K-ary Regression Forests for Continuous Pose and Direction Estimation

Kota Hara and Rama Chellappa
Center for Automation Research, University of Maryland, College Park, MD 20742
{ kotahara, rama }@umiacs.umd.edu

Abstract

In this work, we propose a novel K-ary splitting method for regression trees and incorporate it into the regression forest framework. Unlike standard binary splitting, where the splitting rule is selected from a predefined set of binary splitting rules via trial-and-error, the proposed K-ary splitting method first finds clusters of the training data which at least locally minimize the empirical loss without considering the input space. Then splitting rules which preserve the found clusters as much as possible are determined by casting the problem into a multiclass classification problem. Consequently, our K-ary splitting method enjoys more freedom in choosing the splitting rules, resulting in more efficient tree structures. In addition to the Euclidean target space, we present a variant which can naturally deal with a circular target space by the proper use of circular statistics. We apply the regression forest employing our K-ary splitting to head pose estimation (Euclidean target space) and car direction estimation (circular target space) and demonstrate that the proposed method significantly outperforms state-of-the-art methods as well as regression forests based on standard binary splitting. The code will be available at our website.

1. Introduction

Regression has been successfully applied to various computer vision tasks such as head pose estimation [19, 15], object direction estimation [15, 30], human body pose estimation [3, 28] and facial point localization [10, 6], which require continuous outputs. In regression, a mapping from an input space to a target space is learned from the training data. The learned mapping function is used to predict the target values for new data. In computer vision, the input space is typically the high-dimensional image feature space and the target space is a low-dimensional space which represents some high level concepts presented in the given images. Due to the complex input-target relationship, non-linear regression methods are usually employed for computer vision tasks.

Among several non-linear regression methods, regression forests [4] have been shown to be effective for various computer vision problems [28, 9, 10]. The regression forest is an ensemble learning method which combines several regression trees [5] into a strong regressor. The regression trees define recursive partitioning of the input space and each leaf node contains a model for the predictor (most commonly a constant model). In the training stage, the trees are grown in order to reduce the empirical loss over the training data. In the regression forest, each regression tree is independently trained using a random subset of training data and prediction is done by finding the average/mode of outputs from all the trees.

As a node splitting algorithm, binary splitting is commonly employed for regression trees, however, it has limitations regarding how it partitions the input space. First, tree structures are limited to binary trees. Second, a splitting rule at each node is selected by trial and error from a predefined set of splitting rules, which are typically simple thresholding operations on a single dimension of the input. Due to these problems, the obtained trees are not efficient in reducing the empirical loss.

To overcome the above drawbacks of the standard binary splitting scheme, we propose a new K-ary splitting method and incorporate it into the regression forest framework. In our K-ary splitting method, each node in the tree can have more than two child nodes (hence K-ary). As an extension, we also propose an adaptive K-ary splitting method where K is automatically determined. The key difference from standard binary splitting is that clusters of the training data which at least locally minimize the empirical loss are first found without being restricted by a predefined set of splitting rules. Then splitting rules which preserve the found clusters as much as possible are determined by casting the problem into a multiclass classification problem. Unlike the standard binary splitting method, our K-ary splitting procedure enjoys more freedom in choosing the partitioning rules, resulting in more efficient regression tree structures. In addition to the method for the Euclidean target space, we present an extension which can naturally deal with a circular target space by the proper use of circular statistics.
We refer to regression forests (RF) employing our K-ary splitting algorithm as KRF and those employing the adaptive variant as AKRF. We test KRF and AKRF on Pointing’04 dataset for head pose estimation (Euclidean target space) and EPFL Multi-view Car Dataset for car direction estimation (circular target space) and observe that the proposed methods outperform state-of-the-art with 38.5% error reduction on Pointing'04 and 22.5% error reduction on EPFL Multi-view Car Dataset. Also KRF and AKRF significantly outperform other general regression methods including regression forests with the standard binary splitting.

2. Related work

Many multiway splitting methods have been proposed in the literature for classification. [14] proposed a multi-way splitting based on a single input dimension where the number of child nodes are determined by MDL. [2] obtain a hierarchy of intervals on each input dimension by a hierarchical clustering method which also takes into account the class distributions and selects a set of intervals which minimizes an impurity measure. [22] used linear discriminant analysis as a multiway splitting function which can naturally utilize all the input dimensions at once and does not rely on exhaustive search for the best splitting function. Several works (e.g., [29, 1]) based on SVM as a splitting function also used them for binary splitting.

For regression, a variant of regression trees called regression ferns realize multiway splitting. [11] proposed random regression ferns which partition the input space into $2^5$ regions based on the results of randomly selected $S$ binary splitting functions. In the testing time, multiple regression ferns are evaluated and the one which has the lowest error is selected. [8] employed a fern model in the boosted regression framework. Instead of randomly generating binary splitting functions, they selected a set of feature dimensions based on correlations between features and the targets.

The work most similar to our K-ary regression trees was proposed by Chou [8] who applied k-means like algorithm to the target space to find a locally optimal set of partitions. However, this method is limited to the case where the input is a categorical variable. Although we limit ourselves to continuous inputs, our formulation is more general and can be applied to any type of inputs by choosing appropriate classification methods. Furthermore, incorporating the multiway split regression trees into a regression forest framework has not been previously explored.

Regression has been widely applied for head pose estimation tasks. [19] used kernel partial least squares regression to learn a mapping from HOG features to head poses. Fenzig [15] learned a set of local feature generative model using RBF networks and estimated poses using MAP inference.

A few works considered direction estimation tasks where the direction ranges from $0^\circ$ and $360^\circ$. [20] modified regression forests so that the binary splitting minimizes a cost function specifically designed for direction estimation tasks. [30] applied supervised manifold learning and used RBF networks to learn a mapping from a point on the learnt manifold to the target space.

3. Methods

We denote a set of training data by $\{x_i, t_i\}_{i=1}^N$, where $x \in \mathbb{R}^p$ is an input vector and $t \in \mathbb{R}^q$ is a target vector. The goal of regression is to learn a function $F^*(\hat{X})$ such that the expected value of a certain loss function $\Psi(t, F(\hat{X}))$ is minimized:

$$F^*(\hat{X}) = \arg\min_{F(\hat{X})} \mathbb{E}[\Psi(t, F(\hat{X}))].$$

By approximating the above expected loss by an empirical loss and using the squared loss function, Eq (1) is reformulated as minimizing the sum of squared errors (SSE):

$$F^*(\hat{X}) = \arg\min_{F(\hat{X})} \sum_{i=1}^N ||t_i - F(\hat{X}_i)||^2_2.$$ (2)

However, other loss functions can also be used. In this paper we employ SSE except in Sec.3.5 where we use a loss function specialized for a circular target space.

In the following subsections, we first explain an abstracted regression tree algorithm, followed by the presentation of a binary splitting method normally employed for regression tree training. We then describe the details of our K-ary splitting method. An algorithm to adaptively determine $K$ for our K-ary splitting is presented, followed by a modification of our method for the circular target space, which is necessary for direction estimation tasks. Lastly, the regression forest framework for combining regression trees is presented.

3.1. Abstracted Regression Tree Model

Regression trees are grown by recursively partitioning the input space into a set of disjoint partitions, starting from a root node which corresponds to the entire input space. At each node splitting stage, a set of splitting rules and prediction models for each partition are determined so as to minimize the certain loss (error). A typical choice for a prediction model is a constant model which is determined as a mean target value of training samples in the partition. However, higher order models such as linear regression can also be used. Throughout this work, we employ the constant model. After each partitioning, corresponding children nodes are created and each training sample is forwarded to one of the child nodes. Each child node is further split if the number of the training samples belonging to that node is larger than a predefined number.
The essential component of regression tree training is an algorithm for splitting the nodes. Due to the recursive nature of training stage, it suffices to discuss the splitting of the root node where all the training data are available. The subsequent splitting is done with a subset of the training data belonging to each node in exactly the same manner.

Formally, we denote a set of $K$ disjoint partitions of the input space by $\mathcal{R} = \{r_1, r_2, \ldots, r_K\}$, a set of constant estimates associated with each partition by $\mathcal{A} = \{a_1, \ldots, a_K\}$ and the $K$ clusters of the training data by $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$ where

$$S_k = \{i : x_i \in r_k\}. \quad (3)$$

In the squared loss case, a constant estimate, $a_k$, for the $k$-th partition is computed as the mean target vector of the training samples that fall into $r_k$:

$$a_k = \frac{1}{|S_k|} \sum_{i \in S_k} t_i. \quad (4)$$

The sum of squared errors (SSE) associated with each child node is computed as:

$$SSE_k = \sum_{i \in S_k} ||t_i - a_k||^2_2, \quad (5)$$

where $SSE_k$ is the SSE for the $k$-th child node. Then the sum of squared errors on the entire training data is computed as:

$$SSE = \sum_{k=1}^{K} SSE_k = \sum_{k=1}^{K} \sum_{i \in S_k} ||t_i - a_k||^2_2. \quad (6)$$

The aim of training is to find a set of splitting rules defining the input partitions which minimizes the SSE.

Assuming there is no further splitting, the regression tree is formally represented as

$$H(x; \mathcal{A}, \mathcal{R}) = \sum_{k=1}^{K} a_k \mathcal{I}(x \in r_k), \quad (7)$$

where $\mathcal{I}$ is an indicator function. The regression tree outputs one of the elements of $\mathcal{A}$ depending on to which of the $\mathcal{R} = \{r_1, \ldots, r_K\}$, the new data $x$ belongs. As mentioned earlier, the child nodes are further split as long as the number of the training samples belonging to the node is larger than a predefined number.

### 3.2. Binary Splitting

In binary regression trees [5], $K$ is fixed at two. Each splitting rule is defined as a pair of the index of the input dimension and a threshold. Thus, each binary splitting rule corresponds to a hyperplane that is perpendicular to one of the axes. Among a predefined set of such splitting rules, the one which minimizes the overall SSE (Eq.6) is selected by trial and error.

The major drawback of the above splitting procedure is that the splitting rules are determined by exhaustively searching the best splitting rule among the predefined set of candidate rules. Essentially, this is the reason why only simple binary splitting rules defined as thresholding on a single dimension are considered in the training stage. Since the candidate rules are severely limited, the selected rules are generally not the best among all possible ways to partition the input space.

### 3.3. K-ary Splitting

In order to overcome the drawbacks of the binary splitting procedure, we propose K-ary splitting where $K$ can be larger than two. A graphical illustration of the algorithm is given in Fig[1]. At each node splitting stage, we first find ideal clusters $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ of the training data, those at least locally minimize the following objective function:

$$\min_{\mathcal{T}} \sum_{k=1}^{K} \sum_{i \in T_k} ||t_i - a_k||^2_2 \quad (8)$$

where $T_k = \{i : ||t_i - a_k||_2 \leq ||t_j - a_k||_2, \forall 1 \leq j \leq K\}$ and $a_k = \frac{1}{|T_k|} \sum_{i \in T_k} t_i$. This minimization can be done by applying the k-means clustering algorithm in the target space with $K$ as the number of clusters. Note the similarity between the objective functions in Eq.8 and Eq.6. The difference is that in Eq.8, clusters in $\mathcal{S}$ are indirectly determined by the splitting rules defined in the input space while clusters in $\mathcal{T}$ are directly determined by the k-means algorithm without taking into account the input space.

After finding $\mathcal{T}$, we find partitions $\mathcal{R} = \{r_1, \ldots, r_K\}$ of the input space which preserves $\mathcal{T}$ as much as possible. This task is equivalent to a $K$-class classification problem which aims at determining a cluster ID of each training data based on $x$. Although any multiclass classification method can be used, in this work, we employ L2-regularized L2-loss linear SVM with a one-versus-rest approach. Formally, we solve the following optimization for each cluster using LIBLINEAR [13]:

$$\min_{\mathbf{w}_k} ||\mathbf{w}_k||_2 + C \sum_{i=1}^{N} (\max(0, 1 - l_k^i \mathbf{w}_k^T \mathbf{x}_i))^2, \quad (9)$$

where $\mathbf{w}_k$ is the weight vector for the $k$-th cluster, $l_k^i = 1$ if $i \in T_k$ and $-1$ otherwise and $C > 0$ is a penalty parameter. We set $C = 1$ throughout the paper. Each training sample is forwarded to one of the $K$ children nodes by

$$k^* = \arg\max_{k \in \{1, \ldots, K\}} \mathbf{w}_k^T \mathbf{x}. \quad (10)$$
At the last stage of the node splitting procedure, we compute \( S \) (Eq.3) and \( A \) (Eq.4) based on the constructed splitting rules (Eq.10).

Unlike binary splitting, our K-ary splitting strategy allows each node to have more than two children nodes. Splitting rules are not limited to hyperplanes that are perpendicular to one of the axes. Furthermore, the clusters are found without being restricted by a set of predefined splitting rules in the input space. As a result, it is expected that upon successful clustering and classification, the sum of squared errors is much more efficiently minimized.

3.4. Adaptive determination of \( K \)

Since \( K \) is a parameter, we need to determine the value for \( K \) either manually or by time consuming cross-validation step. In order to avoid the cross-validation step while achieving comparative performance, we propose a method to adaptively determine \( K \) at each node based on the sample distribution. Since the value for \( K \) is adaptively determined at each node, the overall model can be more flexible than the one with fixed \( K \).

In this work we employ Bayesian Information Criterion (BIC) \([21, 22]\) as a measure to choose \( K \). BIC was also used in \([23]\) but with a different formulation. The BIC is designed to balance the model complexity and likelihood. As a result, when a target distribution is complex, a larger number of \( K \) is selected and when the target distribution is simple, a smaller value of \( K \) is selected. This is in contrast to the non-adaptive method where a fixed number of \( K \) is used regardless of the complexity of the distributions.

As k-means clustering itself does not assume any underlying probability distribution, we assume that the data are generated from a mixture of isotropic weighted Gaussians with a shared variance. The unbiased estimate for the shared variance is computed as

\[
\hat{\sigma}^2 = \frac{1}{N-K} \sum_{k=1}^{K} \sum_{i \in T_k} ||t_i - a_k||^2. \tag{11}
\]

We compute a point probability density for a data point \( t \) belonging to the \( k \)-th cluster as follows:

\[
p(t) = \frac{|T_k|}{N} \frac{1}{\sqrt{2\pi \hat{\sigma}^2}} \exp(-\frac{||t-a_k||^2}{2\hat{\sigma}^2}). \tag{12}
\]

Then after simple calculations, the log-likelihood of the data is obtained as

\[
\ln \mathcal{L}(\{t_i\}_{i=1}^N) = \ln \Pi_{i=1}^N p(t_i) = \sum_{k=1}^{K} \sum_{i \in T_k} \ln p(t_i) = \frac{-qN}{2} \ln(2\pi \hat{\sigma}^2) - \frac{N-K}{2} + \sum_{k=1}^{K} |T_k| \ln |T_k| - N \ln N
\]

Finally, the BIC for a particular value of \( K \) is computed as

\[
\text{BIC}_K = -2 \ln \mathcal{L}(\{t_i\}_{i=1}^N) + (K-1+qK+1) \ln N. \tag{14}
\]

At each node splitting stage, we run the k-means algorithm for each value of \( K \) in a manually specified range and select \( K \) with the smallest BIC. Throughout this work, we select \( K \) from \( \{2, 3, \ldots, 40\} \).

3.5. Modification for a Circular Target Space

1D direction estimation of the object such as cars and pedestrians is unique in that the target variable is periodic, namely, \( 0^\circ \) and \( 360^\circ \) represent the same direction angle. Thus, the target space can be naturally represented as a unit circle, which is a 1D manifold in \( R^2 \). To deal with such target space, special treatments are needed since the Euclidean distance is inappropriate. For instance, the distance between \( 10^\circ \) and \( 350^\circ \) should be shorter than that between \( 10^\circ \) and \( 50^\circ \).

\([24, 15, 12]\) deal with this problem by discretizing the direction interval into a set of small intervals and resort to the classification step. After the classification, \([15, 12]\) use regression to finely estimate the direction of the objects. Note that after estimating the direction interval, the target space is no longer a circle, thus standard regression methods can be used.

In our method, such direction estimation problems are naturally addressed by modifying the k-means algorithm and the computation of BIC. The remaining steps are kept unchanged. The k-means clustering method consists of computing cluster centroids and hard assignment of the training samples to the closest centroid. Finding the closest centroid on a circle is trivially done by using the length of the shorter arc as a distance. Due to the periodic nature of the variable, the arithmetic mean is not appropriate for computing the centroids. A typical way to compute the mean of angles is to first convert each angle to a 2D point on a unit circle.
circle. The arithmetic mean is then computed on a 2D plane and converted back to the angular value. More specifically, given a set of direction angles $t_1, \ldots, t_N$, the mean direction $a$ is computed by

$$a = \arctan2 \left( \frac{1}{N} \sum_{i=1}^{N} \sin t_i, \frac{1}{N} \sum_{i=1}^{N} \cos t_i \right). \quad (15)$$

It is known \cite{17} that $a$ minimizes the sum of a certain distance defined on a circle,

$$a = \arg \min_s \sum_{i=1}^{N} d(t_i, s) \quad (16)$$

where $d(q, s) = 1 - \cos(q - s) \in [0, 1]$. Thus, the k-means clustering using the above definition of means finds clusters $T = \{T_1, T_2, \ldots, T_K\}$ of the training data that at least locally minimize the following objective function,

$$\min T \sum_{k=1}^{K} \sum_{i \in T_k} \left(1 - \cos(t_i - a_k)\right) \quad (17)$$

where $T_k = \{i : 1 - \cos(t_i - a_k) \leq 1 - \cos(t_i - a_j), \forall 1 \leq j \leq K\}$.

Using the above k-means algorithm in our K-ary splitting essentially means that we employ distance $d(q, s)$ as a loss function in Eq.\[14\]. Although squared shorter arc length might be more appropriate for the direction estimation task, there is no constant time algorithm to find a mean which minimizes it. Also as will be explained shortly, the above definition of the mean coincides with the maximum likelihood estimate of the mean of a certain probability distribution defined on a circle.

As in the Euclidean target case, we can also adaptively determine the value for $K$ at each node using BIC. As a density function, the Gaussian distribution is not appropriate. A suitable choice is the von Mises distribution, which is a periodic continuous probability distribution defined on a circle,

$$p(t|a, \kappa) = \frac{1}{2 \pi I_0(\kappa)} \exp(\kappa \cdot \cos(t - a)) \quad (18)$$

where $a, \kappa$ are analogous to the mean and variance of the Gaussian distribution and $I_\lambda$ is the modified Bessel function of order $\lambda$. It is known \cite{16} that the maximum likelihood estimate of $a$ is computed by Eq\[15\] and that of $\kappa$ satisfies

$$\frac{I_1(\kappa)}{I_0(\kappa)} = \sqrt{\left(\frac{1}{N} \sum_{i=1}^{N} \sin t_i\right)^2 + \left(\frac{1}{N} \sum_{i=1}^{N} \cos t_i\right)^2}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \cos(t_i - a). \quad (19)$$

Note that, from the second term, the above quantity is the Euclidean norm of the mean vector obtained by converting each angle to a 2D point on a unit circle.

Similar to the derivation for the Euclidean case, we assume that the data are generated from a mixture of weighted von Mises distributions with a shared $\kappa$. The mean $a_k$ of k-th von Mises distribution is same as the mean of the k-th cluster obtained by the k-means clustering. The shared value for $\kappa$ is obtained by solving the following equation

$$\frac{I_1(\kappa)}{I_0(\kappa)} = \frac{1}{N} \sum_{k=1}^{K} \sum_{i \in T_k} \cos(t_i - a_k). \quad (20)$$

Since there is no closed form solution for the above equation, we use the following approximation proposed in \cite{23},

$$\kappa \approx \frac{1}{2(1 - \frac{I_1(\kappa)}{I_0(\kappa)})}. \quad (21)$$

Then, a point probability density for a data point $t$ belonging to the k-th cluster is computed as:

$$p(t|a_k, \kappa) = \frac{|T_k|}{N} \frac{\exp(\kappa \cdot \cos(t - a_k))}{2\pi I_0(\kappa)}. \quad (22)$$

After simple calculations, the log-likelihood of the data is obtained as

$$\ln \mathcal{L(\{t_i\}_{i=1}^{N})} = \ln \prod_{t=1}^{N} p(t_i) = \sum_{k=1}^{K} \sum_{i \in T_k} \ln p(t_i) =$$

$$-N \ln(2\pi I_0(\kappa)) + \kappa \sum_{k=1}^{K} \sum_{i \in T_k} \cos(t_i - a_k) + \sum_{k=1}^{K} |T_k| \ln |T_k|$$

$$-N \ln N. \quad (23)$$

Finally, the BIC for a particular value of $K$ is computed as

$$\text{BIC}_K = -2 \ln \mathcal{L(\{t_i\}_{i=1}^{N})} + 2K \ln N. \quad (24)$$

where the last term is obtained by putting $q = 1$ into the last term of Eq\[14\].

### 3.6. Regression Forest

We use the regression forest as the final regression model. The regression forest is an ensemble learning method for regression which first constructs multiple regression trees from random subsets of training data and outputs the mean of the outputs from each regression tree. We denote the ratio of random samples as $\beta \in (0, 1.0]$. For the Euclidean target case, arithmetic mean is used to obtain the final estimate and for the circular target case, the mean defined in Eq\[15\] is used.

For the regression forest with binary regression trees, an additional randomness is injected. In finding the best splitting function at each node, only a randomly selected subset
of the feature dimensions is considered. We denote the ratio of randomly chosen feature dimensions as $\gamma \in (0, 1.0]$. For the regression forest with our K-ary regression trees, we always consider all feature dimensions. However, another form of randomness is naturally injected by randomly selecting the data points as the initial cluster centroids in the k-means algorithm.

4. Experiments

4.1. Head Pose Estimation

We test the effectiveness of KRF and AKRF for the Euclidean target space on the head pose estimation task. We adopt Pointing’04 dataset [18]. The dataset contains head images of 15 subjects and for each subject there are two series of 93 images with different poses. Each pose is represented as pitch and yaw and 93 poses consists of all elements of $(\text{pitch}, \text{yaw}) \in \{-60^\circ, -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ, 60^\circ\} \times \{-90^\circ, -75^\circ, -60^\circ, -45^\circ, -30^\circ, -15^\circ, 0^\circ, 15^\circ, 30^\circ, 45^\circ, 60^\circ, 75^\circ, 90^\circ\} \times \{0^\circ\}$. The dataset comes with manually specified bounding boxes indicating the head regions. Based on the bounding boxes, we crop and resize the image patches to $64 \times 64$ pixels image patches and compute multiscale HOG from each image patch with cell size $8, 16, 32$ and $2 \times 2$ cell blocks. The orientation histogram for each cell is computed with signed gradients for 9 orientation bins. The resulting HOG feature is 2124 dimensional.

First, we compare the KRF and AKRF with other general regression methods using the same image features. We choose standard binary regression forest (BRF) [4], kernel PLS [26] and $\epsilon$-SVR with RBF kernels [31], all of which have been widely used for various computer vision tasks. The first series of images from all subjects are used as training set and the second series of images are used for testing. The performance is measured by Mean Absolute Error in degree. For KRF, AKRF and BRF, we terminate node splitting once the number of training data associated with each leaf node is less than 5. The number of trees combined is set to 20 for KRF and AKRF and 100 for BRF, which gives the advantage to BRF. We observe that more number of trees leads to better performance but the improvement becomes more and more marginal. $K$ for KRF, $\beta$ for KRF, AKRF and BRF and $\gamma$ for BRF are all determined by 5-fold cross-validation on the training set. For kernel PLS, we use the implementation provided by the author of [26] and for $\epsilon$-SVR, we use LIBSVM package [7]. All the parameters for kernel PLS and $\epsilon$-SVR are also determined by 5-fold cross-validation. As can be seen in Table 1, both KRF and AKRF work significantly better than other regression methods. Also our methods are computationally efficient (Table 1). KRF and AKRF take only 7.7 msec and 8.7 msec, respectively, to process one image including feature computation with a single thread.

Table 1. MAE in degree of different regression methods on the Pointing’04 dataset. Time to process one image including HOG computation is also shown.

<table>
<thead>
<tr>
<th>Methods</th>
<th>yaw</th>
<th>pitch</th>
<th>average</th>
<th>time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRF</td>
<td>5.32</td>
<td>3.52</td>
<td>4.42</td>
<td>7.7</td>
</tr>
<tr>
<td>AKRF</td>
<td>5.49</td>
<td>4.18</td>
<td>4.83</td>
<td>8.7</td>
</tr>
<tr>
<td>BRF [4]</td>
<td>7.47</td>
<td>8.01</td>
<td>7.74</td>
<td>11.1</td>
</tr>
<tr>
<td>Kernel PLS [26]</td>
<td>7.35</td>
<td>7.02</td>
<td>7.18</td>
<td>86.2</td>
</tr>
<tr>
<td>$\epsilon$-SVR [31]</td>
<td>7.34</td>
<td>7.02</td>
<td>7.18</td>
<td>189.2</td>
</tr>
</tbody>
</table>

Table 2 compares KRF and AKRF with prior art. Since the previous work report the 5-fold cross-validation estimate on the whole dataset, we also follow the same protocol. KRF and AKRF advance state-of-the-art with 38.5% and 29.7% reduction in the average MAE, respectively.

Table 2. Head pose estimation results on the Pointing’04 dataset

<table>
<thead>
<tr>
<th>Methods</th>
<th>yaw</th>
<th>pitch</th>
<th>average</th>
<th>time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRF</td>
<td>5.29</td>
<td>2.51</td>
<td>3.90</td>
<td></td>
</tr>
<tr>
<td>AKRF</td>
<td>5.50</td>
<td>3.41</td>
<td>4.46</td>
<td></td>
</tr>
<tr>
<td>Haj [19] PLS</td>
<td>5.94</td>
<td>6.73</td>
<td>6.34</td>
<td></td>
</tr>
<tr>
<td>Haj [19] PLS</td>
<td>6.56</td>
<td>6.61</td>
<td>6.59</td>
<td></td>
</tr>
<tr>
<td>AKRF</td>
<td>11.29</td>
<td>10.52</td>
<td>10.91</td>
<td></td>
</tr>
</tbody>
</table>

Fig 2 shows the effect of $K$ of KRF on the average MAE along with the average MAE of AKRF. In this experiment, the cross-validation process successfully selects $K$ with the best performance. AKRF works better than KRF with the second best $K$. The overall training time is much faster with AKRF since the cross-validation step for determining the value of $K$ is not necessary. To train a single regression tree with $\beta = 1$, AKRF takes only 6.8 sec while KRF takes 331.4sec for the cross-validation and 4.4sec for training a final model. Finally, some estimation results by AKRF on the second sequence of person 13 are shown in Fig 3.
4.2. Car Direction Estimation

We test KRF and AKRF for circular target space (denoted as KRF-circle and AKRF-circle respectively) on the EPFL Multi-view Car Dataset [24]. The dataset contains 20 sequences of images of cars with various directions. Each sequence contains images of only one car. In total, there are 2299 images in the dataset. Each image comes with a bounding box specifying the location of the car and ground truth for the direction of the car. The direction ranges from 0° to 360°. As input features, multiscale HOG features with the same parameters as in the previous experiment are extracted from 64 × 64 pixels image patches obtained by resizing the given bounding boxes.

The algorithm is evaluated by using the first 10 sequences for training and the remaining 10 sequences for testing. In Table 3 we compare the KRF-circle and AKRF-circle with previous work. We also include the performance of BRF, Kernel PLS and ϵ-SVR with RBF kernels using the same HOG features. For BRF, we extend it to directly minimize the same loss function (\(d(q,s) = 1 - \cos(q - s)\)) as with KRF-circle and AKRF-circle (denoted by BRF-circle). For Kernel PLS and ϵ-SVR, we first map direction angles to 2d points on a unit circle and train regressors using the mapped points as target values. In testing phase, a 2d point coordinate \((x,y)\) is first estimated and then mapped back to the angle by \(\text{atan2}(y,x)\). All the parameters are determined by leave-one-sequence-out cross-validation on the training set. The performance is evaluated by the Mean Absolute Error (MAE) measured in degrees. In addition, the MAE of 90-th percentile of the absolute errors and that of 95-th percentile are reported.

As can be seen from Table 3, both KRF-circle and AKRF-circle work much better than existing regression methods. In particular, the improvement over BRF-circle is notable. Our methods also advance state-of-the-art with 22.5% and 20.7% reduction in MAE from the previous best method, respectively. In Fig. 4 we show the MAE of AKRF-circle computed on each sequence in the testing set. The performance varies significantly among different sequences (car models). Fig. 5 shows some representative results from the worst three sequences in the testing set (seq 16, 20 and 15). We notice that most of the failure cases are due to the flipping errors (≈ 180°) which mostly occur at particular intervals of directions. Fig. 6 shows the effect of \(K\) of KRF-circle. The performance of the AKRF-circle is better than that of KRF-circle with \(K\) selected by the cross-validation.

<table>
<thead>
<tr>
<th>Method</th>
<th>MAE (°) 90-th percentile</th>
<th>MAE (°) 95-th percentile</th>
<th>MAE (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KRF-circle</td>
<td>8.32</td>
<td>16.76</td>
<td>24.80</td>
</tr>
<tr>
<td>AKRF-circle</td>
<td>7.73</td>
<td>16.18</td>
<td>24.24</td>
</tr>
<tr>
<td>BRF-circle</td>
<td>20.99</td>
<td>27.97</td>
<td>35.23</td>
</tr>
<tr>
<td>Kernel PLS</td>
<td>16.86</td>
<td>21.20</td>
<td>27.65</td>
</tr>
<tr>
<td>ϵ-SVR</td>
<td>17.38</td>
<td>22.70</td>
<td>29.41</td>
</tr>
<tr>
<td>Fenzi et al.</td>
<td>14.51</td>
<td>22.83</td>
<td>31.27</td>
</tr>
<tr>
<td>Torki et al.</td>
<td>19.4</td>
<td>26.7</td>
<td>33.98</td>
</tr>
<tr>
<td>Ozuysal et al.</td>
<td>-</td>
<td>-</td>
<td>46.48</td>
</tr>
</tbody>
</table>

Figure 4. MAE of AKRF computed on each sequence in the testing set

Figure 5. Representative results from the worst three sequences in the testing set. The numbers under each image are the ground truth direction (left) and the estimated direction (right). Most of the failure cases are due to the flipping error.
K-means nodes, algorithm for regression tree training where the number of children nodes, $K$, can be larger than two and splitting rules are not restricted to hyperplanes that are perpendicular to one of the axes. Unlike previous works, all these features are accomplished without relying on a trial-and-error process to find the best splitting rules from a predefined set of rules. Combined with the regression forest framework, our methods work significantly better than state-of-the-art methods on head pose estimation and car direction estimation tasks.

5. Conclusion

In this paper, we proposed a novel K-ary splitting algorithm for regression tree training where the number of children nodes, $K$, can be larger than two and splitting rules are not restricted to hyperplanes that are perpendicular to one of the axes. Unlike previous works, all these features are accomplished without relying on a trial-and-error process to find the best splitting rules from a predefined set of rules. Combined with the regression forest framework, our methods work significantly better than state-of-the-art methods on head pose estimation and car direction estimation tasks.

References