracy and running time of the implementations.

The developed implementation is documented by this thesis, comparable accurate and slower on bigger datasets than related implementations. Because the implementation is not good scalable in depending on the number of instances a memory efficiency was not measured. As the only implementation enables a graphical representation of the trained model (I also committed extension for a textual only representation of the model into Waffles [15]). The implementation has the rich configuration possibilities and is easily extendable by inheritance.

For the real world usage I would recommend Waffles or R (package *randomForest*) because of longer development time, environment providing many other algorithms and much more experienced authors. Astrophysicists may be interested also in some data mining programs with graphical user interface (for example Weka[1]). For mining of really massive data sets distributed computing is the only possible way (for example see Apache Mahout[2]).

In the future work I would like to reveal bottlenecks in time efficiency (by profiling), measure and optimize the memory usage and extend the program for more file formats, splitting criteria and RDF functionalities (like out-of-bag estimates). The addition of parallelism may be also possible and interesting way to go.

[1]Weka is collection of machine learning algorithms with GUI. See `http://www.cs.waikato.ac.nz/ml/weka/`.

[2]Apache Mahout is collection of scalable machine learning libraries. See `http://mahout.apache.org/`.

Bibliography

[1] J. Albert. Implementation of the random forest method for the imaging atmospheric cherenkov telescope magic. Technical report, Sep 2007.

[2] E. Andrássyová, J. Paralic, E. A. Msc, and J. P. Phd. Knowledge discovery in databases: A comparison of different views. Technical report, 10th International Conference on Information and Intelligent Systems, Varazdin, Croatia, 1999.

[3] A. Asuncion and D. Newman. Uci machine learning repository. 15 Mar. 2012 http://www.ics.uci.edu/~mlearn/MLRepository.html.

[4] M. Bramer. *Principles of Data Mining (Undergraduate Topics in Computer Science)*. Springer-Verlag New York, Inc., 1st edition edition, 2007.

[5] L. Breiman. Bagging predictors. *Machine Learning*, pages 123–140, 1996.

[6] L. Breiman. Arcing classifiers. *Annals of Statistics*, 26(3):801–824, 1998.

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

[8] L. Breiman, M. Last, and J. Rice. Random forests: finding quasars. *Statistical Challenges in Astronomy*, pages 243–254, 2003.

[9] R. Caruana, N. Karampatziakis, and A. Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *In International Conference on Machine Learning (ICML*, pages 96–103, 2008.

[10] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pages 161–168, New York, NY, USA, 2006. ACM.

[11] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[12] T. Erkkilä. Rf-ace – multivariate machine learning with heterogeneous data. 15 Mar. 2012 http://code.google.com/p/rf-ace/.

[13] U. Fayyad, G. Piatetsky-shapiro, P. Smyth, and T. Widener. The kdd process for extracting useful knowledge from volumes of data. *Communications of the ACM*, 39: 27–34, 1996.

[14] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. *In G. Piatetsky-Shapiro and W. Frawley, editors, Knowledge Discovery in Databases. AAAI Press/The MIT Press*, 13:1–30, 1991.

[15] M. S. Gashler. Waffles: A machine learning toolkit. *Journal of Machine Learning Research*, 12:2383–2387, July 2011.

[16] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. Boat-optimistic decision tree construction. In *SIGMOD Conference*, pages 169–180, 1999.

[17] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest - a framework for fast decision tree construction of large datasets. *Data Min. Knowl. Discov.*, 4(2/3):127–162, 2000.

[18] J. Han and M. Kamber. *Data mining: concepts and techniques*. The Morgan Kaufmann series in data management systems. Elsevier, 2006. ISBN 9781558609013.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer series in statistics. Springer, 2008. ISBN 9780387848570.

[20] H. Hinnant. Combinations and permutations. 20 Mar. 2012 `http://home.roadrunner.com/~hinnant/combinations.html`.

[21] T. K. Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on Document Analysis and Recognition*, volume 1, pages 278 –282 vol.1, aug 1995.

[22] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3): 18–22, 2002.

[23] LSST Science Collaborations and LSST Project 2009. *LSST Science Book, Version 2.0*. 2009.

[24] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer, 2010.

[25] M. Mehta, R. Agrawal, and J. Rissanen. Sliq: A fast scalable classifier for data mining. In *Extending Database Technology*, pages 18–32, 1996.

[26] J. Mingers. An empirical comparison of selection measures for decision-tree induction. *Data Mining and Knowledge Discovery*, 3:319–342, 1989.

[27] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[28] S. K. Murthy. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, 2:345–389, 1997.

[29] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1:81–106, March 1986.

[30] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008.

[31] J. W. Richards, D. L. Starr, N. R. Butler, J. S. Bloom, J. M. Brewer, A. Crellin-Quick, J. Higgins, R. Kennedy, and M. Rischard. On machine-learned classification of variable stars with sparse and noisy time-series data. 733:10, 2011.

[32] L. Rokach and O. Maimon. *Data Mining with Decision Trees: Theory and Applications*. World Scientific Publishing Company, 2008.

[33] J. C. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India*, pages 544–555. Morgan Kaufmann, 1996.

[34] T. Sharp. Implementing decision trees and forests on a gpu. In *ECCV (4)*, pages 595–608, 2008.

[35] J. Shotton, A. W. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *CVPR*, pages 1297–1304, 2011.

Nomenclature

CART	Classification and Regression Trees.
CasJobs	Catalog Archive Server Jobs System
DM	Data mining
DR7	Data Release 7
DT	Decision trees
ID3	Iterative Dichotomiser 3
KDD	Knowledge discovery from data
LDA	Linear discriminant analysis
QSO	Quasi-stellar object
RDF	Random decision forests
RF	Random Forests
SDSS	Sloan Digital Sky Survey
TDIDT	Top-down induction of decision trees

Appendix A

Usage info

A.1 Program options

General options (General options are set via command line parameters. Other options must be specified in given configuration file.):

`-I (--input-file)` Path to the file with data in the ARFF format (5.4). Required.

`-C (--command)` The use case – possible values are: *run* or *train*. Required.

`-A (--algorithm)` The algorithm – possible values are: *rdf* (for random decision forest). Required.

`--config-file` The configuration file (A.2) with algorithm dependent options. Required.

Training options (`--command` is "train"):

`--model-out` The trained model will be serialized to the specified file. Required.

`--model-print` The trained model will be printed in XML to the specified file.

`--target-attribute` The index of the target attribute for training (zero-indexed). The default is to use last attribute in data.

Running options (`--command` is "run"):

`--output-file` The target file for the classification/prediction results. Required.

`--model-in` The file with trained model (from training; the –model-out option's file). Required.

RDF training options (command is "train" and algorithm is "rdf):

`rdf.trees` The number of trees. Recommended value is about 20-100. Required.

`rdf.min-size` The stopping criteria – minimum number of instances for splitting node. The default value is 4.

`rdf.max-depth` The stopping criteria – maximum depth of internal node. The default is 30.

`rdf.rand-attribs` The number of random attributes used in each node, zero for using all attributes. The default is zero.

`rdf.inbag-ration` Ratio of the bagged data to all data. The default is 1. A negative number for no bagging.

`rdf.min-sq-err` Only for numerical target attribute. If mean squared error in node is lower or equal than given value, node will not split. The default is 0.

A.2 Configuration file format

The configuration file format is described by self-explanatory example:

```
model-out = model.xml
rdf.trees = 20
[rdf]
min-size = 3
max-depth = 10
rand-attribs = 2
```

Appendix B

Graphical representation of the model

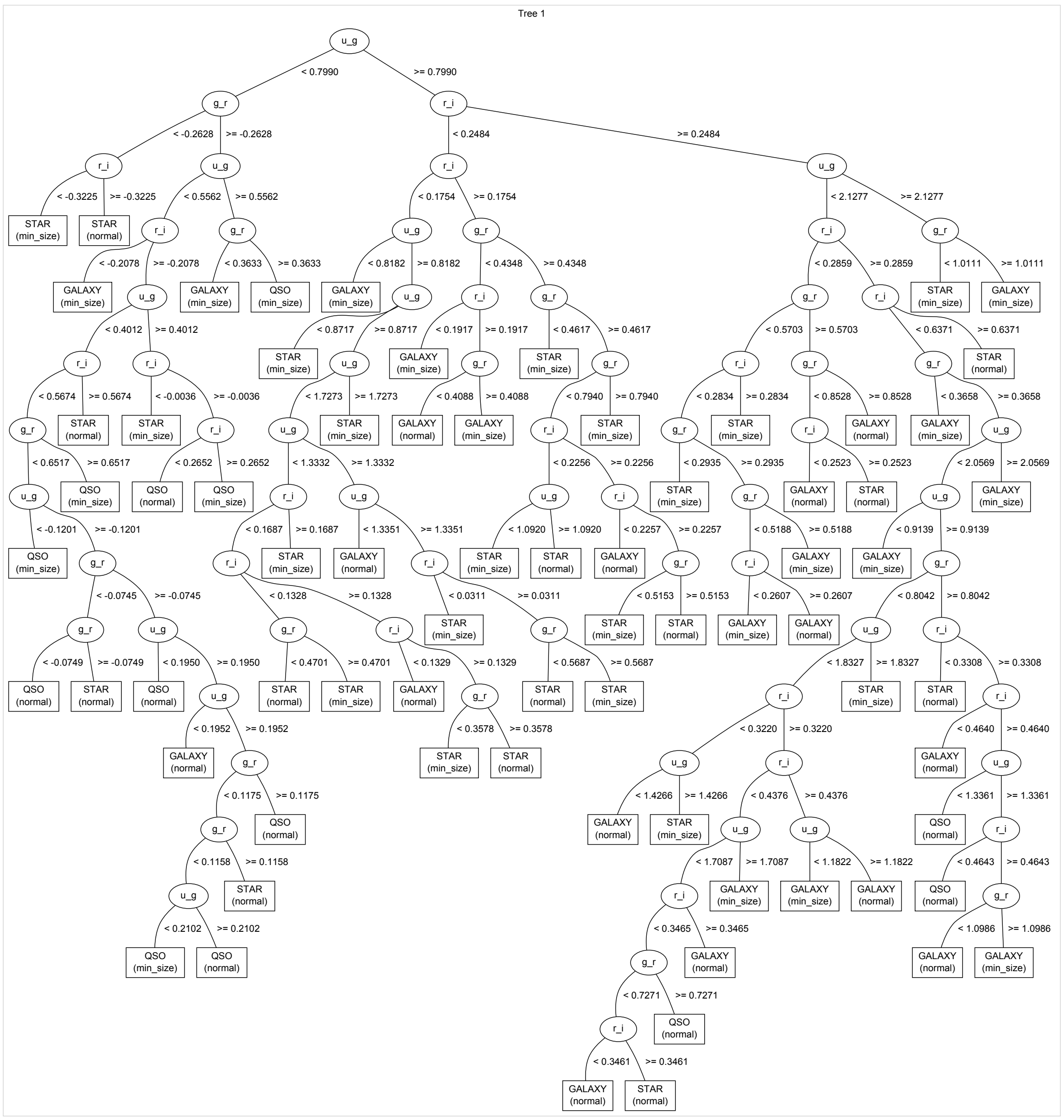


Figure B.1: A trained random decision forest (with only one tree) on the Stars data (unlimited depth, minimal size for splitting is 30 and two random attributes in each node).

Appendix C

Content of CD

In the folder *thesis* are the source codes of this thesis in Latex and Lyx format and the thesis in PDF format.

In the folder *bcrdf* are source codes of the presented implementation. There are also data used in experiments and scripts for running experiments. More detailed description of the folder structure and files is in *README* file.