THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

Adaptive Propositionalisation of Multi-Instance Data

Siva Manoharan

This report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computing and Mathematical Sciences with Honours (BCMS(Hons)) at The University of Waikato.

COMP520-12C (HAM)

Abstract

Multi-instance learning is a type of supervised machine learning where the class labels are attached to bags of instances. This is in contrast to single-instance learning where each individual instance is given a class label. Propositionalisation is the process of converting a multi-instance dataset into a single-instance dataset, allowing standard machine learning algorithms to train on the propositionalised dataset. In this context, the standard machine learning algorithms are known as base learners. We propose a novel multi-instance algorithm, `AdaProp`, which employs a propositionalisation approach that is influenced by the base learner. Hence, `AdaProp` is an adaptive propositionalisation algorithm and thus is able to produce propositionalised representations that are fitted more closely to the specific base learner than standard propositionalisation approaches. Multi-instance learning has been applied to datasets from several domains, including chemical datasets and text classification datasets. We examine one particular application of multi-instance learning, image classification, in detail and evaluate `AdaProp` on existing image classification datasets.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Dr Eibe Frank, for introducing me to the field of machine learning, for granting advice and guidance throughout the development of `AdaProp`, and for providing very useful feedback while writing this report. I would also like to thank the Department of Computer Science of the University of Waikato for making this project possible and for providing financial support.

Contents

List of Figures

Chapter 1

Introduction

Multi-instance (MI) learning is machine learning over multi-instance data, where the instances are grouped together into bags and the learning is performed over these bags rather than the individual instances. Multi-instance learning was originally proposed by Dietterich, Lathrop, and Lozano-Pérez (1997) in the context of drug-activity prediction. An application of multi-instance learning is image classification, where, for example, an image is represented as a bag of segments and the interaction between the segments contributes to the class of the image. In this context, each segment is an instance and thus each image is a bag of instances.

One approach to handling multi-instance data is propositionalisation, where each bag of instances is converted into a single feature vector. This converted dataset can be used with standard single-instance machine learning algorithms (i.e. base learners) such as SVMs and neural networks. In this project, we explore adaptive propositionalisation, where the base learner is used to make decisions when propositionalising the dataset. The propositionalisation approach considered involves the partitioning of the instance space into regions, where each region corresponds to one or more features in the propositionalised dataset. We propose a new multi-instance algorithm, `AdaProp`, which performs this adaptive propositionalisation. `AdaProp` has been implemented in the WEKA framework (Hall et al., 2009).

This report is structured as follows: Chapter 2 introduces concepts relevant to the rest of the report and discusses relevant previous work. Chapter 3 describes the `AdaProp` algorithm in detail. The results of the experiments conducted in this project are split across three chapters. Chapter 4 compares the various choices for the parameters of `AdaProp`, Chapter 5 examines the impact of some refinement techniques on the classification accuracy and Chapter 6 compares `AdaProp` to existing multi-instance algorithms.

Chapter 2

Background

This chapter discusses relevant previous work in the field of multi-instance learning and introduces several key concepts necessary for the rest of the report. Section 2.1 briefly introduces supervised machine learning, while Section 2.2 discusses multi-instance learning and describes several of the multi-instance assumptions used by previous work in this field. Section 2.3 outlines some of the applications of multi-instance learning and examines the image classification application in more detail. This section also specifies how the image classification datasets used in this project were prepared. Section 2.4 summarises some of the existing multi-instance algorithms, including those which perform propositionalisation of multi-instance data. Among these algorithms, we examine two which are closely related to `AdaProp`: TLC (Section 2.4.1) and RELAGGS (Section 2.4.2).

2.1 Supervised machine learning

In supervised machine learning, the training dataset is a set of instances, where each instance consists of a fixed number of attributes and a single class label. The learner is then expected to infer the relationship between the attributes and the class label, resulting in a model which expresses the class label as some function of the attribute values. This model can then be used to predict the class label of any instance from the same underlying domain, including instances which were not part of the original training dataset.

In this project, we consider a generalisation of supervised machine learning, known as multi-instance learning, where the training dataset consists of labelled bags of instances, each of which consists of a set of instances along with a class label.

2.2 Multi-instance learning

Multi-instance (abbreviated MI) learning was introduced by Dietterich et al. (1997) in the context of drug activity prediction. The MI learning problem was defined as a two-class supervised learning problem where the classification occurs over bags of instances rather than individual instances. Dietterich et al. (1997) assumed that each instance has a hidden class label and that a bag belongs to the positive class if and only if at least one of the instances in the bag belongs to the positive class. This assumption is known as the "standard MI assumption" and fits the original domain of drug activity prediction well.

In this project we also consider datasets which may not satisfy the standard MI assumption. Therefore, we require MI assumptions which are more general than the standard assumption. Chen, Bi, and Wang (2006) proposed a generalised MI assumption in which the label of each bag is determined by the distance of the bag's instances to some hidden target points in instance space. Weidmann, Frank, and Pfahringer (2003) introduced a hierarchy of generalisations of the standard MI assumption: presence-based, threshold-based and count-based assumptions.

All generalised assumptions in the Weidmann et al. (2003) hierarchy presume that, for each dataset, there exist a set of hidden concepts, i.e. conditions predicated on the instances. The assumptions also presume that the class label of each bag is determined by the number of instances in the bag which satisfy each concept. Under the presence-based MI assumption, a bag is positive if and only if, for each concept, there exists an instance in the bag which satisfies the concept. Under threshold-based MI assumption, a bag is positive if and only if, for each concept, the number of instances which satisfy the concept is greater than or equal to some threshold. Finally, under count-based MI assumption, the most general of the assumptions in the hierarchy, a bag is positive if and only if, for each concept, the number of instances which satisfy the concept is greater than or equal to some lower threshold and is less than or equal to some upper threshold.

The assumptions form a hierarchy of specialisations: the standard MI assumption can be considered to be a specialisation of the presence-based assumption (where there is only one concept), which in turn can be considered to be a special case of threshold-based assumption (where the threshold is exactly 1), which also is a specialisation of the count-based assumption (where the upper threshold is infinite). The algorithm developed in this project, `AdaProp`, has been designed for datasets corresponding to the most general of these MI assumptions, the count-based assumption. Therefore `AdaProp` is also expected to be able to handle the datasets which satisfy the more specialised MI assumptions.

2.3 Applications

In the original application of multi-instance learning, by Dietterich et al. (1997), each bag represents a single molecule, and each instance of the bag corresponds to a single conformation of the molecule. In this dataset, the "musk" dataset, the class label is positive if and only if the molecule emits a musky odour, in any of its conformations. Multi-instance learning has also been applied to other domains such as image and text classification where the standard MI assumption is less appropriate. For example, Maron and Lozano-Pérez (1998) applied multi-instance learning (more specifically, the Diverse Density algorithm), to several domains, including stock market prediction. As image classification is the target application for this report, we now consider it in more detail.

2.3.1 Image classification

The problem of image classification fits the multi-instance learning setting especially well. In a standard machine learning context, each image must be represented as a single fixed-size feature vector. Since an image may contain multiple objects, a single feature vector may not capture all of the relevant information in the image. Also, since an image can contain pixels which do not correspond to any objects of interest (i.e. background pixels), and some objects maybe partially occluded, a single feature vector can also contain irrelevant information. In general, global representations (which describe the entire image) are not well suited to image classification. Instead, local representations (which describe an image as a set of smaller subregions) are better at capturing the information in an image (Grauman & Leibe, 2011). As a MI learning problem, an image can be represented as a bag of instances, where each instance is a region of the image.

There have been several proposed approaches for converting an image into a multi-instance representation. For example, Maron and Ratan (1998) convert images of natural scenes into a multi-instance dataset by simply partitioning each image into small (2×2 pixel) fixed-size regions. Each such region is converted into an instance by extracting color-based features from the region. Similarly, Yang and Lozano-Perez (2000) convert each image by dividing the image into multiple large overlapping regions of various sizes, where the smallest regions contained a quarter of the pixels of the original image. In contrast, Zhang, Goldman, Yu, and Fritts (2002) use K-means segmentation to obtain the subregions of each image. Another proposed approach is the one used by the SIFT algorithm (Lowe, 2004), which identifies points of interest (i.e. points where the gradient changes rapidly) and extracts small regions surrounding these keypoints.

In this project, we consider datasets prepared by two different multi-instance image representation approaches. The first approach, performed using the Blobworld system (Carson, Thomas, Belongie, Hellerstein, & Malik, 1999), segments each image into blobs (regions) by considering the colour, texture and position of each pixel, while each blob is described, as a feature vector, using colour and texture information only. A key detail of this approach is that each blob partially is described by a binned colour histogram in $L^*a^*b^*$ space, which is a colour space that closely matches the human vision system. In our experiments, we use the datasets `tiger`, `fox` and `elephant` which were prepared using this approach by Andrews, Tsochantaridis, and Hofmann (2002). The second approach, used by Mayo and Frank (2011), divides each image into a fixed number of subregions and extracts features from each subregion. The features extracted are the histograms of LBP (local binary patterns) and histograms in Ohta colour space, capturing texture and colour information respectively. The key detail with this approach is that both the Ohta colour transformation and LBP feature extraction are computationally efficient, allowing large datasets (e.g. with 800 images) to be prepared quickly. The datasets `people`, `bikes` and `cars`, used in our experiments, were prepared using this approach.

2.4 Multi-instance algorithms

A number of algorithms for learning on MI data have been proposed. The original algorithm proposed by Dietterich et al. (1997) used an axis parallel rectangle to identify the region of the instance space where the positive instances lay. The algorithm was designed to find the region which contained at least one instance from each positive bag but no instances from the negative class. Maron and Lozano-Pérez (1998) proposed the Diverse Density algorithm, where the aim was to find target points in instance space which are close to at least one instance from each positive bag and far away from all instances in negative bags. Both algorithms were designed specifically for MI data.

Another approach to handling multi-instance data is to adapt existing standard machine learning algorithms. MISVM (Andrews et al., 2002) is an example of this approach, where the authors extend the concept of a margin. In standard SVMs, the margin is a function dependent on the individual instances, while in MISVM, the margin is a function of all instances in the bag. This bag-margin is then directly minimised to obtain the MISVM classifier. In this project we consider a more general approach, that of propositionalisation, where the multi-instance dataset is converted into a single-instance representation, allowing any existing standard machine learning algorithm to train on the dataset.

There are some existing approaches to propositionalisation in the literature. Chen et al. (2006) introduced MILES, where each bag is propositionalised by considering the shortest distance between any instance in the bag and some target points in the instance space. TLC, an algorithm proposed by Weidmann et al. (2003), partitions the instance space and propositionalises each bag by counting the number of instances which fall into each partition. In RELAGGS (Krogel & Wrobel, 2003), each bag is propositionalised by computing summary statistics across all instances in each bag. Both TLC and RELAGGs are closely related to our algorithm, `AdaProp`, thus we examine them in more detail.

2.4.1 TLC

For multi-instance datasets subject to the generalised MI assumptions (for example, the count-based assumption), Weidmann et al. (2003) proposed a two level learning approach, aiming to separate the learning of the (hidden) instance labels from the learning of the bag labels. In the first level, a C4.5 decision tree (`J48` in WEKA) was built to partition the instance space. This was done by assuming that each instance simply inherited the class label of its parent bag, i.e. by ignoring any bag-level structure. In the second level, the occupancy counts of instances of each bag in each region was used to build a propositionalised dataset, to which a standard machine learner was applied. The aim of the first level is to determine the structure of the overall instance space, while the aim of the second level is to infer the relationship between the instances and the class label of the bag. `AdaProp` uses a similar process as Weidmann et al. (2003) in the second level, but differs from TLC as an adaptive approach is used to partition the instance space.

2.4.2 RELAGGS

RELAGGS was introduced by Krogel and Wrobel (2003) for general relational datasets, which was then specialised for multi-instance datasets as a propositionalisation algorithm in the WEKA software. RELAGGS is a simple algorithm, which ignores any structure between the instances of each bag. Instead, all instances of each bag are grouped together and the summary statistics (min, max, mean, standard deviation and sum) of each bag for each attribute are computed, forming the propositionalised dataset. `AdaProp` with summary-based propositionalisation can be considered to be a generalisation of RELAGGS, as summary-based `AdaProp` also computes summary statistics over instances, albeit after dividing up the instances of each bag by a tree of partitions.

Chapter 3

AdaProp

The approach investigated in this project, named `AdaProp`, is an adaptive propositionalisation algorithm for multi-instance datasets. `AdaProp` divides the instance space into regions and then propositionalises each bag by computing summary statistics for the subset of instances of the bag which lie in each region. `AdaProp` determines the regions by repeatedly splitting the instance space into two partitions. Therefore, `AdaProp` consists of three major components: a base learner (any single-instance learning algorithm), a process for partitioning the instance space, and a process for constructing the propositionalised dataset using the regions.

This chapter is organized as follows: first we introduce some notation and definitions in Section 3.1, followed by the approach used to build the tree of partitions in Section 3.2 and the methods of propositionalisation in Section 3.3. Section 3.4 briefly describes the base learners considered in this project, while Section 3.5 discusses some of the techniques used to improve the cross-validated classification accuracy of `AdaProp`.

3.1 Definitions

A multi-instance dataset $\mathcal{D}$ is a set of labelled bags, where each labelled bag is a set of instances with a class label. Each instance in each bag has a set of k attributes and thus can be considered a vector in $\mathbb{R}^k$ (assuming all attributes are numeric). Therefore, the set of all instances, $\mathcal{I}$, can be defined as a set of k-dimensional vectors. Thus, $\mathcal{I} \subseteq \mathbb{R}^k$.

Each labelled bag in the dataset $\mathcal{D}$ is composed of a set of instances from $\mathcal{I}$, along with a class label. Let $\mathcal{C}$ be the set of all possible class labels. Then we can define $\mathcal{D}$, the multi-instance dataset, as $\mathcal{D} \subseteq (\mathbb{P}(\mathcal{I}) \times \mathcal{C})$, where $\times$ denotes the Cartesian product.

The propositionalisation process can be viewed as a function mapping each labelled bag $(B_i, c_i) \in \mathcal{D}$ to a single labelled instance $\boldsymbol{p_i} \in (\mathbb{R}^j \times \mathcal{C})$. Note that $\boldsymbol{p_i}$ has j (non-class) attributes, which need not be equal to k, the number of attributes of each instance in the original dataset. Also note that the propositionalisation process does not modify the class label.

3.2 Partitioning the instance space

For the purposes of finding an appropriate propositionalisation, we first compute $\mathcal{M}$, an intermediate labelled dataset consisting of all instances in $\mathcal{D}$. $\mathcal{M}$ is built up by collecting together all instances from all the bags in the dataset and attaching the class label of each bag to each instance in the bag[1]. Formally, $\mathcal{M} \subseteq (\mathcal{I} \times \mathcal{C})$, where each instance of $\mathcal{M}$ appears in $\mathcal{D}$:

$$\forall\ (\boldsymbol{a}, c_i) \in \mathcal{M}\ :\ \exists\ (B, c_b) \in \mathcal{D}\ :\ \boldsymbol{a} \in B\ \wedge\ c_i = c_b$$

and each instance of $\mathcal{D}$ appears in $\mathcal{M}$:

$$\forall\ (B, c_b) \in \mathcal{D}\ :\ \forall\ \boldsymbol{a} \in B\ :\ (\boldsymbol{a}, c_b) \in \mathcal{M}$$

For example, consider the small two-class dataset in Table 3.1, consisting of two bags ($|\mathcal{D}| = 2$) containing five instances in total, each with two attributes ($k = 2$). Figure 3.1 shows the intermediate labelled dataset $\mathcal{M}$ for this example dataset with positive instances rendered as squares and negative instances rendered as triangles.

After computing $\mathcal{M}$, our algorithm aims to partition the instance space of $\mathcal{M}$ into regions. These regions are found by an iterative greedy algorithm (Algorithm 1), which builds up a tree of partitioning hyperplanes. For the purposes of this algorithm, a partitioning hyperplane is a hyperplane in the instance space, of the form $\boldsymbol{w} \cdot \boldsymbol{a} = c$, where c and $\boldsymbol{w}$ are parameters of the hyperplane and $\boldsymbol{a}$ is the vector of attribute values. This hyperplane represents a natural partitioning of the instance space into two regions: the instances which lie above the hyperplane, i.e. $\boldsymbol{i} \in \mathcal{I}$ where $\boldsymbol{w} \cdot \boldsymbol{i} > c$; and the instances which lie at or below the hyperplane, i.e. $\boldsymbol{i} \in \mathcal{I}$ where $\boldsymbol{w} \cdot \boldsymbol{i} \leq c$.

At each iteration, the algorithm generates a list of candidate partitioning hyperplanes and selects the best hyperplane to add to the tree. To reduce the search space, `AdaProp`,

[1] The instance class labels are required by our algorithm when applied with discretization-based heuristic discussed in Section 3.2.

Bag	Class	Instance	Attribute a_1	Attribute a_2
b_1	positive	i_1	0.3	0.7
		i_2	0.5	0.1
b_2	negative	i_3	0.2	0.9
		i_4	0.8	0.6
		i_5	0.5	0.7

Table 3.1: Example dataset

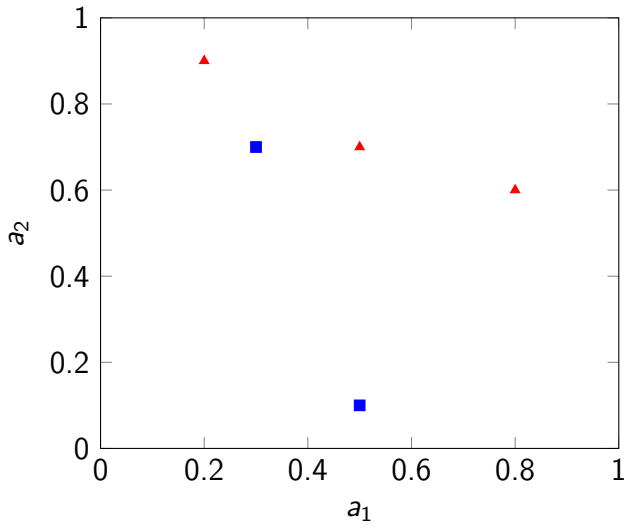


Figure 3.1: Visualisation of $\mathcal{M}$ for the example dataset

Algorithm 1 Building a tree of partitioning hyperplanes

Initialise T, the tree of partitioning hyperplanes as a tree with a single node
while T is not satisfactory **do** (Section 3.2.1)
  $n_i \leftarrow$ Select a leaf node in the tree T (Section 3.2.2)
  $\mathcal{M}_i \leftarrow \{(\boldsymbol{a}, c) \in \mathcal{M} \mid \boldsymbol{a}$ lies in the region corresponding to $n_i\}$
  $H \leftarrow$ Generate candidate partitioning hyperplanes for $\mathcal{M}_i$ (Section 3.2.3)
  $H_i \leftarrow$ Select the optimal hyperplane in H (Section 3.2.4)
  Make n_i an internal node in T, corresponding to the hyperplane H_i
end while

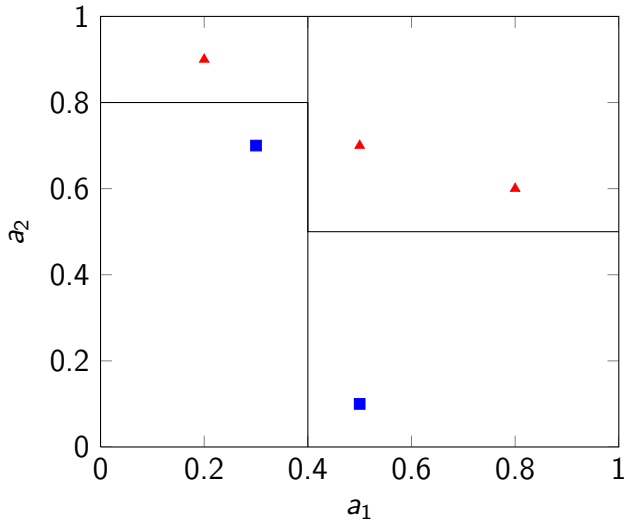


Figure 3.2: Partitioning of $\mathcal{M}$ for the example dataset

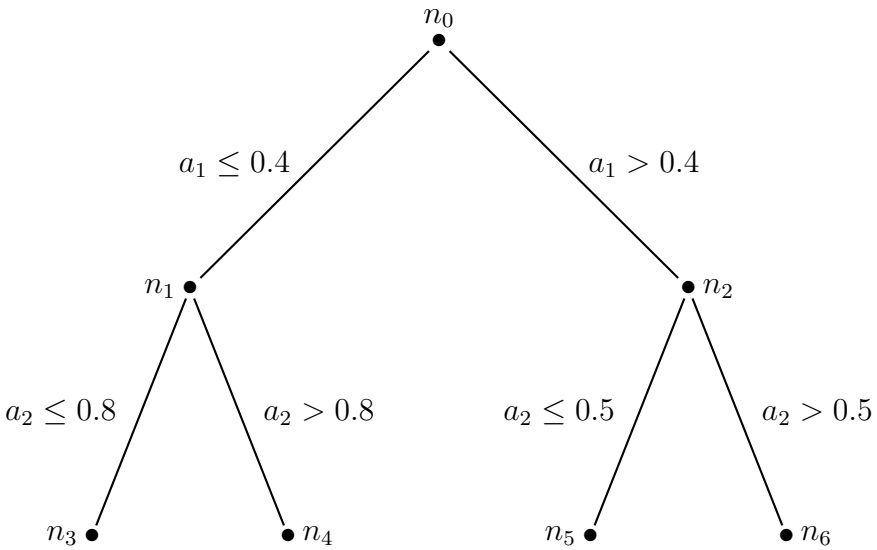


Figure 3.3: A tree corresponding to the partitioning of $\mathcal{M}$ in Figure 3.2

as evaluated in this project, only considers hyperplanes which correspond to testing just one attribute, i.e. hyperplanes which intersect exactly one axis. Therefore, each partition in this algorithm corresponds to a hyperplane of the form $a_i = c$.

Figure 3.2 shows a possible partitioning of the instance space for the example dataset from Table 3.1. Figure 3.3 shows the corresponding tree of partitioning hyperplanes built by the algorithm. We now consider the steps performed by the algorithm in detail.

3.2.1 Stopping conditions

The `AdaProp` algorithm continues to iterate while none of the following stopping conditions are met:

The tree becomes too big:
$Depth(T) \geq maxDepth$, where $maxDepth$ is a user specified parameter.

Each region corresponding to a leaf node contains too few instances:
$\forall\ n_i \in LeafNodes(T)\ :\ Occupancy(n_i) < minOcc$, where $Occupancy(n_i)$ is the number of instances of $\mathcal{M}$ which lie in n_i and $minOcc$ is a user specified parameter.

The tree is sufficiently accurate for the training set:
$Error(BaseLearner, \mathcal{D}_T) < minErr$, where $\mathcal{D}_T$ is the dataset $\mathcal{D}$ propositionalised using T (see Section 3.3), $Error$ is the misclassification error on the training set when the $BaseLearner$ is trained on $\mathcal{D}_T$, and $minErr$ is a user specified parameter.

3.2.2 Selecting a leaf node to expand

At each iteration of the algorithm, a leaf node of T is selected to be expanded into an internal node of the tree. The leaf nodes which are eligible to be expanded are those which, when expanded, will not violate the $maxDepth$ condition. Let $\mathcal{N}$ be the subset of the leaf nodes of T which are eligible for expansion:

$$\mathcal{N} \leftarrow \{n_i \in LeafNodes(T) \mid Depth(n_i) < maxDepth - 1\}$$

We assume that the nodes in T are indexed in breadth first order, as shown in Figure 3.3. Then, among any subset of nodes with equal depth, the leftmost node is the node which has the least index. This concept of leftmost-node is used to break ties in the leaf node selection strategies. `AdaProp` supports three leaf node selection strategies:

Depth first search:

The node selected is the leaf node which is leftmost node among the leaf nodes which have the greatest depth in $\mathcal{N}$.

$greatestDepth \leftarrow \max\{depth(n_j) \mid n_j \in \mathcal{N}\}$
$n_i \leftarrow \text{leftmost}\{n_j \in \mathcal{N} \mid depth(n_j) = greatestDepth\}$

Breadth first search:

The node selected is the leaf node which is leftmost node among the leaf nodes which have the least depth in $\mathcal{N}$.

$leastDepth \leftarrow \min\{depth(n_j) \mid n_j \in \mathcal{N}\}$
$n_i \leftarrow \text{leftmost}\{n_j \in \mathcal{N} \mid depth(n_j) = leastDepth\}$

Best first search:

Given a heuristic function $h(n)$ such that $h(n_j) < h(n_i)$ iff n_j is a better node to expand than n_i, the node selected is the leaf node which is the leftmost among the leaf nodes which have the least value for $h(n)$ in $\mathcal{N}$.

$leastH \leftarrow \min\{h(n_j) \mid n_j \in \mathcal{N}\}$
$n_i \leftarrow \text{leftmost}\{n_j \in \mathcal{N} \mid h(n_j) = leastH\}$

`AdaProp` supports only one heuristic function, the $Error(n)$, which is the training set error on the dataset $\mathcal{D}$ propositionalised using T along with the best partitioning hyperplane at n. Computation of $Error(n)$ involves evaluating all candidate partitions at a given node, thus $Error(n)$ is computationally expensive.

3.2.3 Generating candidate partitioning hyperplanes

Given the set of labelled instances $\mathcal{M}_i \subseteq \mathcal{M}$, we wish to generate a set of candidate partitioning hyperplanes for $\mathcal{M}_i$ in the instance space. As mentioned above, `AdaProp` only considers hyperplanes which intersect exactly one axis (thus are parallel to all other axes). We consider four methods for generating candidate hyperplanes. The first three all attempt to find balanced partitioning hyperplanes, i.e. those which lie somewhat near the center of the instances in $\mathcal{M}_i$. Therefore, the first three methods follow the same template:

for $j = 1 \rightarrow k$ **do**
  $V_j \leftarrow \{a_j \mid \exists c \in \mathcal{C} \ : \ (\boldsymbol{a}, c) \in \mathcal{M}_i\}$
  $m \leftarrow findPt(V_j)$
  Output candidate hyperplane: $a_j = m$
end for

The three different instantiations of the $findPt()$ method are as follows:

Range based midpoint:

For each attribute a_j, the range (i.e. minimum and maximum) of values of a_j for the instances in $\mathcal{M}_i$ is computed, and a candidate hyperplane corresponding to the midpoint between the minimum and maximum values is generated.

$midpt(V_j) \leftarrow \frac{\min(V_j) + \max(V_j)}{2}$

Mean:

For each attribute a_j, the candidate hyperplane generated corresponds to the mean value of a_j for all instances in $\mathcal{M}_i$.

$midpt(V_j) \leftarrow \text{mean}(V_j)$

Median:

For each attribute a_j, the candidate hyperplane generated corresponds to the median value of a_j for all instances in $\mathcal{M}_i$.

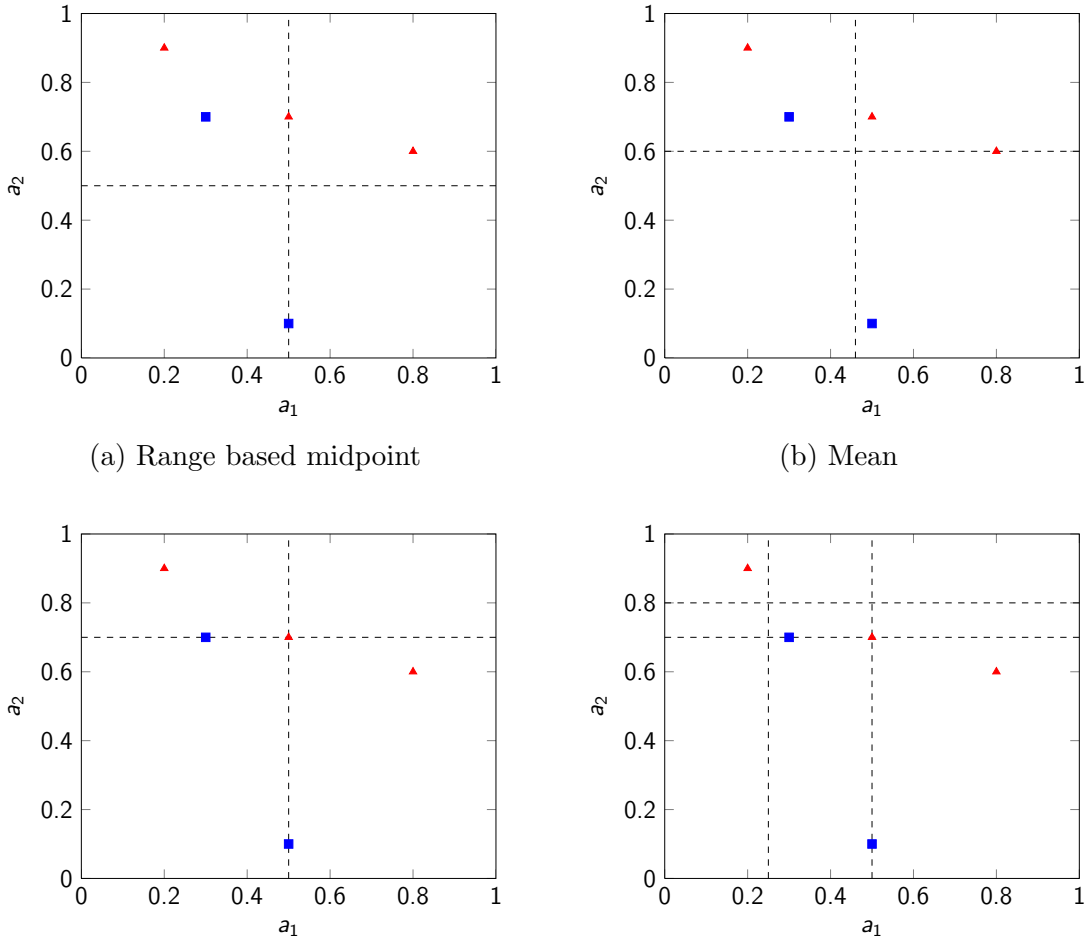
$midpt(V_j) \leftarrow \text{median}(V_j)$

The fourth method is different to the above three methods as it (potentially) generates multiple candidate partitions per attribute. For each attribute a_j, the values of a_j for all instances in $\mathcal{M}_i$ are sorted, and the values at which the class changes (i.e. the class boundaries) are used to generate the partitioning hyperplanes. This is similar to the discretization process of OneR (Holte, 1993), thus is called the discretization-based method.

Discretization-based:

for $j = 1 \rightarrow k$ **do**
  $W_j \leftarrow \{(a_j, c) \mid (\boldsymbol{a}, c) \in \mathcal{M}_i\}$
  Sort W_j
  for all values $(w_j, c_j) \in W_j$ where $c_{j-1} \neq c_j$ **do**
    $m \leftarrow \frac{w_{j-1}+w_j}{2}$
    Output candidate hyperplane: $a_j = m$
  end for
end for

See Figure 3.4 for a visualisation of all the candidate partitioning hyperplanes generated by each method for the example dataset in Table 3.1.



(a) Range based midpoint

(b) Mean

(c) Median

(d) Class boundaries

Figure 3.4: Candidate partitions generated for the $\mathcal{M}$ corresponding to the example dataset

Algorithm 2 Selecting the optimal partitioning hyperplane

function EVALUATEHYPERPLANE(T, n_i, H_j)
  $T_j \leftarrow T$ with the hyperplane H_j added at the node n_i
  $\mathcal{D}_{T_j} \leftarrow \mathcal{D}$ propositionalised using T_j
  return $Error_{eval\_metric}(BaseLearner, \mathcal{D}_{T_j})$
end function

function FINDOPTIMALHYPERPLANE(T, n_i, H)
  $minErr = \min\{\text{EVALUATEPARTITION}(T, n_i, H_j) \mid H_j \in H\}$
  return any$\{H_j \in H \mid \text{EVALUATEPARTITION}(T, n_i, H_j) = minErr\}$
end function

3.2.4 Selecting the optimal partitioning hyperplane

Given a set of candidate partitioning hyperplanes H for the labelled set of instances $\mathcal{M}_i$, `AdaProp` aims to find the optimal hyperplane H_i in H, which is the hyperplane with the least error (as measured using some hyperplane evaluation metric) on $\mathcal{D}$ when using the base learner.

The overall process is specified in Algorithm 2. To summarize: for each candidate hyperplane H_j, a new tree T_j is obtained by adding H_j to T. Then T_j is used to propositionalise $\mathcal{D}$ and the base learner is trained on the propositionalised dataset, resulting in a classifier. This classifier is then evaluated on the propositionalised training set, using the hyperplane evaluation metric. The candidate hyperplane which results in the least error during this evaluation process is selected as the optimal hyperplane.

In this project, we consider three hyperplane evaluation metrics:

Classification error:

The classification error is the misclassification rate, i.e. the number of incorrectly classified bags divided by the total number of bags in $\mathcal{D}$. Formally:

$$CE = \frac{|\{\ (b,c) \in \mathcal{D} \mid f(b) \neq c\ \}|}{|\ \mathcal{D}\ |}$$

where $f(b)$ is the class label predicted by the base learner for the bag b.

Root mean squared error:

The root mean squared error takes into account the probability estimates emitted by the classifier. As suggested by its name, the root mean squared error is computed by taking the square root of the average squared error over all bags, where the error for each bag is defined as the difference between the actual probability and the probability estimate emitted by the classifier. Note that the actual probability of a bag B taking on a specific class value c is defined as 1 if B has the class label c attached, and 0 otherwise. Formally:

$$RMSE = \sqrt{\frac{1}{|\mathcal{D}| \cdot |\mathcal{C}|} \sum_{(b,c_b)\in\mathcal{D}} \sum_{c\in\mathcal{C}} \Big(g(b,c) - I(c,c_b)\Big)^2}$$

$$I(c,c_b) = \begin{cases} 1 & : \quad c = c_b \\ 0 & : \quad c \neq c_b \end{cases}$$

where $g(b,c)$ is the probability, as estimated by the base learner, that b has class c.

Information gain:

The information gain, in this context, is defined as the difference in entropy between the probability estimates emitted by the classifier and the null model. The hyperplane evaluation metric based on information gain is implemented as the negation of the information gain value, in order to obtain an error function.

$$IG = \sum_{(b,c)\in\mathcal{D}} \Big(h_g(b,c) - h_z(b,c) \Big)$$

$$h_g(b,c) = -g(b,c)\ \ln g(b,c)$$

$$h_z(b,c) = -z(b,c)\ \ln z(b,c)$$

where $g(b,c)$ is the probability that bag b has class c, as estimated by the base learner, and $z(b,c)$ is the probability that bag b has class c as estimated by the null model (i.e. `ZeroR` in WEKA). Note that both h_g and h_z are entropy functions.

3.3 Propositionalisation

The tree of partitioning hyperplanes T defines a set of regions in the instance space which cover the instance space completely: each node in the tree corresponds to a region, described by the tests used along the path to that node. Each such region will correspond to one or more attributes in the propositionalised dataset. `AdaProp` implements two different approaches to propositionalisation of each bag (Algorithm 3): count-based propositionalisation and summary-based propositionalisation.

3.3.1 Count-based propositionalisation

In count-based propositionalisation, each bag is converted into a feature vector by counting the number of instances of the bag which fall into each region. Therefore, each region corresponds to a single attribute in the propositionalised vector. Formally, the Prop function in Algorithm 3 for count-based propositionalisation is given by:

$$\textsc{Prop} : \mathbb{P}(\mathcal{I}) \to \mathbb{R},\ \text{where}\ \textsc{Prop}(X) = |X|$$

Table 3.2 shows the count-based propositionalised form of the example dataset given in Table 3.1, partitioned by the tree shown in Figure 3.3.

Algorithm 3 Propositionalisation (Given a function $\textsc{Prop} : \mathbb{P}(\mathcal{I}) \to \mathbb{R}^m, m \geq 1$)

Initialise the propositionalised dataset $\mathcal{P}$ as an empty dataset
for all bags $(B_i, c) \in \mathcal{D}$ **do**
  Initialise the propositionalised vector $\boldsymbol{p}$ as an empty list
  for all nodes n_j in the tree T **do**
    $X^{(j)} \leftarrow \{i \in B_i \mid i \text{ lies in the region corresponding to } n_j\}$
    $\boldsymbol{p} \leftarrow \boldsymbol{p} \mid\mid \textsc{Prop}(X^{(j)})$ ▷ $\mid\mid$ denotes concatenation
  end for
  Add the labelled vector $(\boldsymbol{p}, c)$ to the dataset $\mathcal{P}$
end for

Bag	n_0	n_1	n_2	n_3	n_4	n_5	n_6	Class
b_1	2	1	1	1	0	1	0	positive
b_2	3	1	2	0	1	0	2	negative

Table 3.2: Count-based propositionalised form of the example dataset

Bag	$n_0\ a_1$				$n_0\ a_2$				$n_1\ a_1$				...
	min	max	sum	avg	min	max	sum	avg	min	max	sum	avg	
b_1	0.3	0.5	0.8	0.4	0.1	0.7	0.8	0.40	0.3	0.3	0.3	0.3	
b_2	0.2	0.8	1.5	0.5	0.6	0.9	2.2	$0.7\dot{3}$	0.2	0.2	0.2	0.2	

...	$n_1\ a_2$				$n_2\ a_1$				$n_2\ a_2$				Class
	min	max	sum	avg	min	max	sum	avg	min	max	sum	avg	
	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.70	0.1	0.1	0.1	0.10	positive
	0.9	0.9	0.9	0.9	0.5	0.8	1.3	0.65	0.6	0.7	1.3	0.65	negative

Table 3.3: Summary-based propositionalised form of the example dataset (showing only the nodes n_0, n_1, and n_2).

3.3.2 Summary-based propositionalisation

In summary-based propositionalisation, the instances which fall into each region are aggregated using summary statistics, such as minimum, maximum, sum, and average (i.e. mean). For each region, the summary statistics are computed for all attributes, regardless of which attribute was used in the chosen partitioning hyperplane. Therefore, summary-based propositionalisation is more computationally expensive than count-based propositionalisation, but is able to preserve significantly more information from each bag in the propositionalisation.

Table 3.3 shows the propositionalised form of the example dataset given in Table 3.1 when using summary-based propositionalisation.

Formally, the PROP function in Algorithm 3 for summary-based propositionalisation is given by:

$$\textsc{Prop} : \mathbb{P}(\mathcal{I}) \to \mathbb{R}^{4k}$$

$$\begin{aligned}\textsc{Prop}(X) = (\,&\min(X_1),\ \max(X_1),\ \text{sum}(X_1),\ \text{avg}(X_1),\\ &\min(X_2),\ \max(X_2),\ \text{sum}(X_2),\ \text{avg}(X_2),\\ &\ldots,\\ &\min(X_k),\ \max(X_k),\ \text{sum}(X_k),\ \text{avg}(X_k))\end{aligned}$$

where k is the number of attributes and $X_j = \{i_j \mid i \in X\}$, i.e. the values of the j^{th} attribute for all instances in X.

3.4 The base learner

Any standard machine learning algorithm can be used as the base learner to guide the hyperplane selection process. The choice of base learner is left up to the user. This report considers two common base learners for the experiments: `RandomForest` and `LogitBoost`.

A random forest (Breiman, 2001) constructs an ensemble of randomized decision trees, where each node of each tree is built by selecting from a random subset of the attributes. In this project, random forests with 100 trees were used, using the `RandomForest` implementation in the WEKA software.

`LogitBoost`, introduced by Friedman, Hastie, and Tibshirani (2000), is an adaptive boosting algorithm which minimises a logistic loss function. In this project, LogitBoost with 50 iterations was used as a compromise between runtime and accuracy, using decision stumps (1-level decision trees, implemented as `DecisionStump` in WEKA) as the base learner.

3.5 Refinements

Initial experiments (discussed in Chapter 4) indicated that `AdaProp` significantly overfits some datasets. In order to reduce this overfitting, and thus improve the accuracy of `AdaProp`, two standard techniques for reducing overfitting were implemented: cross validated parameter selection and randomized bagging.

3.5.1 Parameter selection

Initial results showed that the size of the trees of partitioning hyperplanes built up by `AdaProp` has substantial impact on the cross-validated classification accuracy. In fact, the experiments showed that increasing the tree size increased the accuracy initially up to some maximum, after which it declined. This suggests that there is some optimal size for the tree of partitioning hyperplanes. From the experiments, it was also clear that this optimal size is dependent on the dataset, i.e. different datasets have different optimal tree sizes.

Thus, in order to find an appropriate tree size for any given dataset, `AdaProp` can perform 5-fold cross-validated selection over all tree sizes (Algorithm 4). This process maintains a set of 5 trees, each built on a different 80% of the training dataset, and evaluated on the remaining 20%. When partitioning the dataset, stratification is used to ensure that each partition has approximately the same distribution of class labels as the entire dataset. The size of each tree is then iteratively increased until the total error on the test datasets ceases to decrease.

This process requires more time than running the `AdaProp` process with a fixed tree size, but provides an increase in accuracy. However, compared to the usual method of parameter optimisation, where the entire training process is repeated for each possible value of the parameter and for each fold of the cross-validation, this method is significantly more efficient.

Algorithm 4 Cross-validated tree-size selection

Partition, with stratification, the training set into 5 folds: $\{X_1, X_2, \ldots, X_5\}$.
for $j = 1 \to 5$ **do**
  Build a tree T_j of size 1, using all folds except X_j as the training set.
  $e_j^0 \leftarrow 1$ ▷ e_j^t is the error of the tree T_j when of size t
  $e_j^1 \leftarrow$ the misclassification error rate of T_j when evaluated on X_j.
end for
$t \leftarrow 1$ ▷ t is the current tree size.
while $e_1^t + e_2^t + \ldots + e_5^t \ \leq \ e_1^{t-1} + e_2^{t-1} + \ldots + e_5^{t-1}$ **do**
  $t \leftarrow t + 1$
  for $j = 1 \to 5$ **do**
    Expand T_j by one node, using all folds except X_j as the training set.
    $e_j^t \leftarrow$ the misclassification error rate of T_j when evaluated on X_j.
  end for
end while
Output $t - 1$ as the optimal tree size.

3.5.2 Randomized bagging

Another refinement implemented to combat overfitting is randomized bagging. Bagging was performed using WEKA's standard bagging implementation, where the training set is resampled (by sampling bags with replacement) to generate n independent datasets, where n is the number of iterations. The resampling is performed such that each new dataset is the same size as, i.e. contains the same number of bags as, the original dataset. Then, an ensemble of n `AdaProp` trees is built (i.e. one tree per dataset), and the average of each tree's class probability estimate is used as the ensemble's probability estimate.

When bagging is performed on standard `AdaProp` trees, the runtime grows linearly with the number of iterations (i.e. 50 iterations of bagging will take 50 times longer to complete than training a single `AdaProp` tree). In order to reduce the runtime and also to further reduce overfitting, randomization in attribute selection was implemented, where at each node in each tree, only a randomly chosen subset of attributes was considered when selecting the optimal partitioning hyperplane. The subset of attributes was chosen to be of size $\log_2(K) + 1$ (as in WEKA's `RandomForest`), where K is the number of attributes. Therefore the randomization reduced the runtime by a factor of $\frac{K}{\log_2(K)+1}$.

Chapter 4

Results: `AdaProp` parameters

We conducted a number of experiments over a set of multi-instance datasets, with the aim of identifying the impact of the various parameters of `AdaProp` on classification accuracy. All experiments were conducted using the WEKA Experimenter (version 3.7) and measured the 10×10-fold cross-validated classification accuracy. The experiments consisted of running `AdaProp` over multiple configurations, varying the following parameters:

Propositionalisation: count-based or summary-based (Section 3.3)

Candidate Generation: mean, median, range or discretized (Section 3.2.3)

Hyperplane Evaluation: CE, RMSE or IG (Section 3.2.4)

Leaf Node Selection: breadth first search or best first search (Section 3.2.2)

Base Learner: `RandomForest` (with 100 trees) or `LogitBoost` (with 50 iterations)

4.1 Datasets

The experiments in this project were ran over twelve multi-instance datasets, consisting of six chemical datasets and six image classification datasets:

- `atoms`, `bonds` and `chains` are mutagenesis datasets, where the aim is to predict the mutagenicity of molecules. The datasets were prepared by Reutemann, Pfahringer, and Frank (2005) from the original mutagenicity problem (Srinivasan, Muggleton, King, & Sternberg, 1994). In all datasets, each bag corresponds to a molecule. Therefore, the datasets differ only in the representation of each molecule, where each instance corresponds to an atom, a bond or a chain (a pair of bonds) respectively.

- `musk1` and `musk2` are datasets which arise from the original multi-instance application (Dietterich et al., 1997), where the goal is to predict whether the molecules emit an odour. Each molecule is described as a bag of the possible conformations (i.e. shapes) of the molecule. The `musk1` dataset is simply a subset of the `musk2` dataset.

- In `trx`, the aim is to identify proteins which belong to the Thioredoxin-fold family (Wang et al., 2004). In this dataset, approximately 87% of the bags belong to the positive class, thus the majority class classifier (i.e. `ZeroR` in WEKA) obtains 87% accuracy. In our experiments, `AdaProp` often struggles to improve on the performance of `ZeroR` on this dataset.

- `tiger`, `fox` and `elephant` are image classification datasets prepared by Andrews et al. (2002) from the COREL dataset, where the aim is to detect whether the objects of interest (tigers, foxes and elephants, respectively) appear in each image. Each bag corresponds to an image and the instances of each bag are produced by segmentation. Each segment is described by colour histograms and texture features.

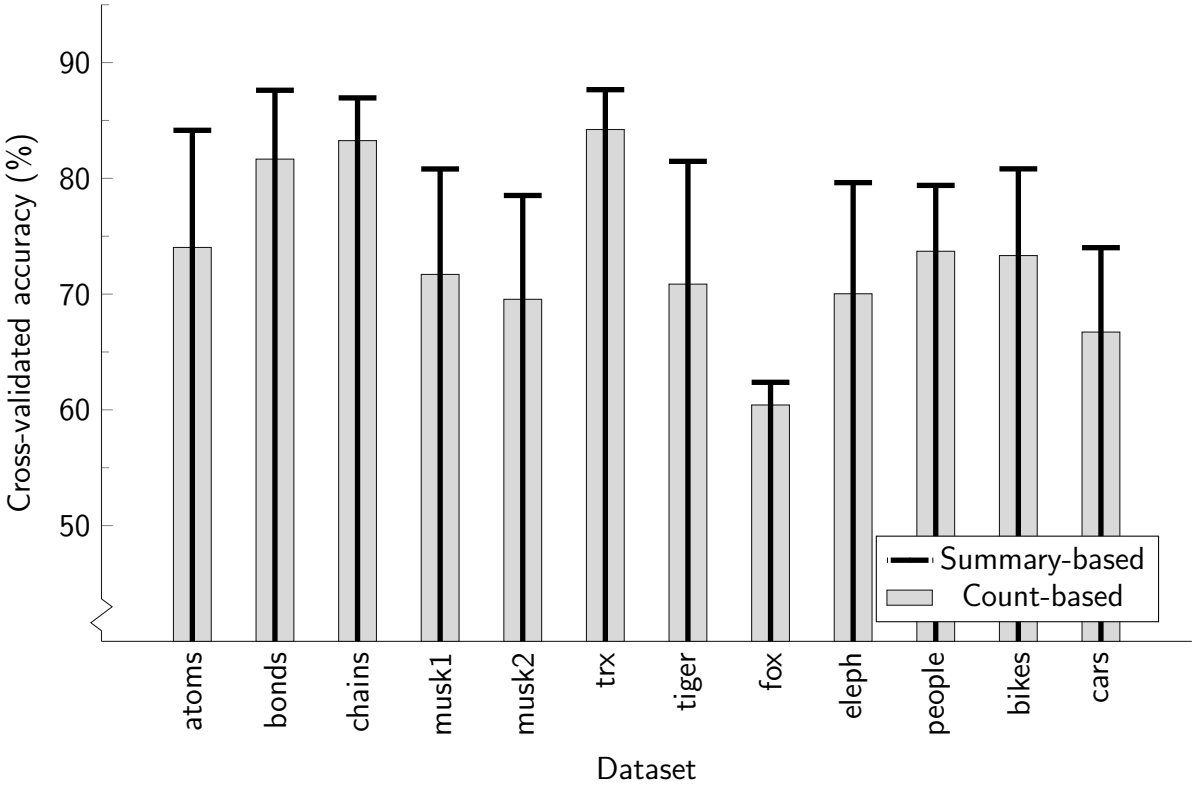
- `people`, `bikes` and `cars` are image classification datasets prepared by Mayo (2007), from the images in GRAZ02 (Opelt, Pinz, Fussenegger, & Auer, 2006), where the objects of interest are people, bikes and cars respectively. In these datasets, each image is described by subdividing the image into regions and extracting local binary patterns and colour histograms from each subdivision.

4.2 Propositionalisation

The algorithm developed in this project, `AdaProp`, defines a partitioning of the instance space into multiple overlapping regions and uses this partitioning to propositionalise each bag. In our experiments, we consider two propositionalisation approaches: count-based and summary-based. Count-based propositionalisation is identical to the propositionalisation method proposed by Weidmann et al. (2003) and involves simply counting the number of instances of each bag which fall into each region (see Section 3.3.1). Summary-based propositionalisation is inspired by the RELAGGS algorithm (Krogel and Wrobel (2003)) and involves computing summary statistics (such as average, minimum and maximum) over all attributes for the subset of instances in each region (see Section 3.3.2).

Summary-based propositionalisation is computationally expensive, therefore only relatively small `AdaProp` trees were practical for our experiments, especially as 10×10-fold

Figure 4.1: Average accuracy (for small trees) by propositionalisation method



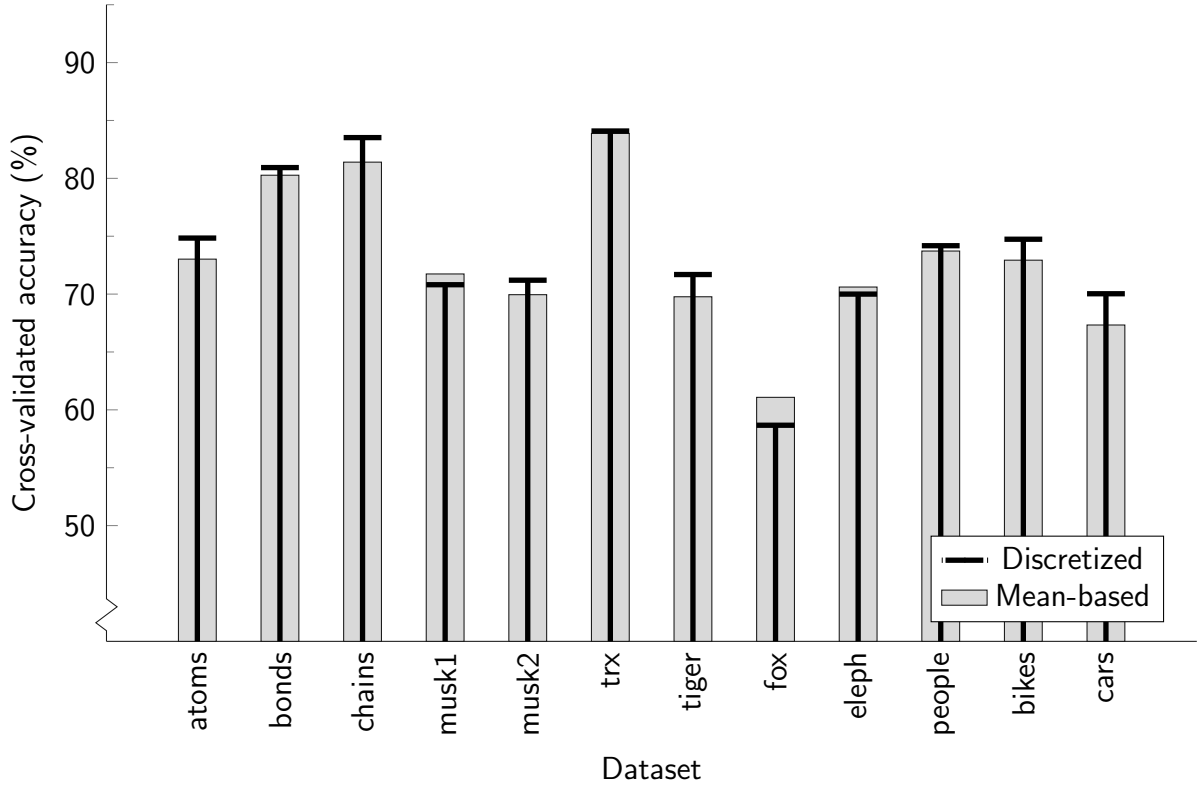
cross-validation was employed. Therefore, in this section (when comparing the two propositionalisation approaches), we only consider trees which have 7 or fewer nodes.

Figure[1] 4.1 shows the average accuracy over all parameter settings stated at the start of this chapter and all tree sizes from 1 to 7. It is clear that summary-based propositionalisation performs significantly better than count-based propositionalisation. This is the expected result, as the summary-based propositionalisation preserves much more information about the instances in each region, i.e. describes the instances in more detail, than count-based propositionalisation. The results show a similar relationship when the results are grouped by base learner (Figures A.1 and A.2, in Appendix A) and when grouped by hyperplane evaluation method (Figures A.3 and A.4, in Appendix A).

In fact, even when larger trees with up to 15 nodes per tree are used for count-based propositionalisation, the accuracy of summary-based propositionalisation with small trees remains higher than the accuracy of count-based propositionalisation. This suggests that summary-based propositionalisation performs uniformly better than count-based propositionalisation regardless of the choice of base learner or evaluation method, and should be used if running time is unimportant.

[1]For each figure, there is a corresponding table (showing the numeric values) in Appendix B.

Figure 4.2: Average accuracy by candidate generation method



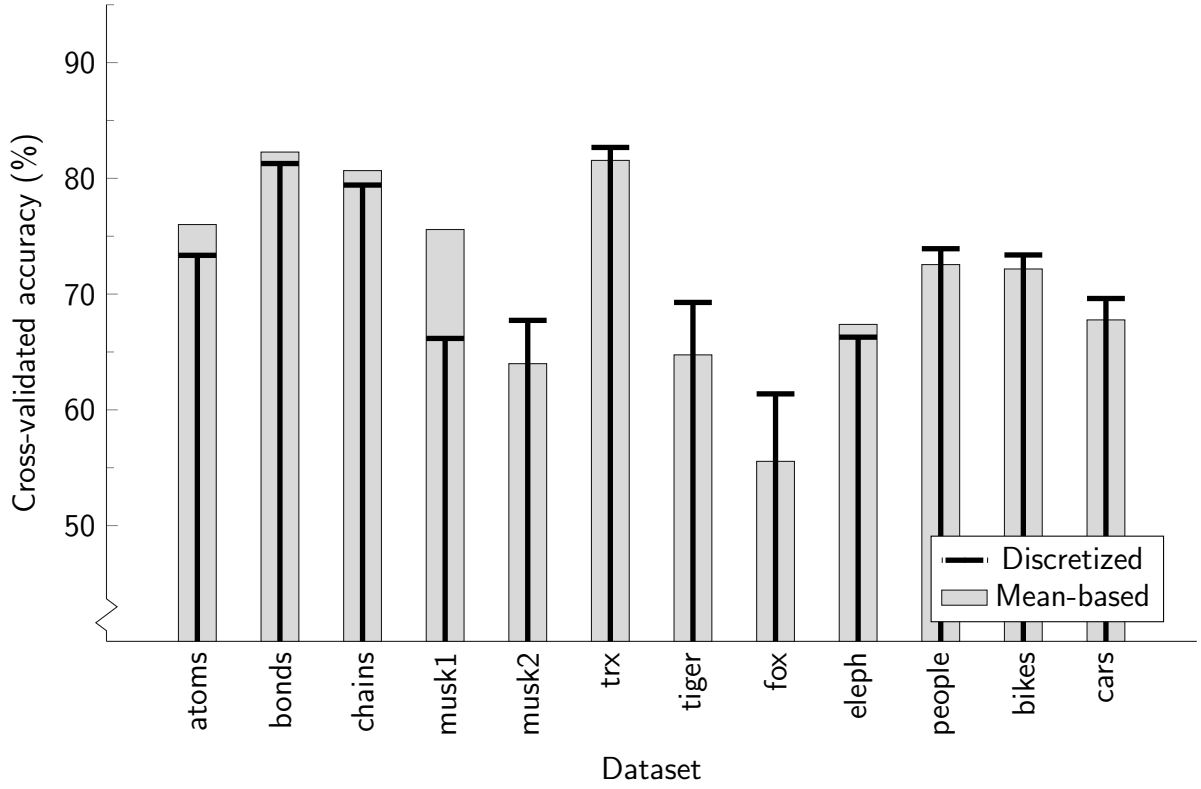
4.3 Candidate partition generation

At each iteration of `AdaProp`, a set of candidate partitioning hyperplanes are generated. We originally considered three methods for generating candidate partitions: mean-based, median-based and range-based (see Section 3.2.3). However, in practice, the three methods generated similar candidate hyperplanes, and the experiments indicated that all three methods perform very similarly for most datasets.

From these three methods, the mean-based one was chosen as the default. This method is expected to perform better than range-based splitting in datasets which have significantly skewed distributions of attribute values, as the range-based method ignores the value of all instances except those at the extremes. Also, the median based approach is computationally more expensive than both the mean-based method and the range-based method. Therefore, the mean-based candidate generation technique is the only method considered in our remaining experiments.

Additionally, we considered a candidate generation technique which differs significantly from the above methods: the discretization-based method (see Section 3.2.3). In the discretization-based method, each class boundary is evaluated as a potential location for

Figure 4.3: Average accuracy (for `randomforest`) by candidate generation method

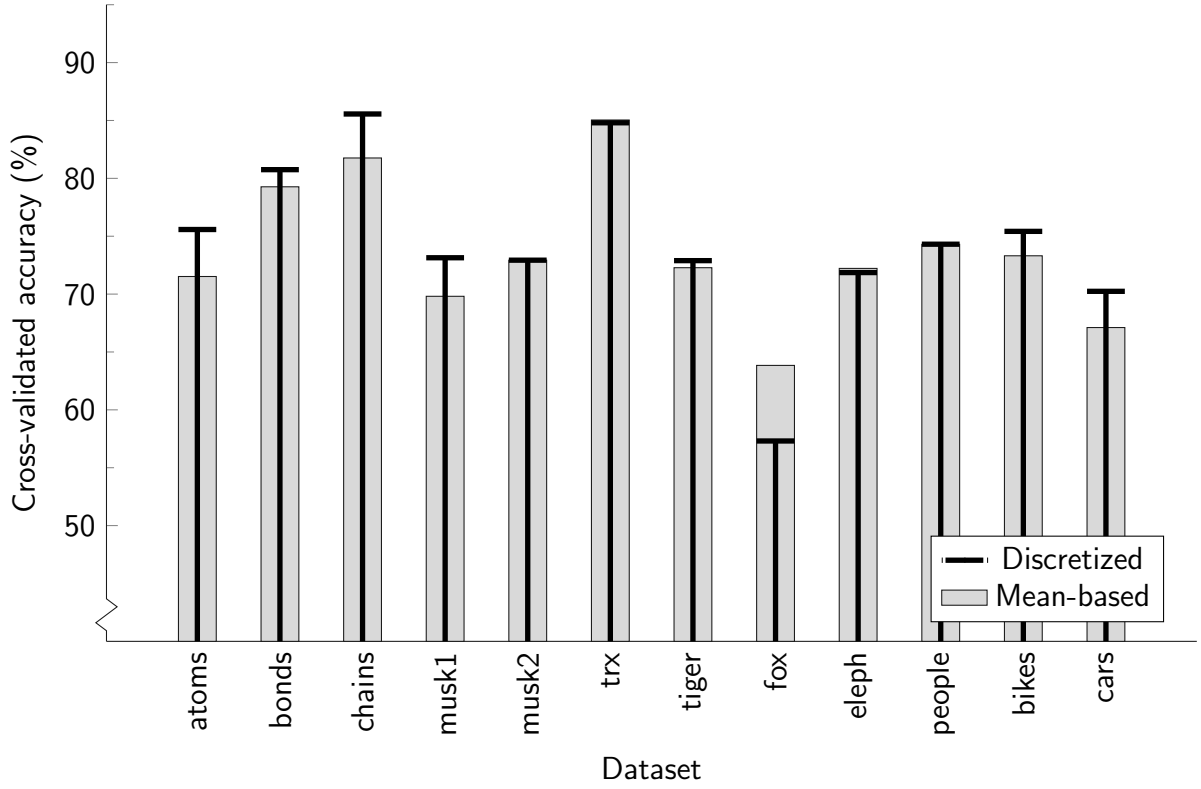


partitioning hyperplanes. Figure 4.2 compares the performance[2] of the discretization-based method against the mean based method. It can be seen that the discretization-based method performs slightly better than the mean-based method on most datasets. However, the difference does not appear to warrant the additional computational effort required.

In contrast, when the results are grouped by base learner (Figures 4.3 and 4.4), a clear pattern emerges. In the datasets `atoms`, `bonds`, `chains` and `musk1`, the discretization-based method performs significantly better than the mean-based method for experiments using the `LogitBoost` base learner, while the mean-based performs better than the discretization-based method in the experiments using the `RandomForest` base learner. Similarly, in the datasets `musk2`, `tiger`, `fox` and `people`, discretization performs significantly better for `RandomForest`, while there is little difference in classification accuracy when using `LogitBoost`. This clearly indicates that the choice of candidate generation method and choice of base learner are interdependent. A consequence of this interdependence is that any parameter optimisation of `AdaProp` over base learners and candidate generation methods must be performed simultaneously.

[2]For each figure, the corresponding table in Appendix B lists the parameters configurations used in the figure.

Figure 4.4: Average accuracy (for `logitboost`) by candidate generation method



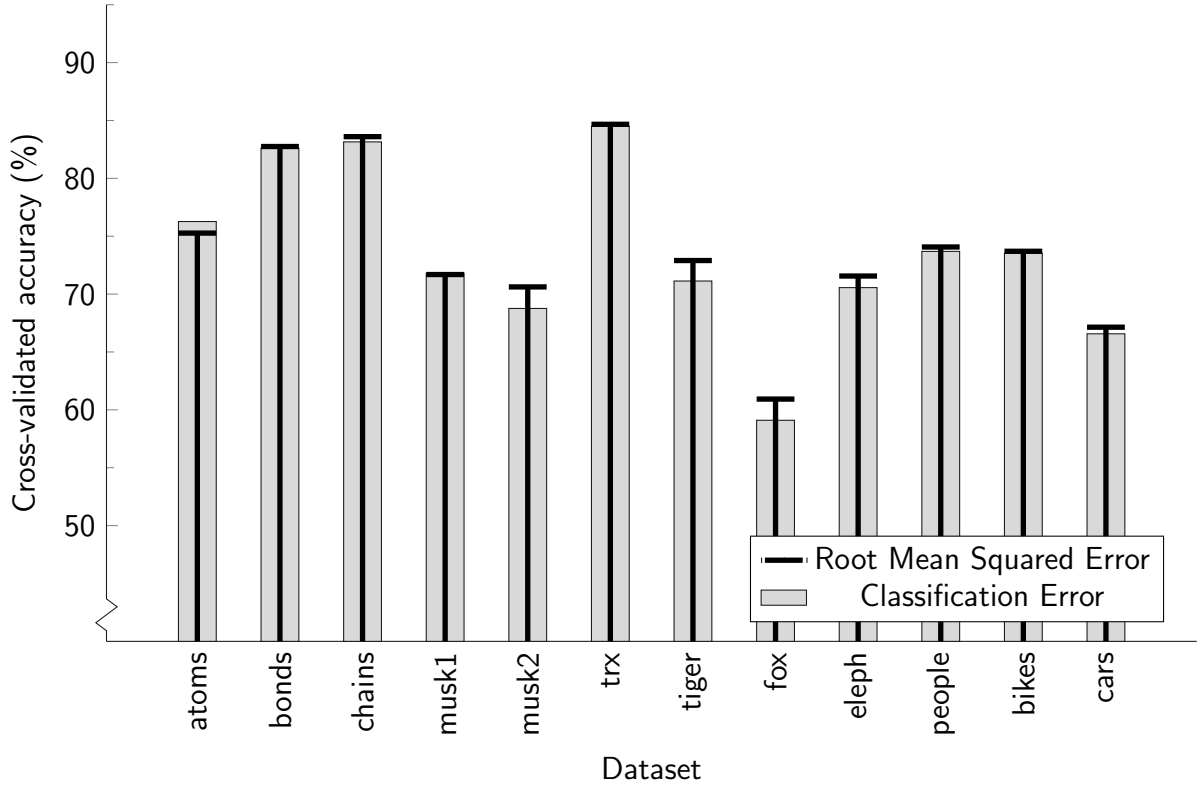
4.4 Hyperplane evaluation

From a set of candidate hyperplanes, `AdaProp` aims to find the optimal hyperplane. The original measure used to evaluate each hyperplane was the classification error metric (abbreviated CE). The CE metric uses the class predictions made by the base learners to compute the error on each prediction. In essence, any bag which is misclassified contributes 1 to the total classification error, while any correctly classified bag contributes no error.

However, CE ignores the probabilities produced by the base learners, which give a better indication of how well the base learner has fit to the dataset, and the confidence of each prediction. In order to make use of these probabilities, we also examined the root mean squared error (abbreviated RMSE) to evaluate each hyperplane. For each bag, RMSE computes the error as the square of the difference between the predicted class probabilities and the actual class value (See Section 3.2.4 for more detail).

Figure 4.5 compares the average classification accuracy over all experiments when using the CE metric and the RMSE metric. Overall, results indicate that RMSE performs comparably to CE on most datasets, but in the `musk2`, `tiger`, `fox` and `elephant`

Figure 4.5: Average accuracy by evaluation method

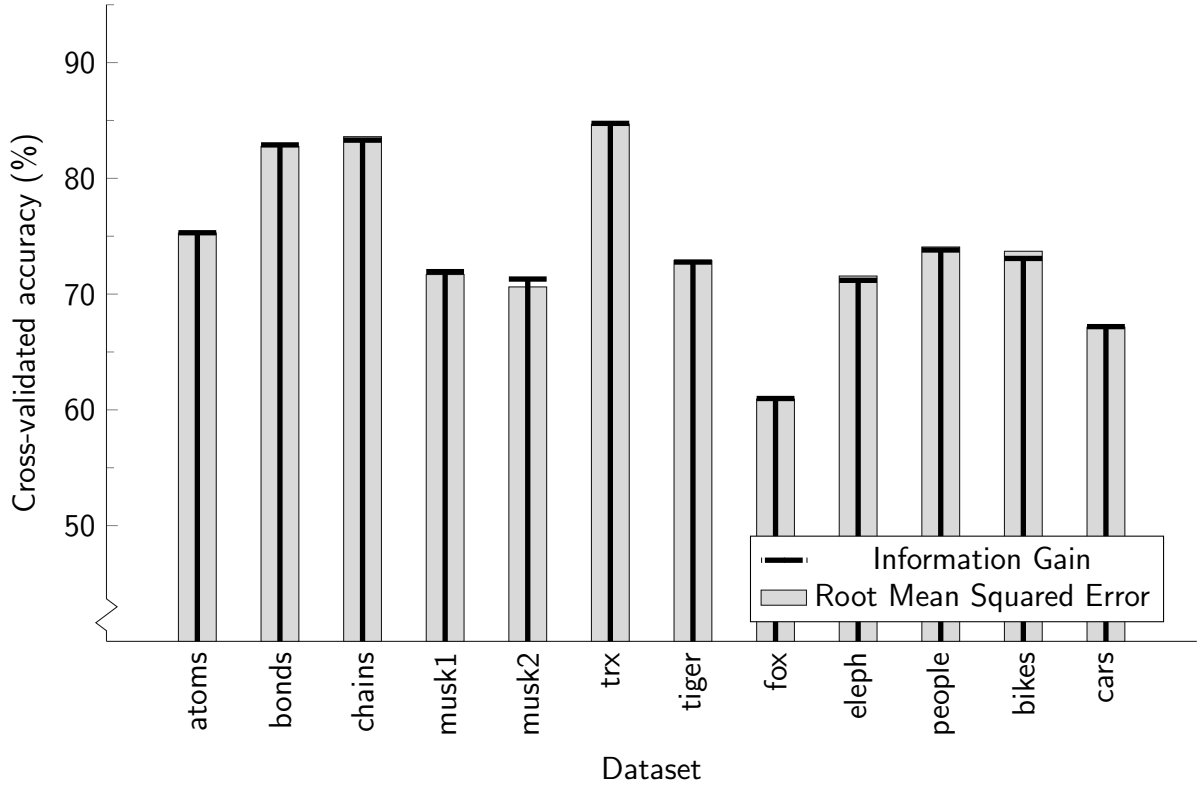


datasets, it performs somewhat better. When the results are grouped by base learner (Figures A.5 and A.6, in Appendix A), a very similar pattern can be seen, although the difference is more pronounced when using the `LogitBoost` base learner.

Another evaluation approach, using the Information Gain (abbreviated IG) was also considered. IG is similar to RMSE since both methods make use of the probabilities emitted by the base learner. Our experiments (Figure 4.6) indicate that RMSE and IG perform very similarly as well. In some datasets, e.g. `musk2`, IG appears to perform slightly better, however this difference is not significant and the pattern does not remain when the results are grouped by the base learner. In fact, in almost all individual experiment results, RMSE and IG obtained very similar classification accuracies.

Therefore, RMSE was chosen as the default metric for evaluating hyperplanes in all further experiments, as it performs better than CE and comparably to IG.

Figure 4.6: Average accuracy by evaluation method



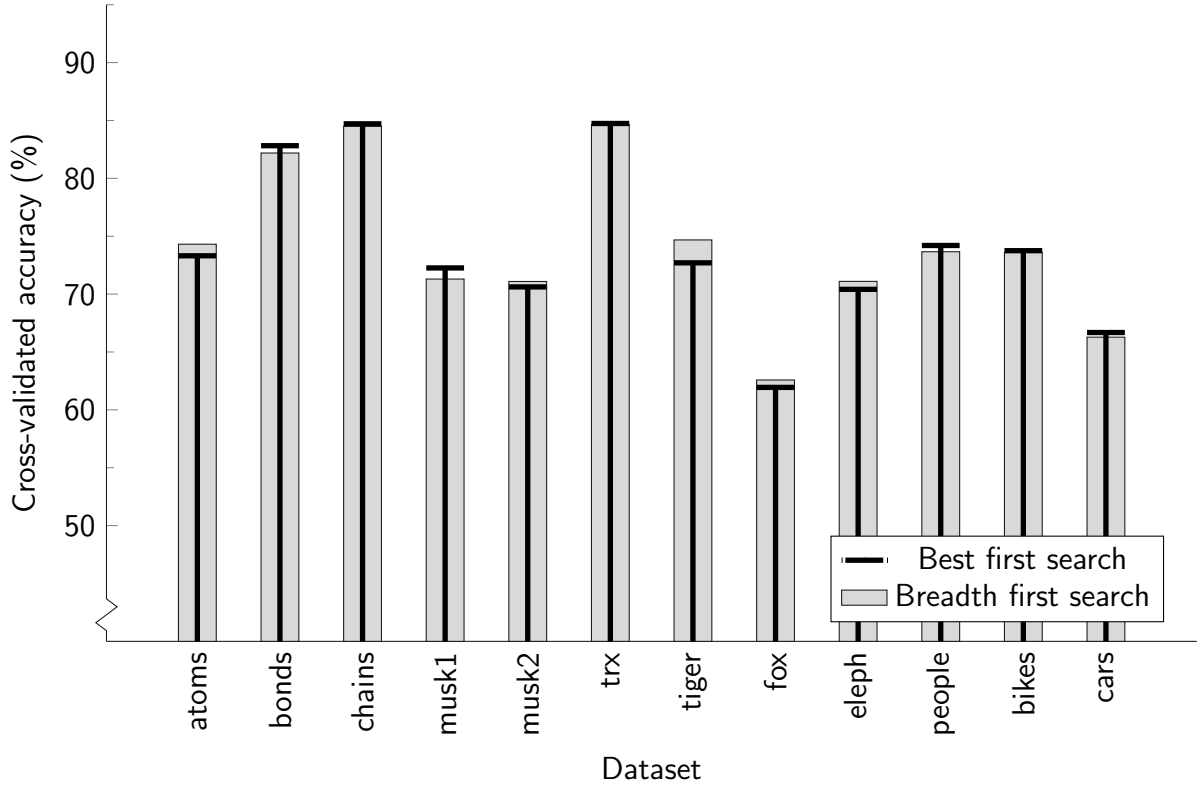
4.5 Leaf node selection

The `AdaProp` algorithm builds a tree of partitions during the training phase. As discussed in Section 3.2.2, we considered two search approaches for building this tree: breadth first search and best first search. When using the breadth first search, nodes are expanded in order of depth, i.e. the generated tree of partitions will be a complete binary tree.[3] In contrast, best first search strategy allows nodes to be expanded in any order, thus requires searching over a much larger search space than breadth first search.

Figure 4.7 shows the classification accuracy over all experiments, grouped by the leaf node selection strategy. The results indicate that using best first search does not improve the classification accuracy significantly in any of the datasets. When the results are grouped by base learner (Figures A.7 and A.8, in Appendix A), the pattern is very similar, with breadth first and best first search performing very similarly. This suggests that there is no benefit to using best first search and therefore we have selected breadth first search as the default method in `AdaProp`.

[3]A binary tree is complete if every level, except the last (i.e. deepest) level, is completely filled.

Figure 4.7: Accuracy by leaf node selection strategy



4.6 Base learners

`AdaProp` is a meta learner, thus requires a base learner in order to perform the adaptive propositionalisation. Our early, exploratory experiments indicated that among the base learners in WEKA, `LogitBoost` (with 50 boosting iterations) and `RandomForest` (with 100 trees) were consistently the best performers. Therefore, our final experiments only use these base learners.

Figure 4.8 shows the average accuracy over all experiments grouped by base learner. The results suggest that each base learner is suited to different datasets, with often very noticeable differences in performance. However, neither base learner outperforms the other consistently, i.e. over all datasets.

For example, in the mutagenesis datasets (`atoms`, `bonds`, `chains`) and in `musk1`, the `RandomForest` base learner performs better than the `LogitBoost` base learner. However, for the image classification datasets (`tiger`, `fox`, `elephant`, `people`, `bikes`, `cars`) and `musk2`, `LogitBoost` performs better than `RandomForest`. This pattern holds consistently when the results are broken down by hyperplane evaluation method and leaf node selection strategy, as shown by Table 4.1.

Figure 4.8: Average accuracy by base learner

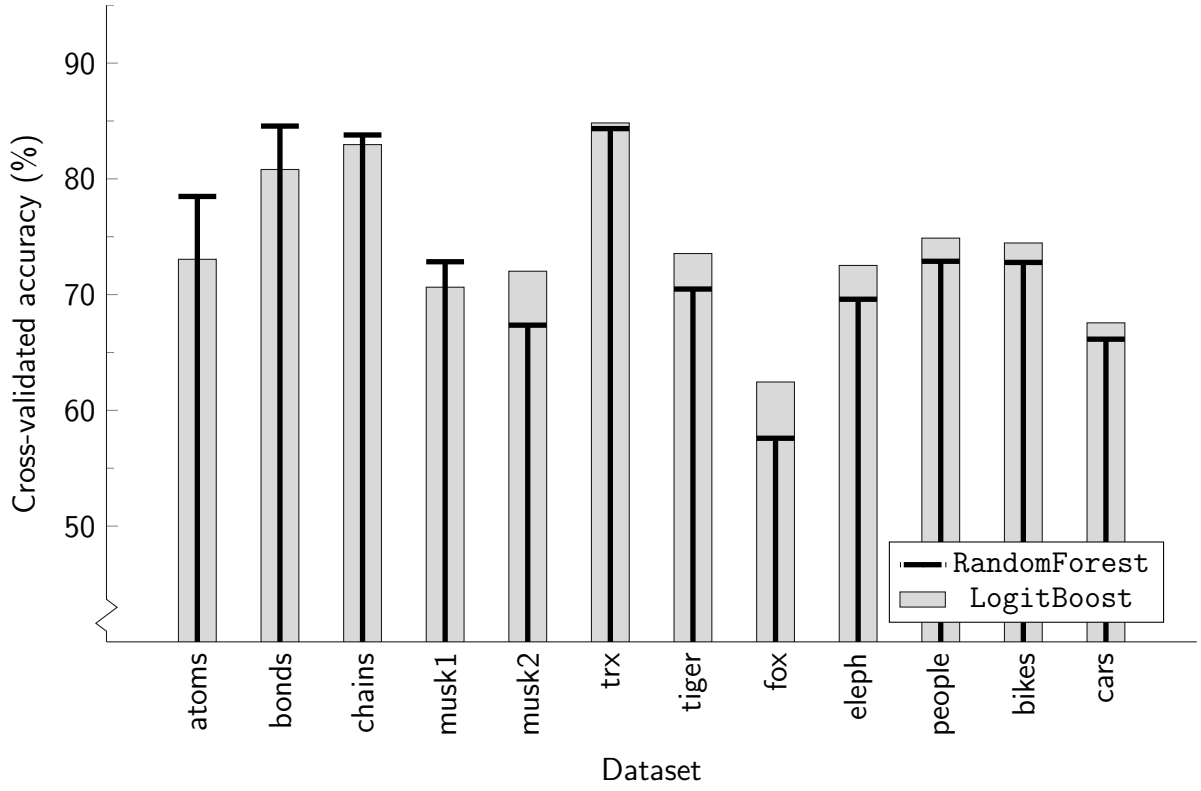


Table 4.1: The best performing base learner by dataset and parameter configuration

Dataset	# of wins		Winner by configuration			
	RF	LB	CE		RMSE	
			BR	BE	BR	BE
`atoms`	4	0	RF	RF	RF	RF
`bonds`	4	0	RF	RF	RF	RF
`chains`	4	0	RF	RF	RF	RF
`musk1`	2	2	RF	LB	LB	RF
`musk2`	0	4	LB	LB	LB	LB
`trx`	3	1	RF	RF	RF	LB
`tiger`	0	4	LB	LB	LB	LB
`fox`	0	4	LB	LB	LB	LB
`elephant`	0	4	LB	LB	LB	LB
`people`	0	4	LB	LB	LB	LB
`bikes`	0	4	LB	LB	LB	LB
`cars`	0	4	LB	LB	LB	LB

Key:

RF:	`RandomForest`	CE:	Classification error	BR:	Breadth first search
LB:	`LogitBoost`	RMSE:	Root mean squared error	BE:	Best first search

Chapter 5

Results: Refinements

Given the results of the previous chapter (Chapter 4), we hypothesized that `AdaProp` is overfitting noticeably to some datasets. In order to reduce the extent of overfitting, and therefore improve the classification accuracy, two standard refinement techniques were examined: parameter selection of the maximum tree size, and bagging with and without randomization.

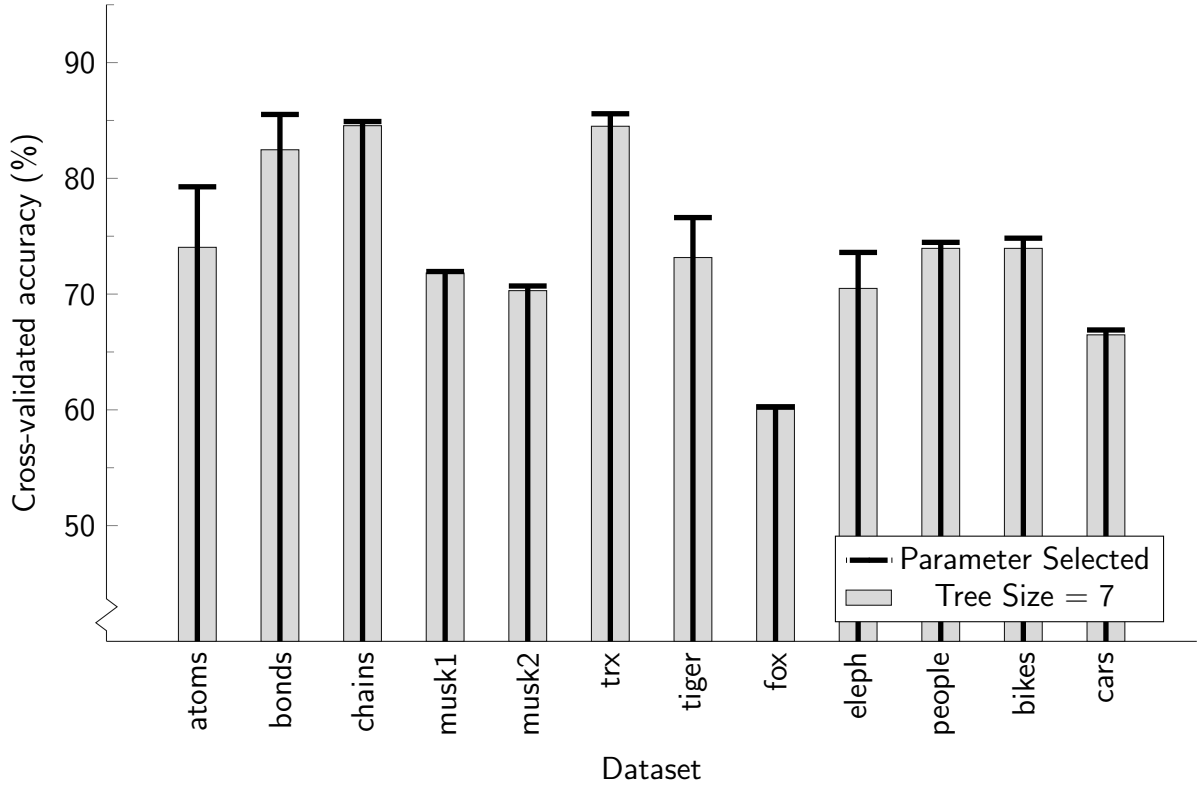
The refinement techniques, in theory, can be applied independently of the choice of the other `AdaProp` parameters, such as the propositionalisation method. However, the experiments presented in Section 4.2 indicate that summary-based propositionalisation performs significantly better than count-based propositionalisation, which suggests that overfitting is more prevalent when using count-based propositionalisation. In fact, early exploratory experiments with bagging and parameter selection showed that summary-based propositionalisation obtained very little increase in accuracy when the refinements were applied.

Therefore, we restrict our investigation of overfitting and the impact of the refinements to count-based propositionalisation only. Thus, all experiments, charts and discussions in this chapter consider only the results derived via count-based propositionalisation.

5.1 Parameter selection

A parameter of the `AdaProp` algorithm is the size of the tree of partitioning hyperplanes. Early experiments indicated that this parameter had some impact on the performance of the algorithm, where, in general, the cross-validated accuracy increased as the tree size increased, up to some optimum tree size, after which the accuracy decreased. The

Figure 5.1: Average accuracy - Impact of parameter selection

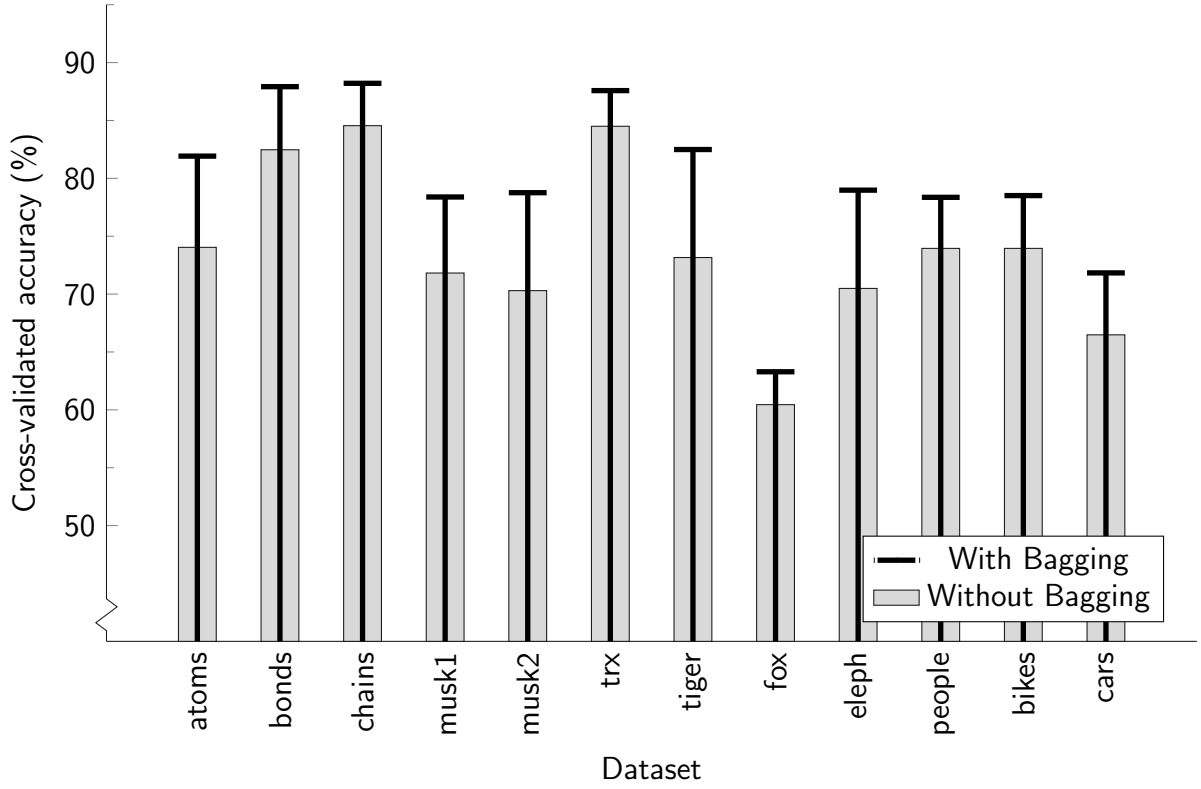


early experiments also indicated that this optimum size is heavily dependent on the dataset being learned. In order to determine an appropriate size automatically for any given dataset, 5-fold cross-validated parameter selection was implemented, as discussed in Section 3.5.1.

Figure 5.1 shows cross-validated accuracy averaged over all count-based experiments when using parameter selection to determine the tree size. This figure also compares the parameter selected accuracy to the accuracy obtained when the tree size is set to 7 across all datasets, which was the best constant limit: when `AdaProp` was run with a constant maximum tree size parameter across all datasets, the experiments where the tree size parameter was set to 7 obtained the best average accuracy.

Figure 5.1 clearly indicates that parameter selection does not produce lower classification accuracies than applying a constant tree size. In fact, in datasets such as `atoms`, `bonds`, `tiger` and `elephant`, parameter selection obtains a noticeable improvement in classification accuracy. The parameter selection process chose very small trees in the cases of `atoms` and `bonds` (sizes 3 and 5 respectively), while choosing larger trees for the `tiger` and `elephant` datasets. In the remaining datasets, the chosen tree size was close to 7, therefore suggesting that for these datasets, a tree size of 7 is near optimal.

Figure 5.2: Average accuracy - Impact of bagging



5.2 Randomized bagging

A standard solution to reduce the amount of overfitting is to perform bagging, as discussed in Section 3.5.2, where n new datasets are generated by sampling with replacement from the original dataset and n `AdaProp` trees are generated independently, one for each dataset. The final model combines the predictions of the n models by averaging class probability estimates. In the experiments, we performed bagging with 50 iterations (i.e. $n = 50$), with RMSE evaluation, mean-based hyperplane generation, count-based propositionalisation, breadth-first search, both base learners, and with no parameter selection.

Figure 5.2 shows the impact of bagging on the classification accuracy. It is clear that bagging improves performance significantly across all datasets, with the largest improvement (of 9%) occurring in `tiger` and the smallest improvement (of 3%) occurring in `fox`. In fact, the average improvement with bagging over all datasets is 5.8%. This strongly suggests that count-based `AdaProp`, without bagging, overfits every dataset: its classifications depend too much on the particular training set used.

When performing bagging, the runtime is linear with respect to the number of iterations, therefore the 50-iteration bagged `AdaProp` experiments require 50 times longer to run than the single `AdaProp` tree experiments. Due to this runtime cost, it was not practical to run standard bagging over larger trees, and therefore the standard bagging experiments were limited to a maximum tree size of 15. In order to allow experiments on larger trees to be conducted, and also in order to reduce overfitting further, randomization of attribute selection was implemented, where only a subset of attributes was considered when generating candidate hyperplanes (see discussion in Section 3.5.2).

First, we examine the impact of randomization on the bagging process, while keeping the size of the trees constant, limited to a maximum tree size of 15, and the number of trees at 50. Since the same number of equally sized trees are used and randomization reduces the amount of information used to make each decision, the classifier will fit the dataset less closely. Figure 5.3 compares the average cross validated accuracy for standard bagging and randomized bagging. Decreases in cross-validated accuracy can be seen for almost all datasets, especially in the image classification datasets such as `tiger` and `elephant`, where a significant decrease in accuracy is evident. Therefore, it appears that randomized bagging underfits most datasets. However, somewhat surprisingly, `atoms` and `musk1` show slight improvements when using randomization, which suggests that perhaps even `AdaProp` with standard bagging overfits these datasets.

The main advantage of randomization is that it allows much larger trees to be used in the bagging experiments. With randomized bagging, experiments involving `AdaProp` trees of size up to 30 became practical. In order to compare standard bagging (performed only over small trees) against randomized bagging (performed over larger trees), we plot the maximum (instead of the average) accuracy achieved for each dataset across all parameter settings. This maximum accuracy can be used as an indicator of the best case performance of each method.

Figure 5.4 compares the maximum[1] accuracy with and without randomization. For the image classification datasets, the performance of randomized bagging still lags behind that of standard bagging and the difference is significant in `tiger`, `elephant`, and `bikes`, despite the larger trees used by randomized bagging. However, for the chemical datasets, randomized bagging performs at least as well as standard bagging and shows clear improvement in `atoms` and `musk1`. This shows that randomization is able to improve classification accuracy of bagged `AdaProp` for some datasets, but can also result in significant decreases in accuracy over other datasets.

[1] Average accuracy is less meaningful in this case as the set of experiments differs for each series.

Figure 5.3: Average bagged accuracy - Impact of randomization (for small trees)

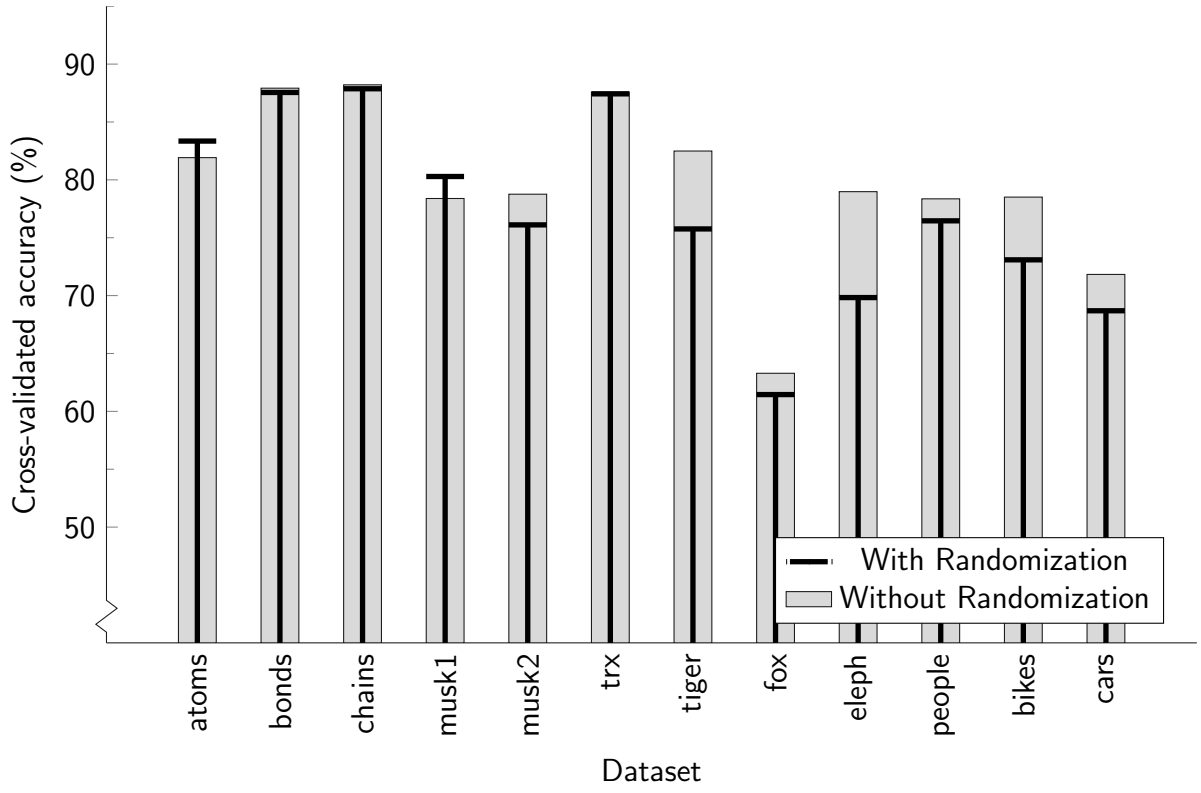
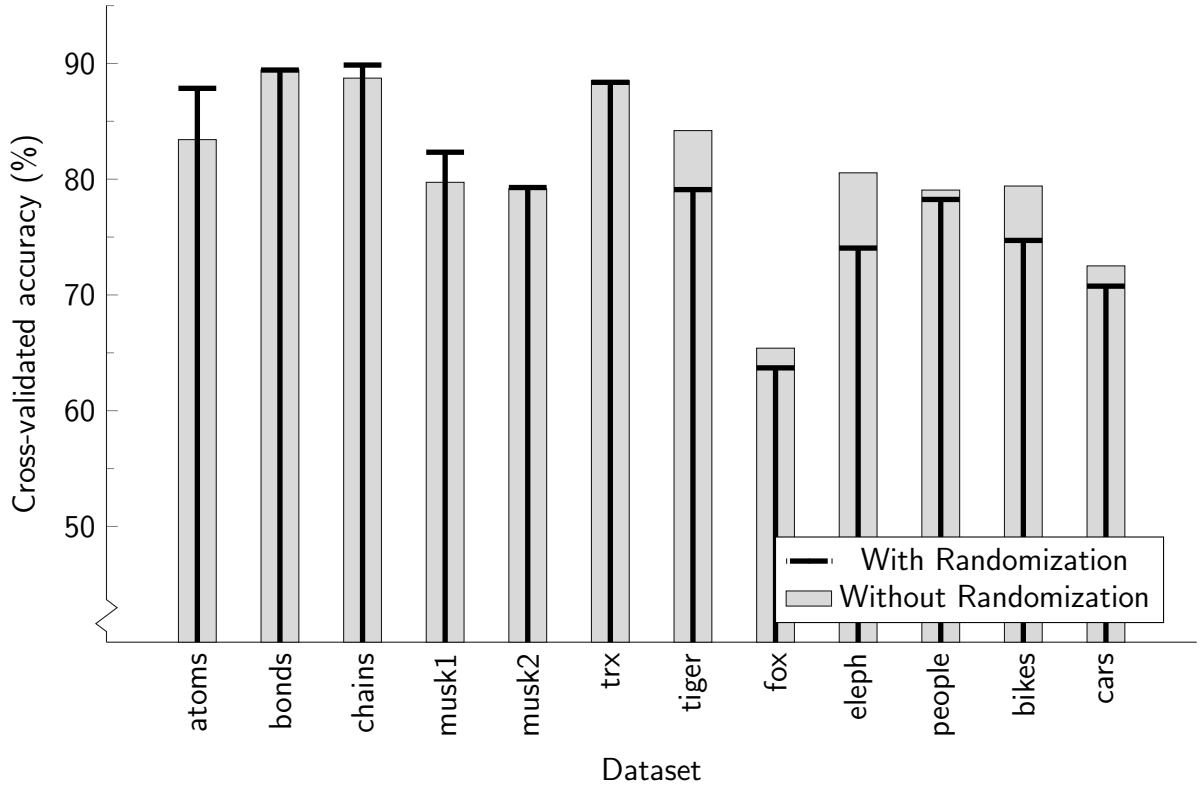


Figure 5.4: Maximum bagged accuracy - Impact of randomization



Chapter 6

Results: Comparisons

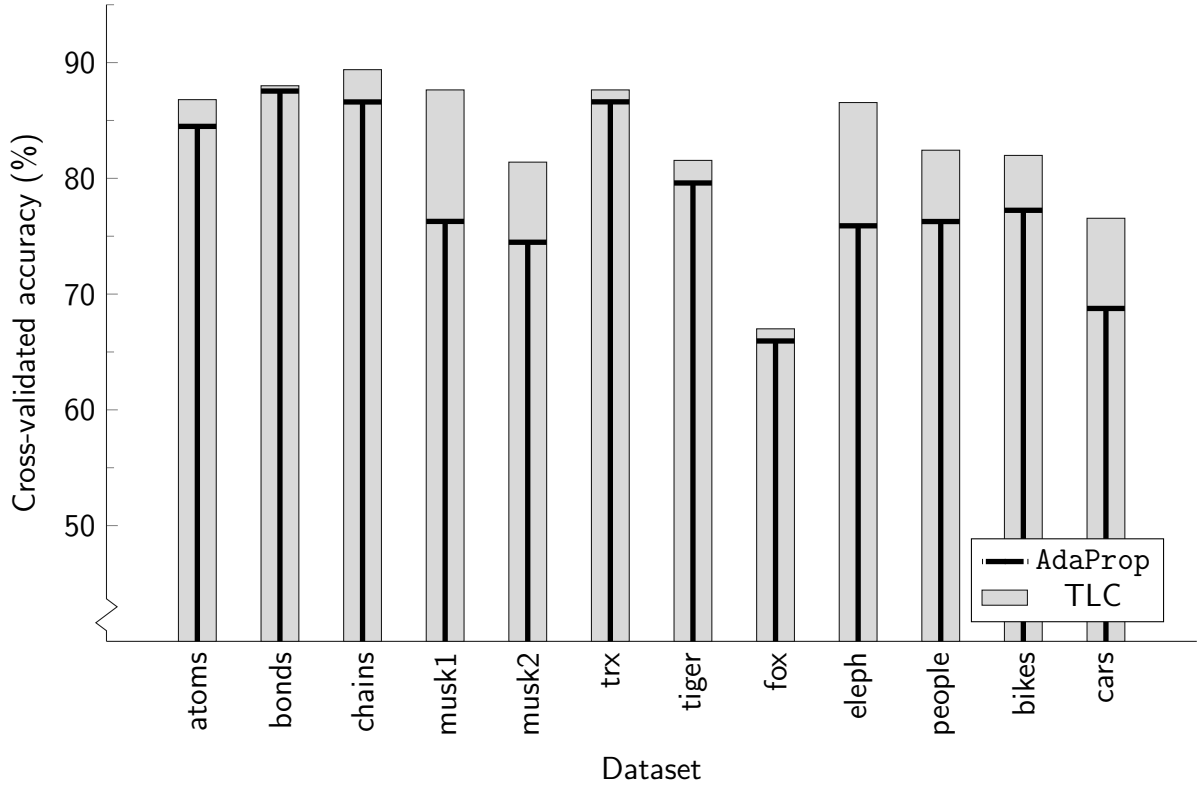
The experiments conducted in Chapters 4 and 5 were limited to the `AdaProp` algorithm, as they were aimed at comparing the various choices for each parameter of `AdaProp` and the impact of each choice on the classification accuracy. However, many other multi-instance machine learning algorithms have been proposed in the literature (see Section 2.4). Therefore, in this chapter, we compare the classification accuracy achieved by `AdaProp` against that of other multi-instance algorithms implemented in WEKA. Among these algorithms, `AdaProp` is closely related to TLC (Weidmann et al., 2003) and RELAGGS (Krogel & Wrobel, 2003), therefore we compare it against these two algorithms in more detail.

6.1 Count-based `AdaProp` vs. TLC

TLC, which was discussed in Section 2.4.1, is an algorithm introduced by Weidmann et al. (2003) which uses a two level learning approach to handling multi-instance data. The aim of the two level approach is to separate the inference of the instance labels from the learning of the bag labels. TLC builds up a tree of partitions at the first level, which is then used to propositionalise each bag at the second level. The count-based propositionalisation approach of `AdaProp` is identical to TLC's second level, therefore count-based `AdaProp` and TLC only differ at the first level, i.e. how the tree of partitions is built up. Therefore, comparing TLC to count-based `AdaProp` is equivalent to comparing the tree generation method of TLC against that of `AdaProp`.

In the experiments, TLC was run with the same base learners as that of `AdaProp`, i.e. `RandomForest` with 100 trees and `LogitBoost` with 50 iterations. Figure 6.1 shows the best cross-validated accuracy that TLC was able to achieve in each dataset, plotted

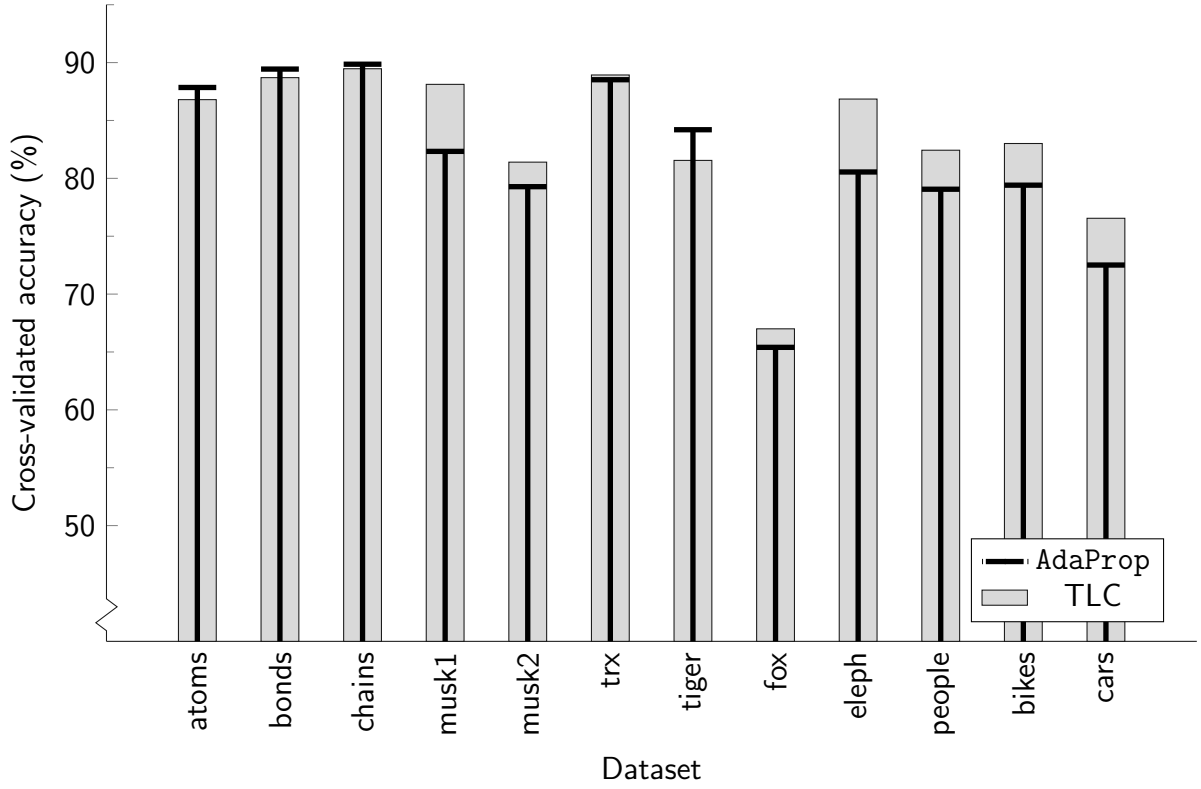
Figure 6.1: Maximum accuracy - TLC vs count-based AdaProp



against that of the best count-based AdaProp configuration (i.e. without bagging). TLC outperforms AdaProp in all datasets, and by a large margin in musk1, musk2 and the last four image classification datasets (elephant to cars). These results clearly indicate that TLC's tree building approach results in better classification accuracies than that of count-based AdaProp.

Experiments in Chapters 4 and 5 indicated that count-based AdaProp significantly overfits some datasets. In order to determine whether this overfitting was responsible for the poor performance of count-based AdaProp, experiments were conducted for TLC with 50 iterations of bagging. Figure 6.2 compares the maximum accuracy obtained by bagged TLC against the maximum accuracy of bagged count-based AdaProp (also with 50 iterations). The figure shows that bagged TLC and bagged AdaProp perform similarly across the mutagensis datasets (atoms, bonds, chains) and trx, while bagged AdaProp performs noticeably better in tiger. However, even with bagging applied, TLC performs better than count-based AdaProp in musk1, musk2 and the last four image classification datasets, but the difference in classification accuracy is somewhat smaller than that in Figure 6.1. This shows that overfitting is more prevalent in AdaProp than in TLC, but even with bagging applied, TLC outperforms count-based AdaProp.

Figure 6.2: Maximum bagged accuracy - TLC vs count-based `AdaProp`

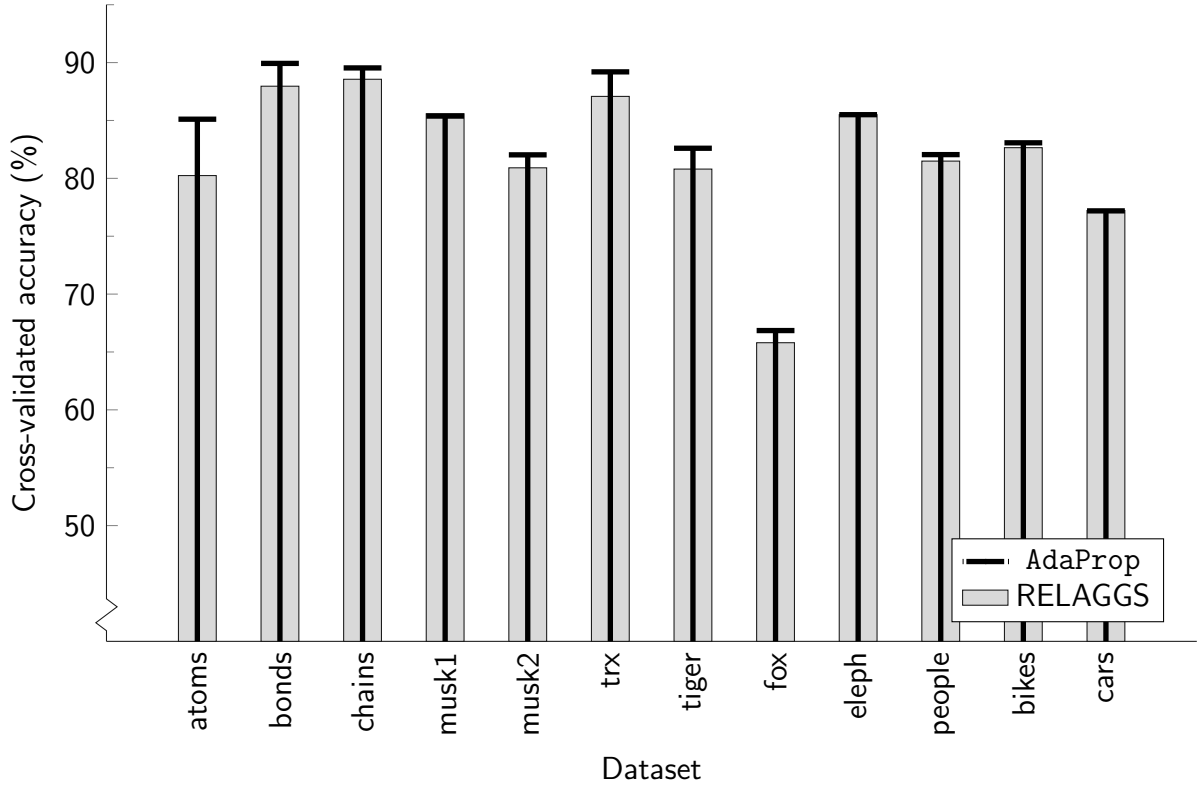


Further investigation of the experiment results showed that TLC produces much larger partitioning trees than `AdaProp`. All count-based `AdaProp` trees, across all datasets, were limited to 30 nodes (by design). However, Table 6.1 shows that the trees produced by TLC are significantly larger, especially in `trx` and the GRAZ02 datasets (`people`, `bikes` and `cars`). This suggests that more experiments involving larger `AdaProp` trees should be conducted in the future, to determine if significantly increasing the tree size can contribute towards increasing the cross-validated accuracy. However, such very large trees are impractical as `AdaProp` is computationally much more expensive than TLC.

Table 6.1: Size of trees built by TLC (when run on entire dataset)

Dataset	TLC tree size (# nodes)
atoms	71
bonds	247
chains	317
musk1	37
musk2	205
trx	1315
tiger	161
fox	153
elephant	195
people	711
bikes	769
cars	861

Figure 6.3: Maximum accuracy - RELAGGS vs summary-based `AdaProp`

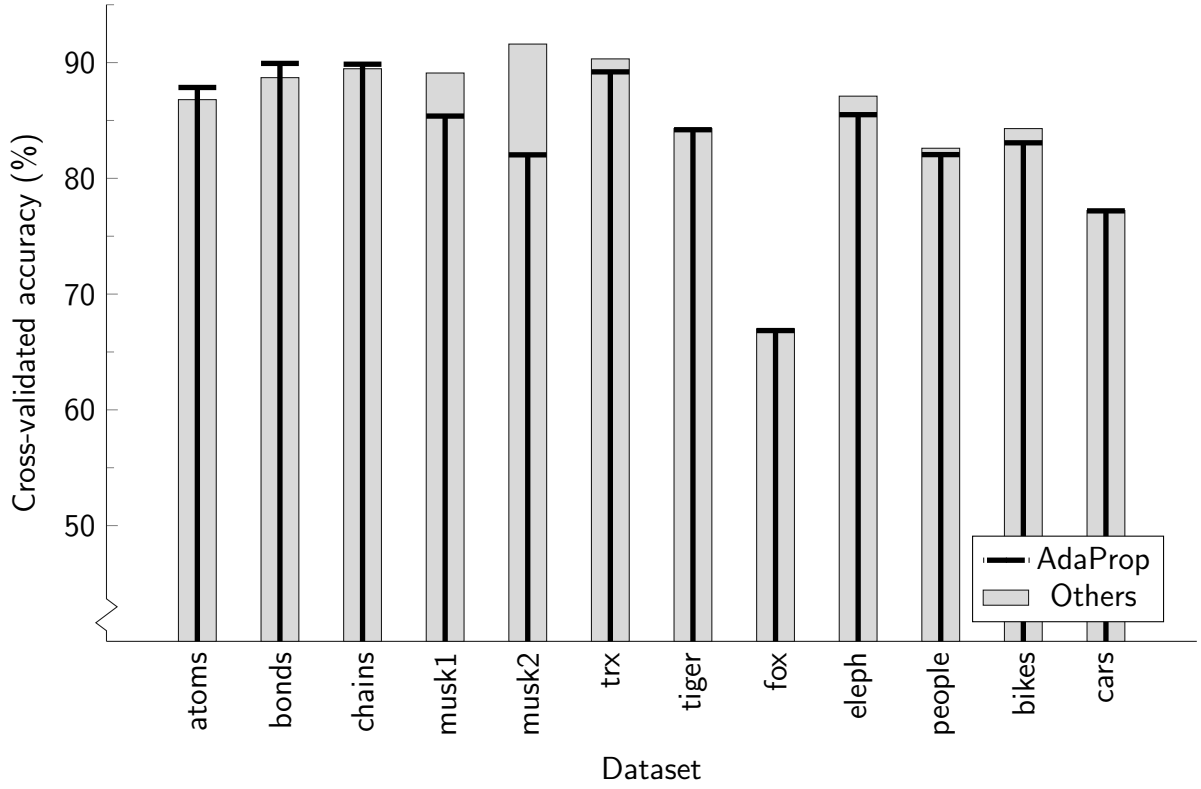


6.2 Summary-based `AdaProp` vs. RELAGGS

RELAGGS is an algorithm introduced by Krogel and Wrobel (2003) which propositionalises each bag by computing the summary statistics for each attribute over all instances in the bag. Summary-based `AdaProp` performs the same propositionalisation, albeit after grouping the instances of each bag by a tree of partitions. Therefore, summary-based `AdaProp` can be considered to be a generalisation of RELAGGS, as summary-based `AdaProp` with a one-node tree produces exactly the same result as RELAGGS.

Similar to the comparison against TLC, RELAGGS was run using both base learners, `RandomForest` and `LogitBoost`. Figure 6.3 compares the best cross-validated accuracy achieved on each dataset by RELAGGS and summary-based `AdaProp`. Summary-based `AdaProp` performs at least as well as RELAGGS in all datasets, while producing noticeable improvements in performance in the `atoms`, `bonds`, `trx` and `tiger` datasets. This shows that the generalisation performed by summary-based `AdaProp`, i.e. dividing the instance space by its tree of partitions before performing the propositionalisation, is worthy of consideration, that is, can produce an improvement in the classification accuracy over RELAGGS.

Figure 6.4: Comparing `AdaProp` against the other MI algorithms



6.3 `AdaProp` vs. existing MI algorithms

As noted in Section 2.4, there are many well known multi-instance machine learning algorithms in the literature. Previous work (Foulds & Frank, 2008) compared various existing multi-instance algorithms implemented in WEKA and determined the maximum cross-validated accuracy obtained for each dataset. Figure 6.4 plots this maximum classification accuracy, updated by our experiments with TLC and RELAGGS. The figure also shows the best cross-validated accuracy that `AdaProp` was able to achieve across all experiments in this project. The results indicate that `AdaProp` achieves slight improvements in performance over the other algorithms in the `atoms`, `bonds`, and `tiger` datasets, while performing noticeably worse in `musk1` and `musk2`. Across the image classification datasets, there is very little difference between `AdaProp` and the best of the other multi-instance algorithms.

Therefore, from the results in this chapter, it can be concluded that count-based `AdaProp` performs worse than TLC, perhaps as the result of the constraint on the size of the tree, while summary-based `AdaProp` improves on the results of RELAGGS. In general, the best of `AdaProp` seems to perform comparably to the best of the other multi-instance algorithms.

Chapter 7

Conclusion

In this project, we propose `AdaProp`, an algorithm which propositionalises multi-instance data using an approach which is influenced by the base learner. `AdaProp` divides up the instance space by building up a tree of partitioning hyperplanes, where each node of the tree is selected by consulting the base learner. More specifically, at each node of the tree, a set of candidate partitioning hyperplanes is generated, from which a single hyperplane is chosen, by evaluating each hyperplane via the base learner. The resultant tree of partitioning hyperplanes is then used to propositionalise each bag, by either counting the number of instances of or computing the summary statistics of, the subset of instances which fall into each region.

Our experiments show that the mean-based method is the best method for candidate partition generation and RMSE evaluation is the best hyperplane evaluation method, while the two leaf node selection methods perform similarly. The experiments also show that summary-based propositionalisation performs significantly better than count-based propositionalisation, albeit at the cost of noticeably increased running time, and that the relative ordering of the base learners, in terms of the classification accuracy, is highly dependent on the dataset.

Bagging of count-based `AdaProp` trees results in significant increases in accuracy over all datasets, while the randomization in attributes produces further increases in performance over some datasets. Count-based `AdaProp` performs poorly when compared to TLC, while summary-based `AdaProp` improves on the results of RELAGGS. Overall, the best results achieved by `AdaProp` are comparable to the best results achieved by the other existing multi-instance machine learning algorithms, especially for the image classification datasets considered in this study.

References

Andrews, S., Tsochantaridis, I., & Hofmann, T. (2002). "Support Vector Machines for Multiple-Instance Learning". *Advances in Neural Information Processing Systems*, *15*, 561–568.

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

Carson, C., Thomas, M., Belongie, S., Hellerstein, J. M., & Malik, J. (1999). "Blobworld: A System for Region-based Image Indexing and Retrieval". In *Visual Information and Information Systems* (pp. 509–517). Springer.

Chen, Y., Bi, J., & Wang, J. Z. (2006). "MILES: Multiple-Instance Learning via Embedded Instance Selection". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(12), 1931–1947.

Dietterich, T. G., Lathrop, R. H., & Lozano-Pérez, T. (1997). "Solving the Multiple Instance Problem with Axis-parallel Rectangles". *Artificial Intelligence*, *89*(1-2), 31–71.

Foulds, J., & Frank, E. (2008). "Revisiting Multiple-Instance Learning via Embedded Instance Selection". In *AI 2008: Advances in Artificial Intelligence* (pp. 300–310). Springer.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). "Additive Logistic Regression: a Statistical View of Boosting". *The Annals of Statistics*, *28*(2), 337–407.

Grauman, K., & Leibe, B. (2011). *"Visual Object Recognition"* (Vol. 11). Morgan & Claypool Publishers.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). "The WEKA Data Mining Software: an Update". *ACM SIGKDD Explorations Newsletter*, *11*(1), 10–18.

Holte, R. C. (1993). "Very Simple Classification Rules Perform Well on Most Commonly Used Datasets". *Machine Learning*, *11*(1), 63–90.

Krogel, M.-A., & Wrobel, S. (2003). "Facets of Aggregation Approaches to Propositionalization". In *Proceedings of the Work-in-Progress Track at the 13th International Conference on Inductive Logic Programming* (pp. 30–39).

Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". *International Journal of Computer Vision, volume = 60, number = 2, pages = 91–110, publisher = Springer*.

Maron, O., & Lozano-Pérez, T. (1998). "A Framework for Multiple-Instance Learning". In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems* (pp. 570–576). MIT Press.

Maron, O., & Ratan, A. L. (1998). "Multiple-Instance Learning for Natural Scene Classification". In *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 341–349). Morgan Kaufmann.

Mayo, M. (2007). "Effective Classifiers for Detecting Objects". In *Proceedings of the Fourth International Conference on Computational Intelligence, Robotics, and Autonomous Systems.*

Mayo, M., & Frank, E. (2011). "Experiments with Multi-View Multi-Instance Learning for Supervised Image Classification".

Opelt, A., Pinz, A., Fussenegger, M., & Auer, P. (2006). "Generic Object Recognition with Boosting". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(3), 416–431.

Reutemann, P., Pfahringer, B., & Frank, E. (2005). "A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners". In *AI 2004: Advances in Artificial Intelligence* (Vol. 3339, p. 1017-1023). Springer.

Srinivasan, A., Muggleton, S., King, R. D., & Sternberg, M. J. (1994). "Mutagenesis: ILP Experiments in a Non-determinate Biological Domain". In *Proceedings of the 4th International Workshop on Inductive Logic Programming* (Vol. 237, pp. 217–232).

Wang, C., Scott, S., Zhang, J., Tao, Q., Fomenko, D. E., & Gladyshev, V. N. (2004). "A Study in Modeling Low-conservation Protein Superfamilies". *CSE Technical Reports*, 35.

Weidmann, N., Frank, E., & Pfahringer, B. (2003). "A Two-Level Learning Method for Generalized Multi-instance Problems". In *Proceedings of the Fourteenth European Conference on Machine Learning* (pp. 468–479). Springer.

Yang, C., & Lozano-Perez, T. (2000). "Image Database Retrieval with Multiple-Instance Learning Techniques". In *Proceedings of the Sixteenth International Conference on Data Engineering* (pp. 233–243). IEEE.

Zhang, Q., Goldman, S. A., Yu, W., & Fritts, J. (2002). "Content-Based Image Retrieval Using Multiple-Instance Learning". In *Proceedings of the Nineteenth International Conference on Machine Learning* (pp. 682–689). Morgan Kaufmann.

Appendix A

Result charts

Figure A.1: Average accuracy (for `RandomForest`) by propositionalisation method

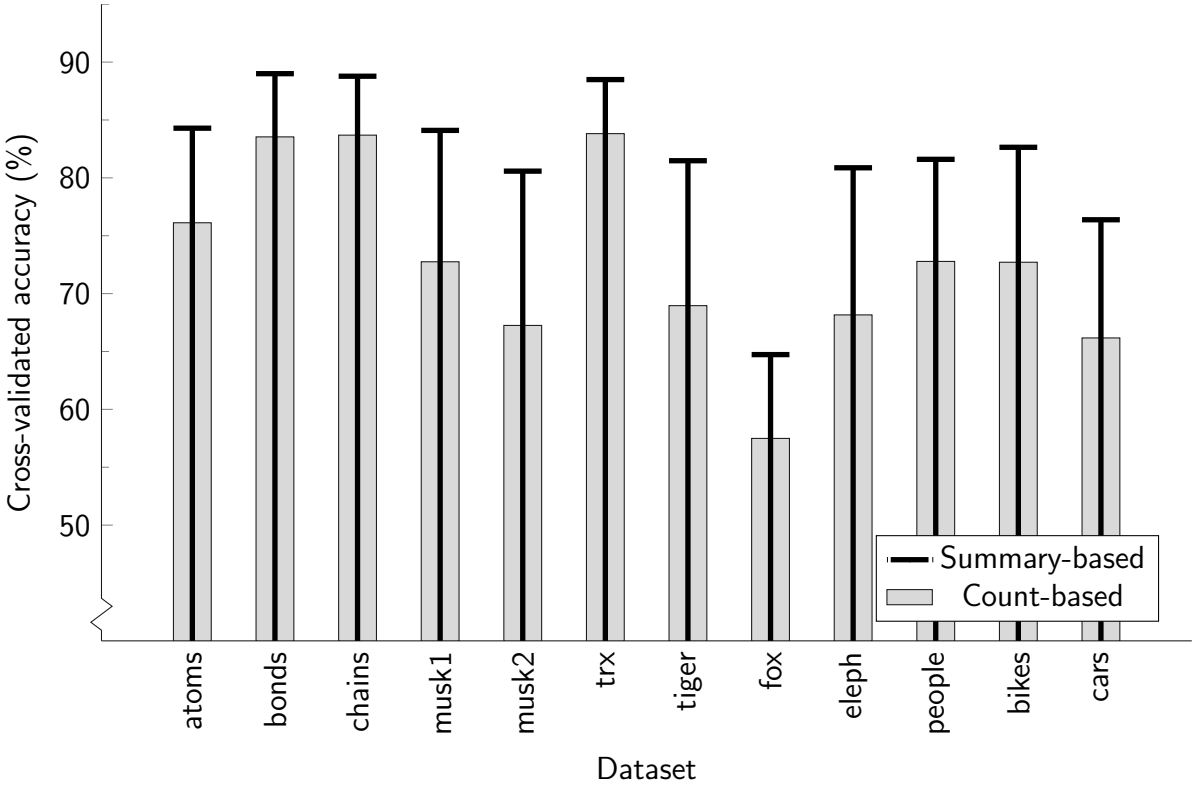


Figure A.2: Average accuracy (for `LogitBoost`) by propositionalisation method

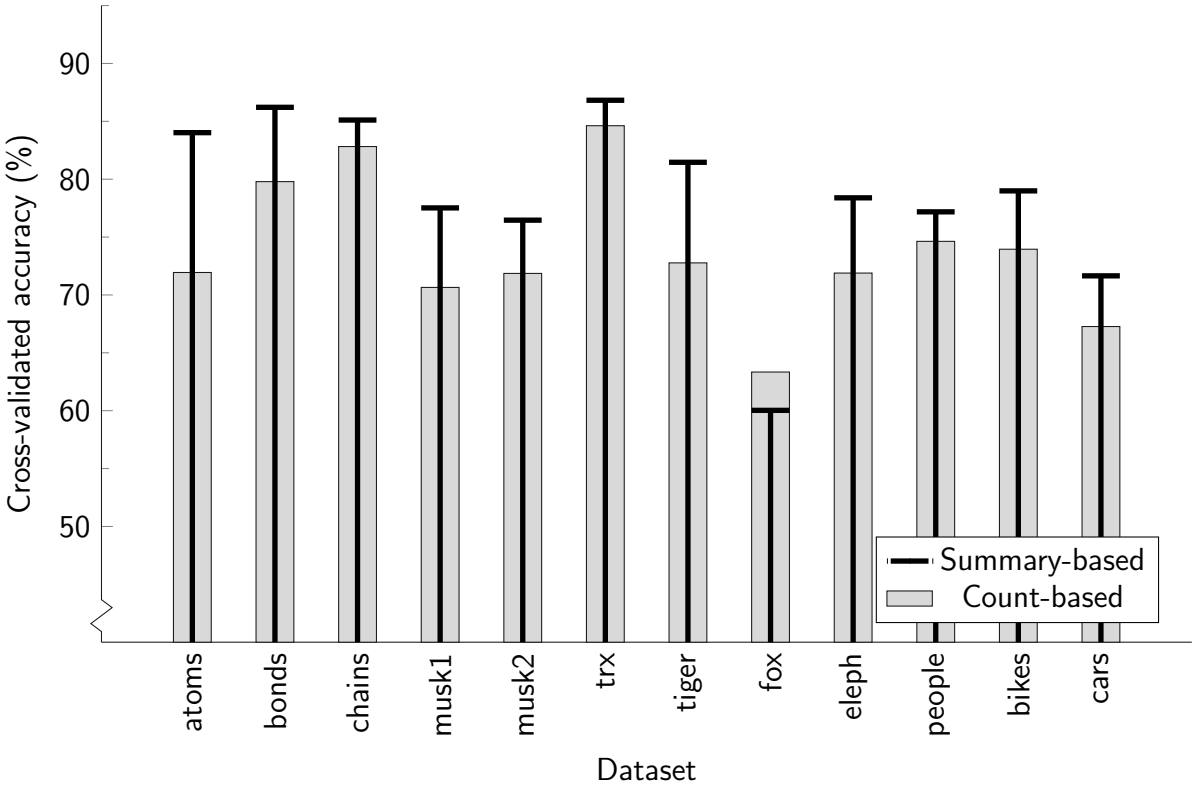


Figure A.3: Average accuracy (for CE) by propositionalisation method

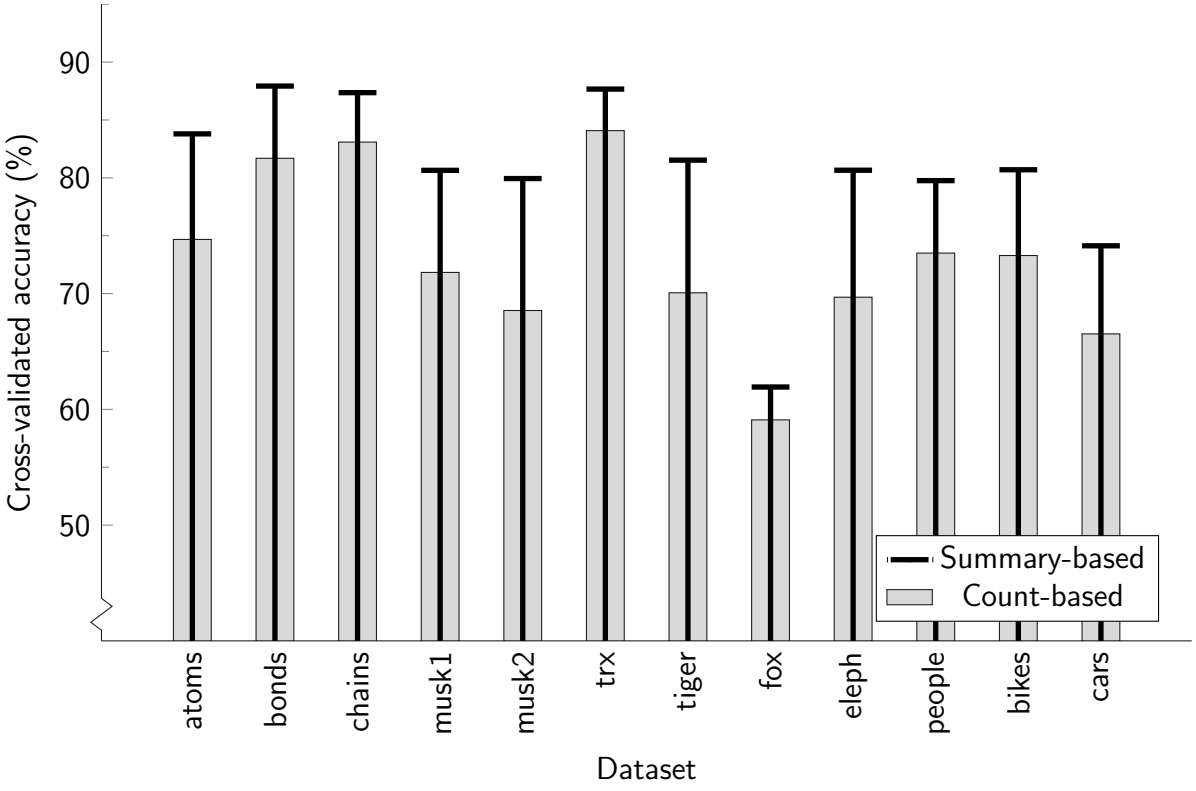


Figure A.4: Average accuracy (for RMSE) by propositionalisation method

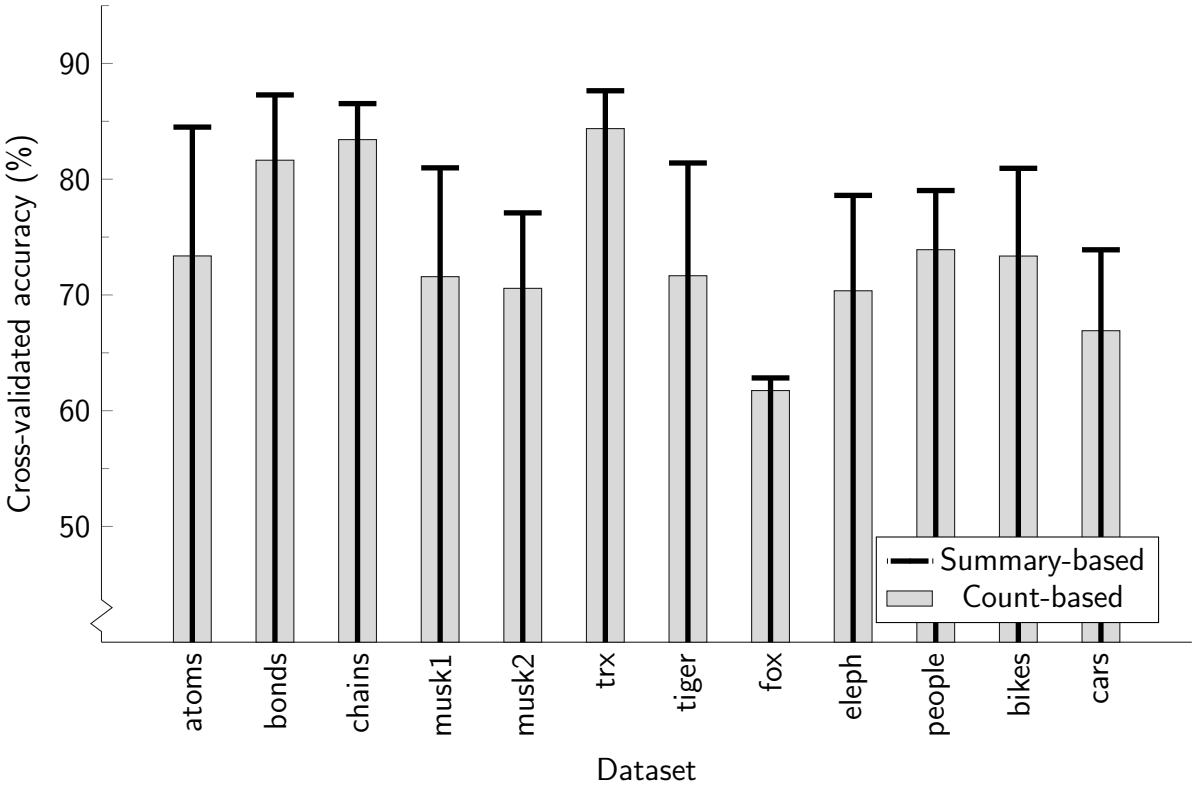


Figure A.5: Accuracy (for `RandomForest`) by evaluation method

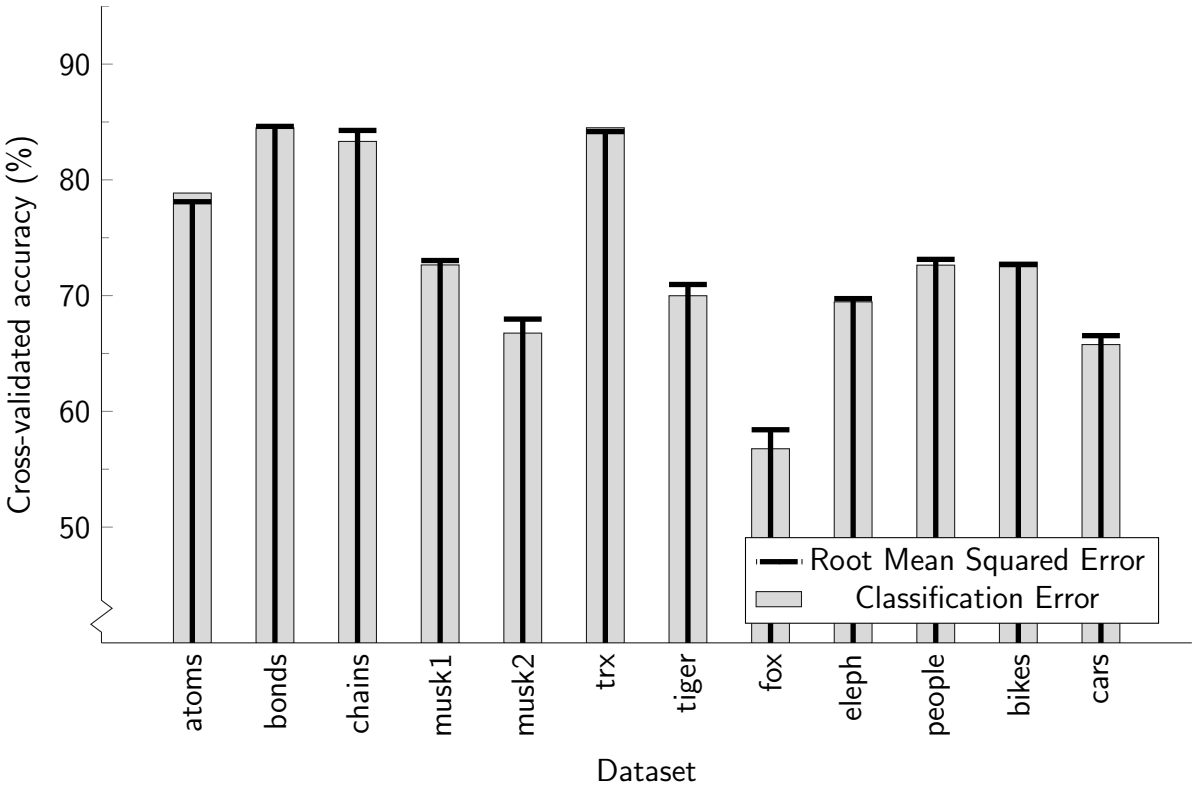


Figure A.6: Accuracy (for `LogitBoost`) by evaluation method

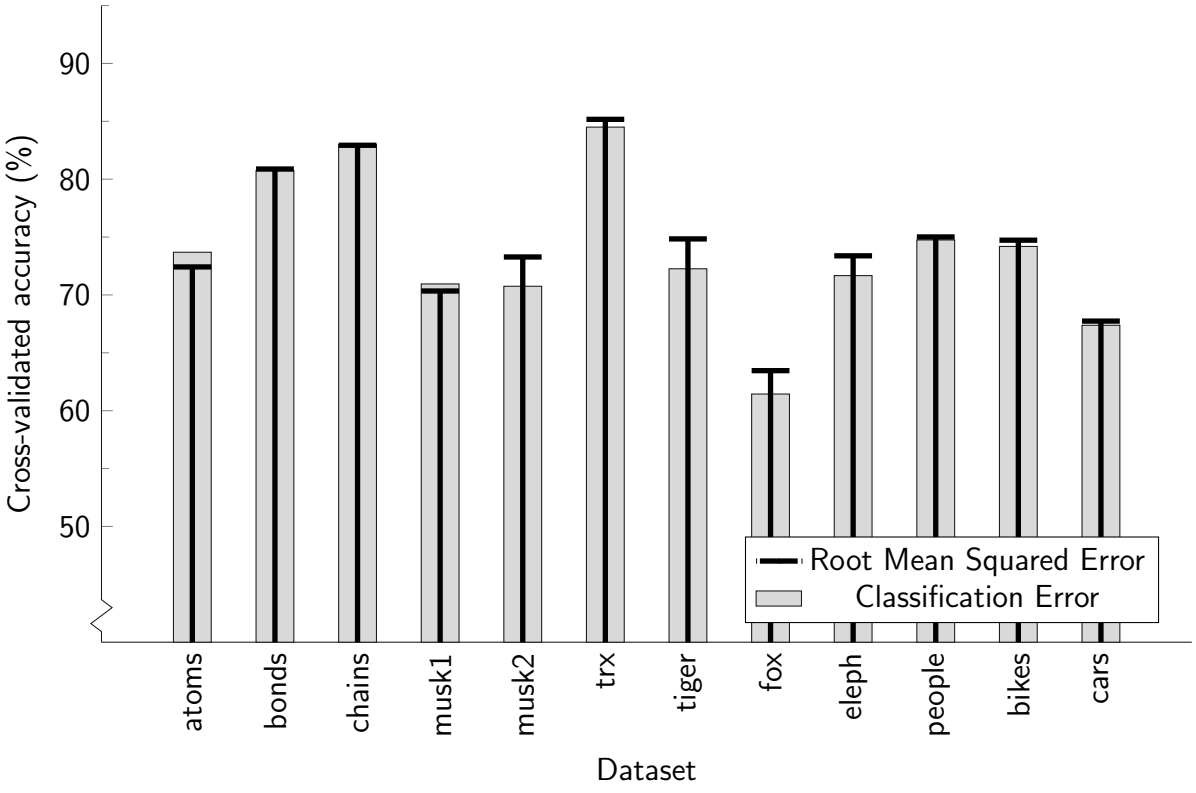


Figure A.7: Accuracy (with `RandomForest`) by leaf node selection strategy

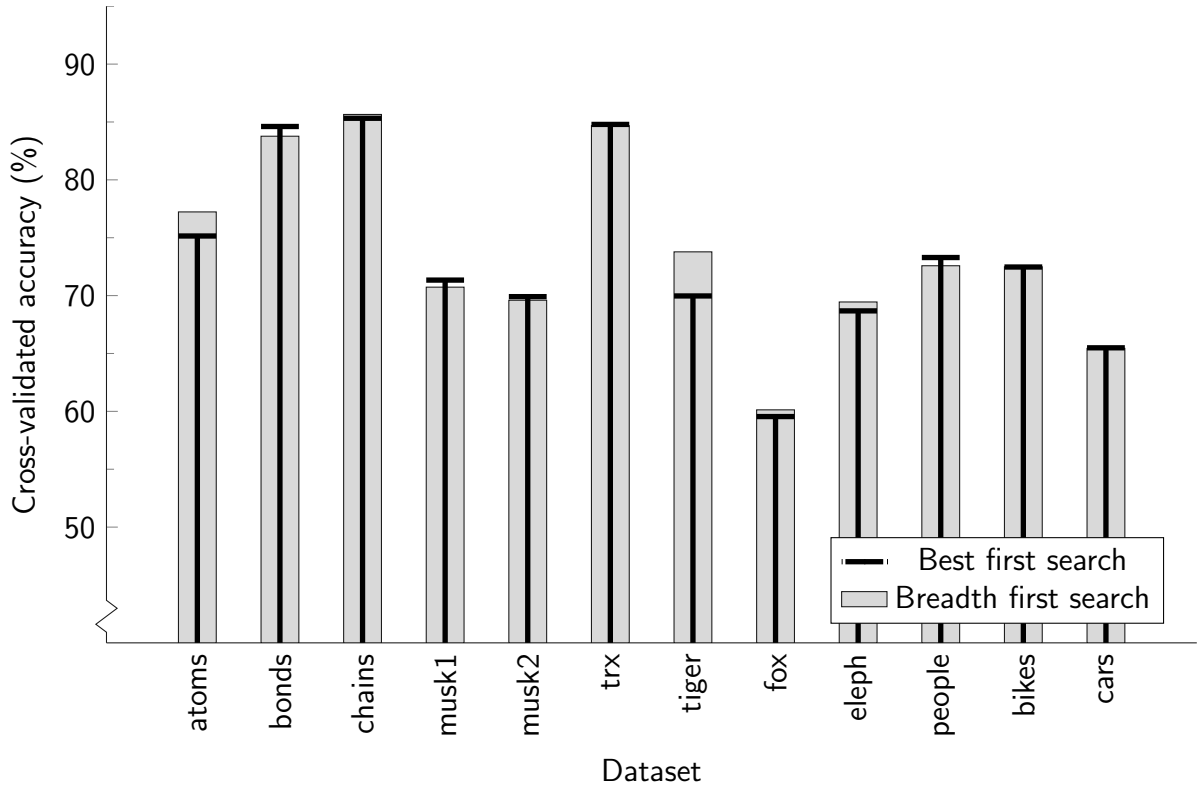


Figure A.8: Accuracy (with `LogitBoost`) by leaf node selection strategy

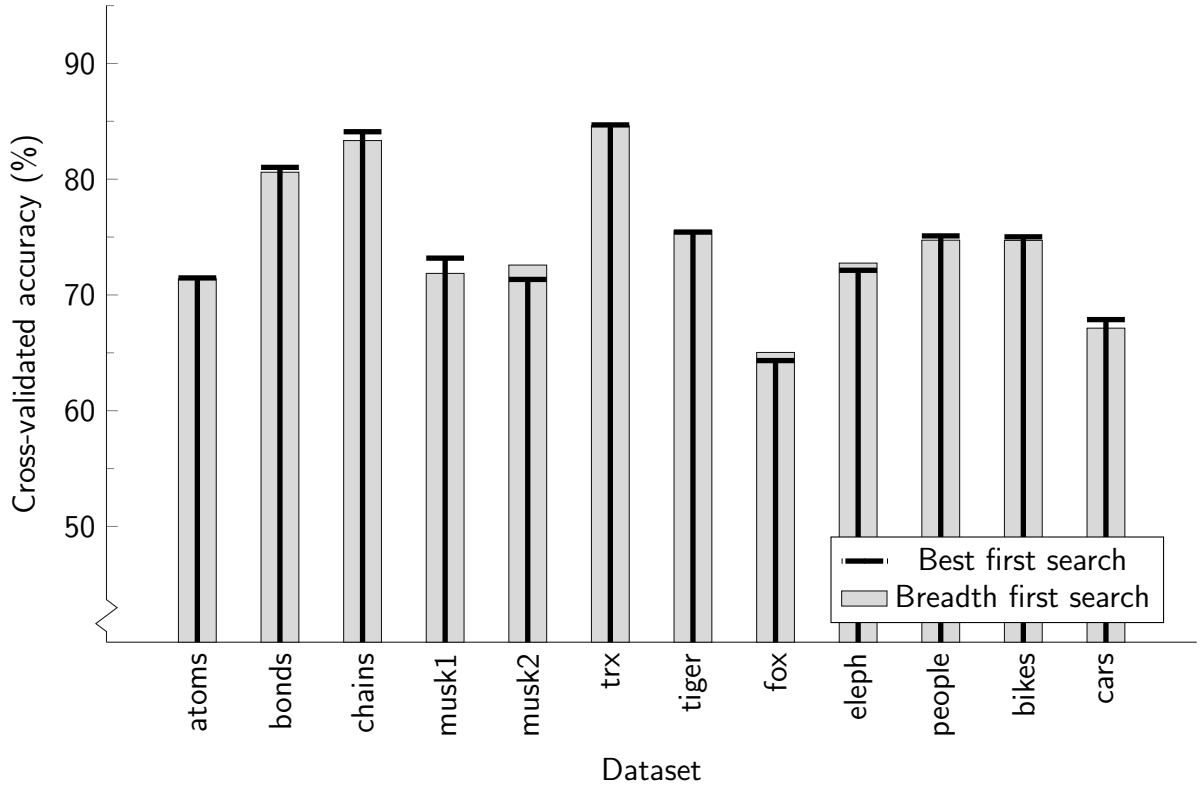


Figure A.9: Accuracy (with `LogitBoost`) - parameter selection

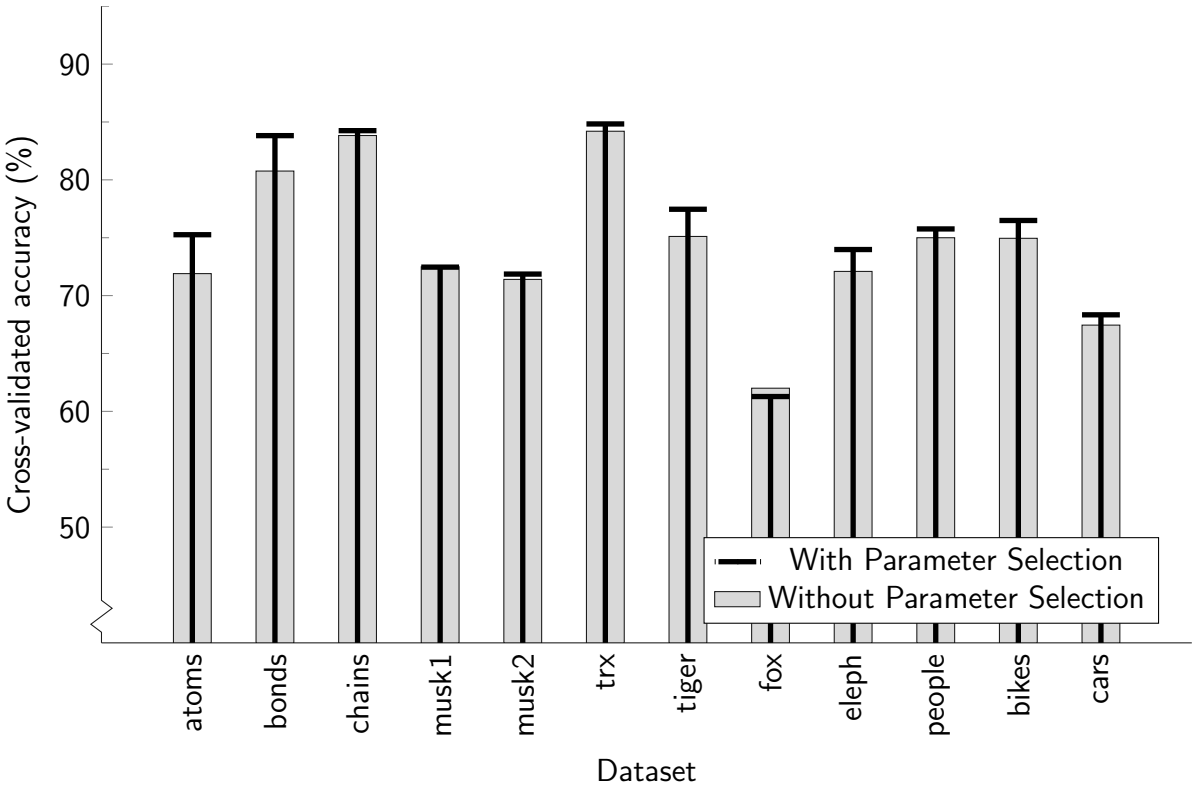
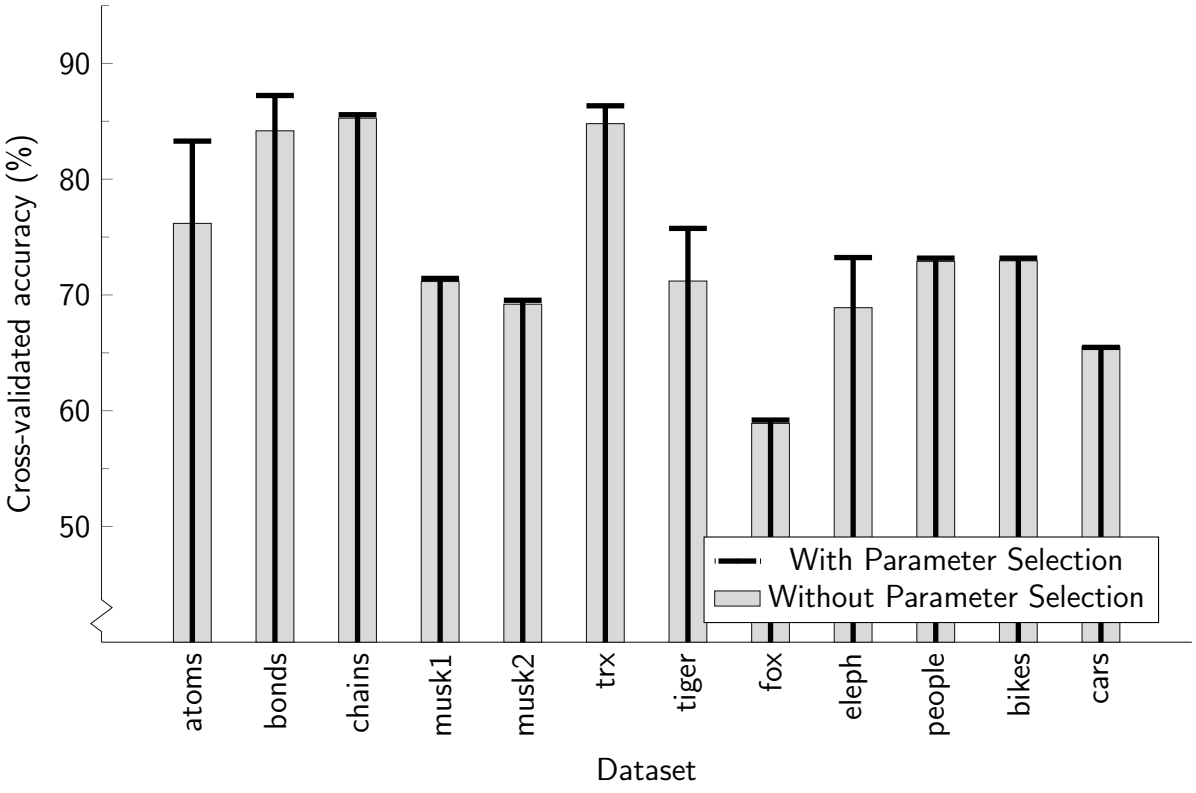


Figure A.10: Accuracy (with `RandomForest`) - parameter selection



Appendix B

Result tables

Table B.1: Data table for Figure 4.1: Average accuracy (for small trees) by propositionalisation method

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Count-based	74.0	81.7	83.3	71.7	69.6	84.2	70.9	60.4	70.0	73.7	73.3	66.7
Summary-based	84.2	87.6	87.0	80.8	78.5	87.7	81.5	62.4	79.6	79.4	80.8	74.0

The results were averaged over each combination of the following parameter settings:
Candidate Generation: mean, median or range
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`

Table B.2: Data table for Figures 4.2, 4.3, and 4.4: Average accuracy by candidate generation method

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Mean-based (avg)	73.0	80.3	81.4	71.7	70.0	83.9	69.8	61.1	70.6	73.7	72.9	67.3
Discretized (avg)	74.8	80.9	83.5	70.8	71.2	84.1	71.7	58.7	70.0	74.2	74.7	70.0
Mean-based (RF)	76.0	82.3	80.7	75.6	64.0	81.6	64.8	55.6	67.4	72.6	72.2	67.8
Discretized (RF)	73.4	81.3	79.4	66.2	67.7	82.7	69.3	61.4	66.3	73.9	73.4	69.6
Mean-based (LB)	71.5	79.3	81.8	69.8	72.9	85.0	72.3	63.8	72.2	74.3	73.3	67.1
Discretized (LB)	75.6	80.8	85.6	73.1	72.9	84.8	72.9	57.3	71.9	74.3	75.4	70.2

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search

Table B.3: Data table for Figures 4.5 and 4.6: Average accuracy by evaluation method

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
CE	76.3	82.6	83.2	71.8	68.8	84.5	71.1	59.1	70.6	73.7	73.5	66.6
RMSE	75.3	82.8	83.6	71.7	70.6	84.7	72.9	60.9	71.6	74.1	73.7	67.1
IG	75.3	82.9	83.3	71.9	71.3	84.8	72.8	61.0	71.2	73.8	73.1	67.2

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`

Table B.4: Data table for Figure 4.7: Accuracy by leaf node selection strategy

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Breadth first search	74.3	82.2	84.5	71.3	71.1	84.7	74.7	62.6	71.1	73.7	73.6	66.3
Best first search	73.3	82.8	84.7	72.3	70.6	84.7	72.7	61.9	70.4	74.2	73.8	66.7

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG
Base Learner: `RandomForest` or `LogitBoost`

Table B.5: Data table for Figure 4.8: Average accuracy by base learner

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
`LogitBoost`	73.1	80.8	83.0	70.6	72.0	84.8	73.6	62.5	72.5	74.9	74.5	67.6
`RandomForest`	78.5	84.6	83.8	72.8	67.4	84.3	70.5	57.6	69.6	72.9	72.8	66.2

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search

Table B.6: Data table for Figure 5.1: Average accuracy - Impact of parameter selection

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Tree Size = 7	74.0	82.5	84.6	71.8	70.3	84.5	73.2	60.5	70.5	74.0	74.0	66.5
Parameter Selected	79.3	85.5	84.9	72.0	70.7	85.6	76.6	60.2	73.6	74.5	74.8	66.9

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`

Table B.7: Data table for Figure 5.2: Average accuracy - Impact of bagging

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Without Bagging	74.0	82.5	84.6	71.8	70.3	84.5	73.2	60.5	70.5	74.0	74.0	66.5
With Bagging	81.9	87.9	88.2	78.4	78.8	87.6	82.5	63.3	79.0	78.4	78.5	71.8

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: RMSE only
Leaf Node Selection: breadth first search only
Base Learner: `RandomForest` or `LogitBoost`

Table B.8: Data table for Figure 5.3: Average bagged accuracy - Impact of randomization (for small trees)

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Without Randomization	81.9	87.9	88.2	78.4	78.8	87.6	82.5	63.3	79.0	78.4	78.5	71.8
With Randomization	83.4	87.6	87.9	80.3	76.1	87.4	75.8	61.5	69.8	76.5	73.1	68.7

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: RMSE only
Leaf Node Selection: breadth first search only
Base Learner: `RandomForest` or `LogitBoost`

Table B.9: Data table for Figure 5.4: Maximum bagged accuracy - Impact of randomization

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Without Randomization	83.4	89.4	88.7	79.7	79.2	88.5	84.2	65.4	80.6	79.1	79.4	72.5
With Randomization	87.9	89.4	89.9	82.3	79.3	88.4	79.1	63.7	74.1	78.3	74.7	70.8

The maximum result was selected over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: RMSE only
Leaf Node Selection: breadth first search only
Base Learner: `RandomForest` or `LogitBoost`

Table B.10: Data table for Figure 6.1: Maximum accuracy - TLC vs count-based `AdaProp`

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
TLC	86.8	88.0	89.4	87.6	81.4	87.6	81.6	67.0	86.6	82.4	82.0	76.6
`AdaProp`	84.5	87.5	86.6	76.3	74.5	86.6	79.6	66.0	75.9	76.3	77.2	68.8

The maximum result was selected over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`

Table B.11: Data table for Figure 6.2: Maximum bagged accuracy - TLC vs count-based `AdaProp`

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
TLC	86.8	88.7	89.5	88.1	81.4	88.9	81.6	67.0	86.9	82.4	83.0	76.6
`AdaProp`	87.9	89.4	89.9	82.3	79.3	88.5	84.2	65.4	80.6	79.1	79.4	72.5

The maximum result was selected over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: RMSE only
Leaf Node Selection: breadth first search only
Base Learner: `RandomForest` or `LogitBoost`
TLC: Default settings with `RandomForest` or `LogitBoost`

Table B.12: Data table for Figure 6.3: Maximum accuracy - RELAGGS vs summary-based `AdaProp`

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
RELAGGS	80.2	88.0	88.6	85.6	80.9	87.1	80.8	65.8	85.5	81.5	82.7	77.2
`AdaProp`	85.1	89.9	89.5	85.4	82.0	89.2	82.6	66.9	85.5	82.1	83.1	77.2

The maximum result was selected over each combination of the following parameter settings:
Propositionalisation: summary-based only
Candidate Generation: mean, median or range
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`
RELAGGS: Default settings with `RandomForest` or `LogitBoost`

Table B.13: Data table for Figure 6.4: Comparing `AdaProp` against the other MI algorithms

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Others	86.8	88.7	89.5	89.1	91.6	90.3	84.3	67.0	87.1	82.6	84.3	77.2
AdaProp	87.9	89.9	89.9	85.4	82.0	89.2	84.2	66.9	85.5	82.1	83.1	77.2

The results were maximised over all experiments conducted in this report and combined with the results from Foulds and Frank (2008). The respective algorithms are shown in Table B.14.

Table B.14: Configurations list for Table B.13 (and Figure 6.4)

Dataset	Others	AdaProp
atoms	Bagged TLC	Bagged count-based
bonds	Bagged TLC	Summary-based
chains	Bagged TLC	Bagged count-based
musk1	MILES with SMO (RBF)	Summary-based
musk2	MILES with 1-Norm SVM	Summary-based
trx	AdaBoost with Opt.Ball	Summary-based
tiger	MIWrapper over RandomForest	Bagged count-based
fox	SimpleMI over AdaBoost with DecisionStump	Summary-based
elephant	MIWrapper over RandomForest	Summary-based
people	MIWrapper over RandomForest	Summary-based
bikes	SimpleMI over 1-Norm SVM	Summary-based
cars	RELAGGS	Summary-based

Table B.15: Data table for Figures A.1, and A.2: Average accuracy by propositionalisation method and base learner

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Count, RF	76.1	83.5	83.7	72.8	67.3	83.8	69.0	57.5	68.2	72.8	72.7	66.2
Summary, RF	84.3	89.0	88.8	84.1	80.6	88.5	81.5	64.7	80.9	81.6	82.6	76.4
Count, LB	71.9	79.8	82.8	70.7	71.9	84.6	72.8	63.3	71.9	74.6	74.0	67.3
Summary, LB	84.0	86.2	85.1	77.5	76.5	86.8	81.5	60.0	78.4	77.2	79.0	71.7

The results were averaged over each combination of the following parameter settings:
Candidate Generation: mean, median or range
Hyperplane Evaluation: CE, RMSE or IG
Leaf Node Selection: breadth first search or best first search

Table B.16: Data table for Figures A.3, and A.4: Average accuracy by propositionalisation method and evaluation method

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
Count, CE	74.7	81.7	83.1	71.8	68.5	84.1	70.1	59.1	69.7	73.5	73.3	66.5
Summary, CE	83.8	87.9	87.4	80.7	79.9	87.7	81.5	61.9	80.7	79.8	80.7	74.1
Count, RMSE	73.4	81.6	83.4	71.6	70.6	84.4	71.7	61.7	70.4	73.9	73.4	66.9
Summary, RMSE	84.5	87.3	86.5	81.0	77.1	87.6	81.4	62.8	78.6	79.0	80.9	73.9

The results were averaged over each combination of the following parameter settings:
Candidate Generation: mean, median or range
Leaf Node Selection: breadth first search or best first search
Base Learner: `RandomForest` or `LogitBoost`

Table B.17: Data table for Figures A.5, and A.6: Average accuracy by base learner and evaluation method

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
RF, CE	78.9	84.5	83.3	72.7	66.8	84.5	70.0	56.8	69.5	72.6	72.9	65.8
RF, RMSE	78.1	84.6	84.3	73.0	68.0	84.2	71.0	58.4	69.7	73.1	72.7	66.5
LB, CE	73.7	80.7	83.0	71.0	70.8	84.5	72.3	61.4	71.7	74.8	74.2	67.4
LB, RMSE	72.4	80.9	82.9	70.3	73.3	85.2	74.8	63.5	73.4	75.0	74.7	67.7

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Leaf Node Selection: breadth first search or best first search

Table B.18: Data table for Figures A.7, and A.8: Average accuracy by base learner and leaf node selection strategy

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
RF, Breadth	77.2	83.8	85.7	70.7	69.6	84.7	73.8	60.1	69.5	72.6	72.5	65.4
RF, Best	75.2	84.6	85.3	71.3	69.9	84.8	70.0	59.6	68.7	73.3	72.5	65.5
LB, Breadth	71.4	80.6	83.3	71.9	72.6	84.6	75.6	65.0	72.8	74.7	74.7	67.1
LB, Best	71.5	81.0	84.1	73.2	71.3	84.7	75.4	64.3	72.1	75.1	75.0	67.9

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG

Table B.19: Data table for Figures A.9, and A.10: Average accuracy by base learner and parameter selection

	atoms	bonds	chains	musk1	musk2	trx	tiger	fox	eleph	people	bikes	cars
RF, Without	76.2	84.2	85.3	71.2	69.2	84.8	71.2	58.9	68.9	72.9	73.0	65.5
RF, With	83.3	87.2	85.6	71.4	69.5	86.3	75.8	59.2	73.2	73.2	73.2	65.5
LB, Without	71.9	80.8	83.8	72.5	71.4	84.2	75.1	62.0	72.1	75.0	75.0	67.5
LB, With	75.3	83.8	84.3	72.5	71.9	84.8	77.5	61.3	74.0	75.8	76.5	68.3

The results were averaged over each combination of the following parameter settings:
Propositionalisation: count-based only
Candidate Generation: mean only
Hyperplane Evaluation: CE, RMSE or IG