Non-Iterative Methods for Classification, Forecasting and Visual Tracking

Part II: Random Forest and Variants

Dr P. N. Suganthan **epnsugan@ntu.edu.sg**
School of EEE, NTU, Singapore

Some Software Resources Available from:
https://github.com/P-N-Suganthan

DeepLearn 2019
Warsaw, Poland
22 July – 26 July 2019

SEMCCO 2020 & FANCCO 2020

(includes all neural, fuzzy, swarm and evolutionary topics)

HUST, Wuhan, China

June 2020

IEEE SSCI 2019, CIEL 2019, SDE 2019, SIS 2019, etc.

http://ssci2019.org/

(includes all neural, fuzzy, swarm and evolutionary topics)

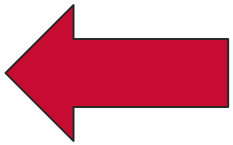
Randomization-Based ANN, Pseudo-Inverse Based Solutions, Kernel Ridge Regression, Random Forest and Related Topics

http://www.ntu.edu.sg/home/epnsugan/index_files/RNN-Moore-Penrose.htm
http://www.ntu.edu.sg/home/epnsugan/index_files/publications.htm

Outline

Part I: Classification by

- Neural Networks such as the RVFL
- Kernel Ridge Regression (KRR)
- Random Forest (RF)



Part II: Time Series Forecasting & Classification

(by RVFL, RF, Deep Learners and others)

Part III: Visual Tracking

(by RVFL, RF, KRR, etc.)

Beware of Notational inconsistency ….

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Ensemble of Classifiers

- A committee of classifiers trained either independently or sequentially.
- Obtained by perturbing the training set or injecting some randomness in each classifier and aggregating the outputs of the these classifiers in a suitable way.
- Decision trees (DT) and Neural networks (NN): commonly used classifiers for constructing ensembles

Y. Ren, L. Zhang, and P. N. Suganthan, "Ensemble Classification and Regression – Recent Developments, Applications and Future Directions," IEEE Computational Intelligence Magazine, **DOI:** 10.1109/MCI.2015.2471235 **, Feb 2016**.

Decision Tree

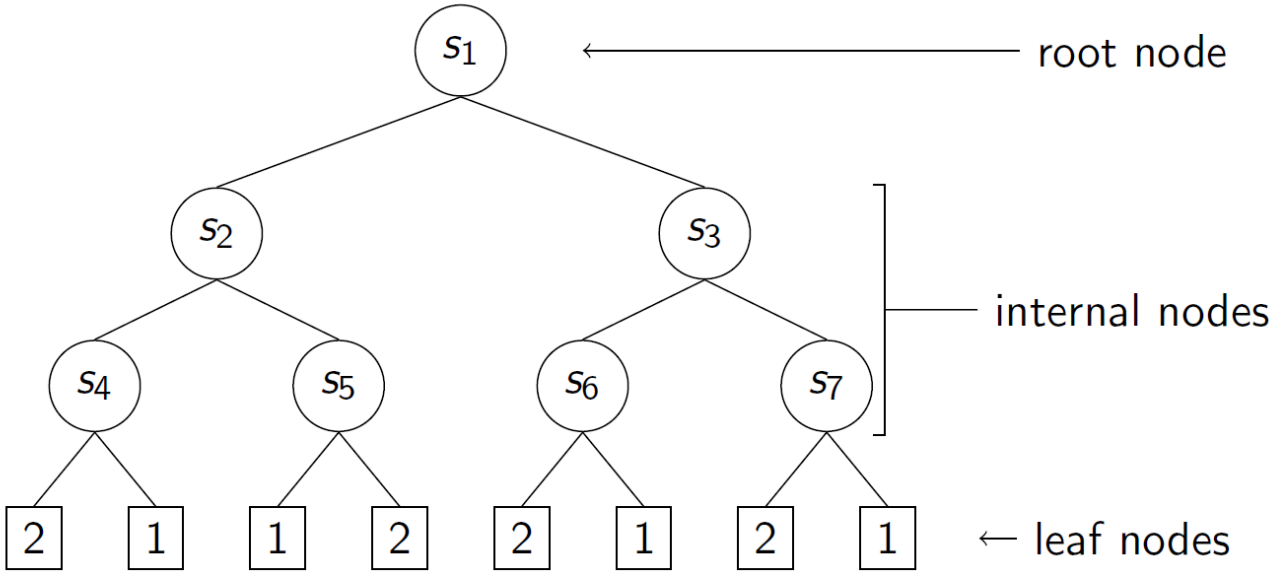
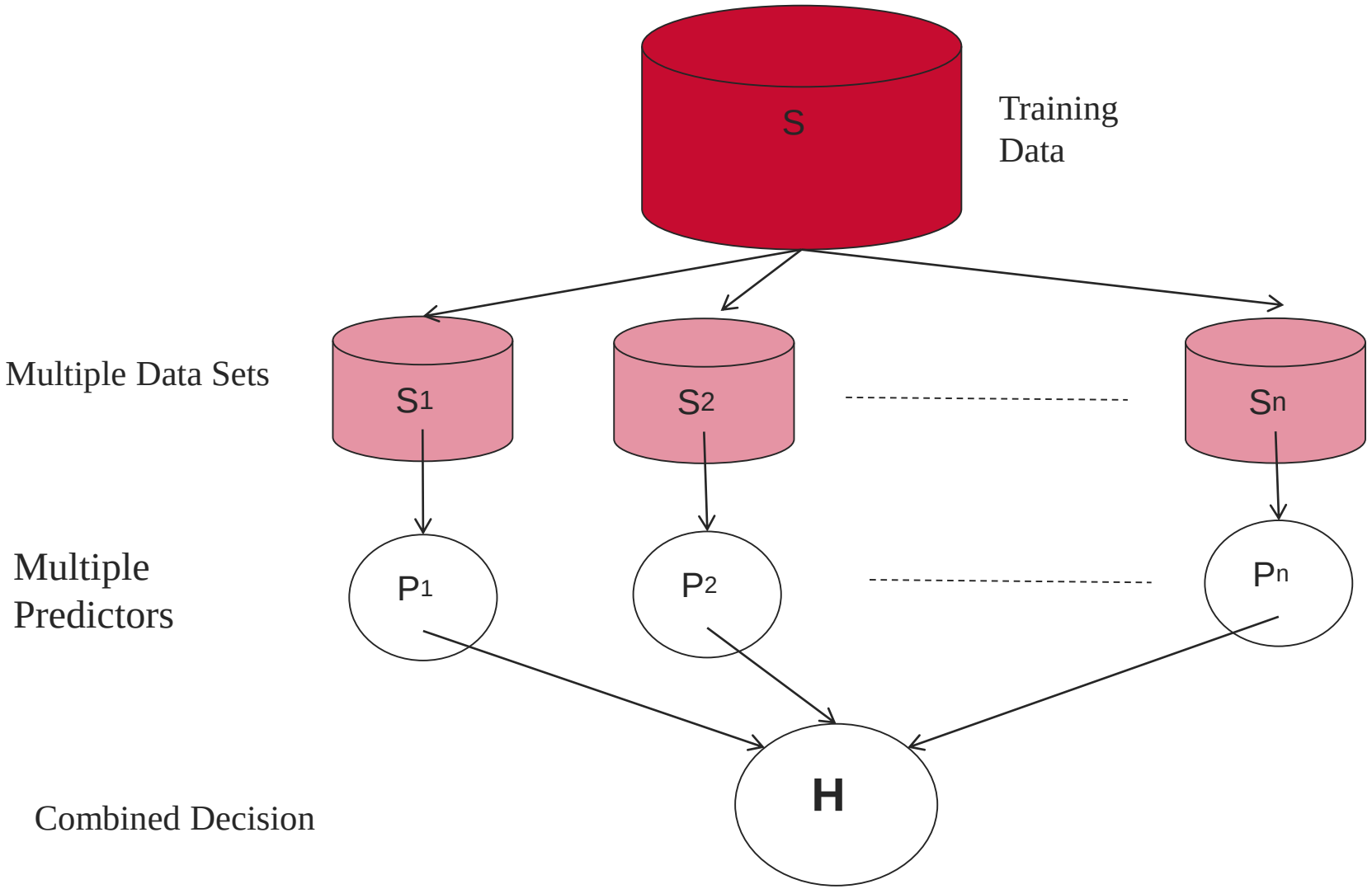


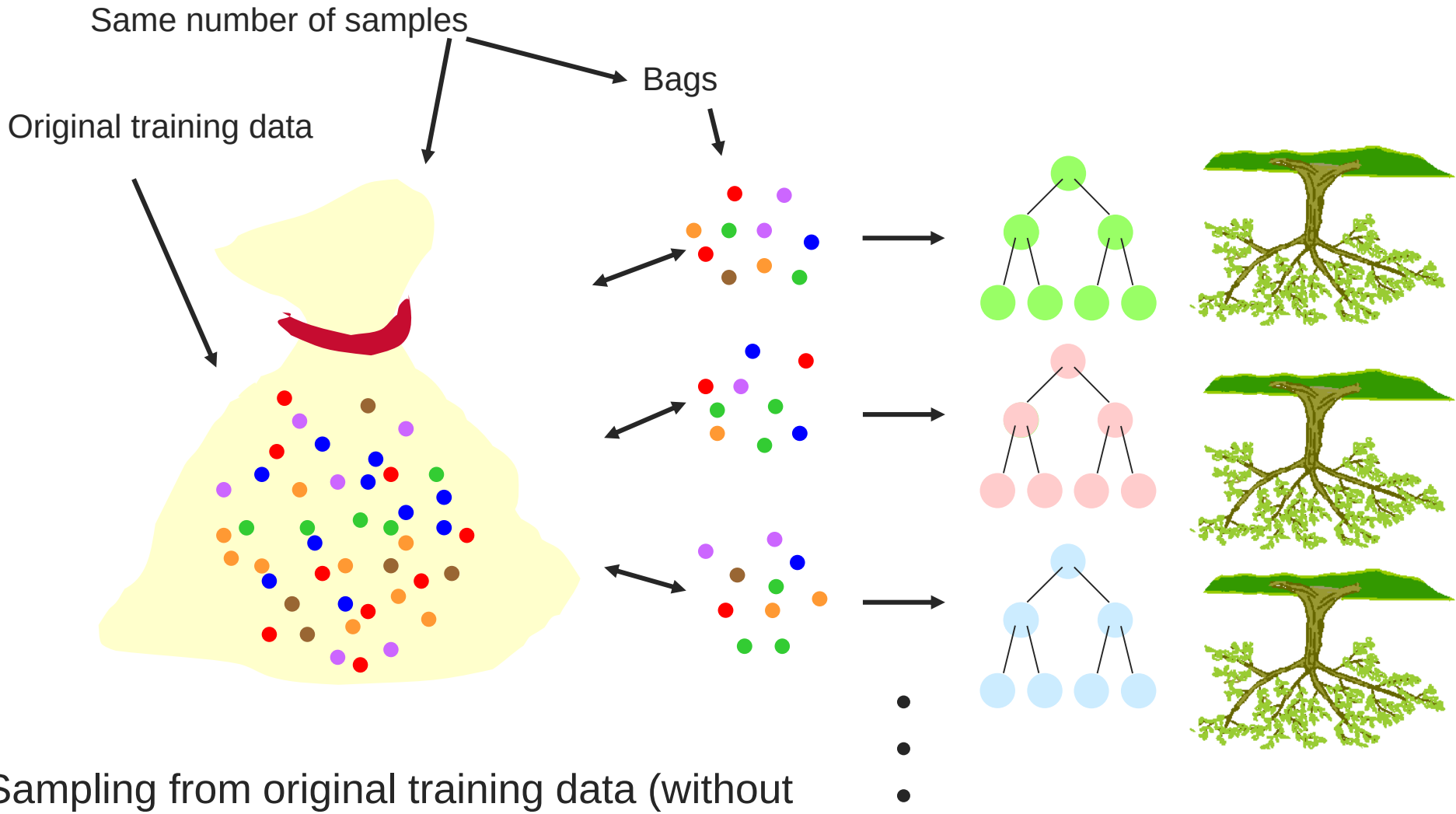
Figure: Decision Tree

- a popular machine learning algorithm
- employs recursive partitioning

General Overview



Bagging(1)



Sampling from original training data (without deletion) Usually to the same number of samples In each bag. Around 65% of distinct samples and 35% duplicates.

Bagging(2)

1. For i=1: No.Trs // No.Trs is number of trees or bags

a) Draw (with replacement) from the training set to generate the training data S_i

b) Learn a classifier C_i on S_i .

2. For each test example

a) Try all classifiers C_i

b) Predict the class that receives the highest number of votes.

Random Forests

Random Forest Algorithm:

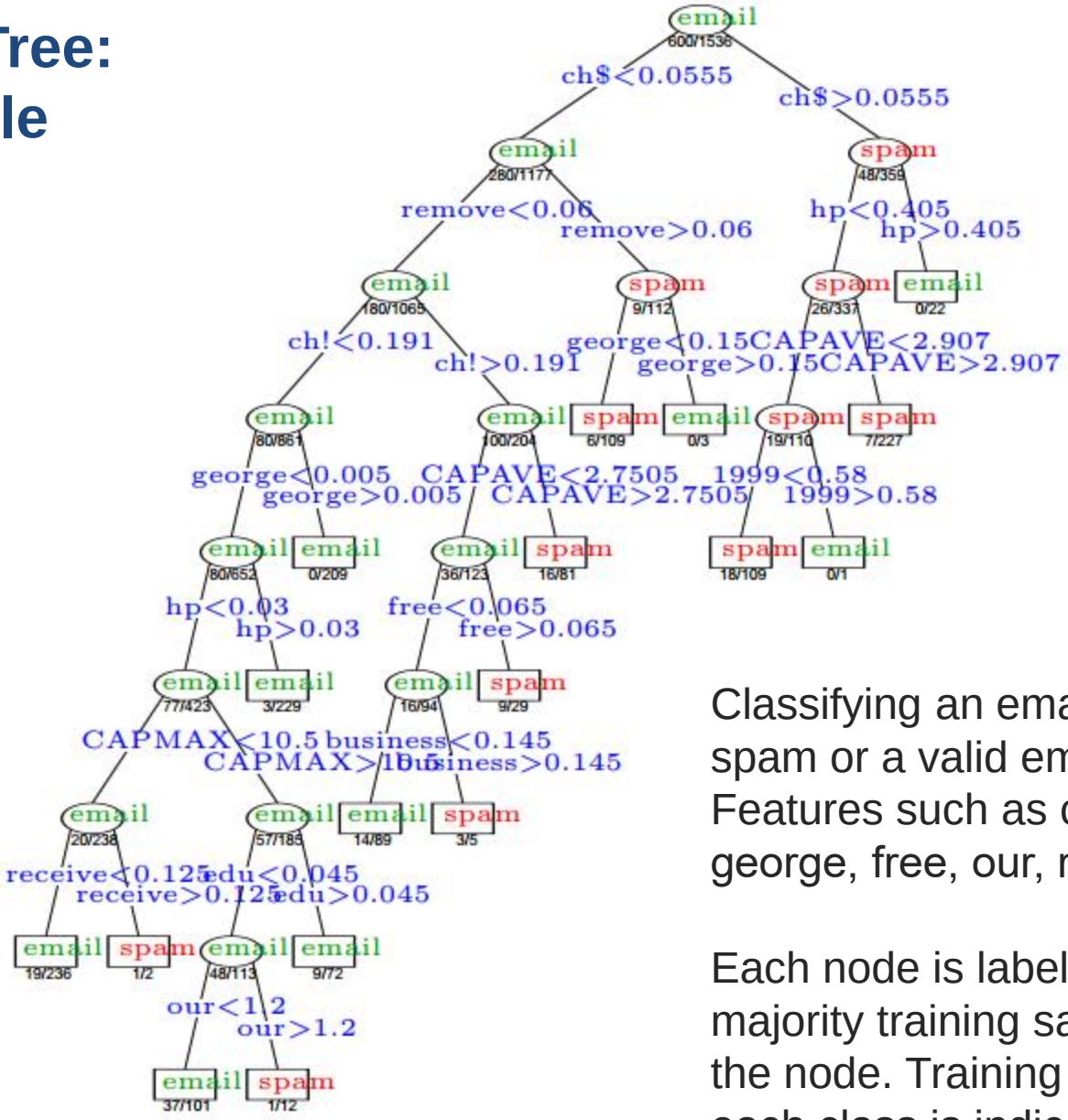
· Training phase:
Given:
$X := N \times M$ is the training dataset, where N is the number of the training data, M is the dimension of each data.
$Y := N \times 1$ is the labels of the training set.
L is the ensemble size, which means the number of trees in the forests.
T_i refers to each random tree in the forest, $i = 1...L$.
m is the number of features randomly selected to split in each non-leaf node.
$minleaf$ is the maximum number of samples in an impure terminal node.

1. Generate the training set for T_i by sampling N times from all N available training cases with replacement.
2. At each node the best split is calculated using the m randomly chosen features in the training set for T_i.
3. Go to Step 2 until T_i is fully grown until reach an pure node or the number of samples are smaller than "$minleaf$" .

· Classification phase:
For a given sample, it is pushed down each tree in the forests and each tree in the forests will give one vote on the predicted label of this sample. In this case, the predicted label of this sample is determined as the one which has the most votes in the forests.

L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.

Decision Tree: An Example



Classifying an email as either as spam or a valid email using Features such as ch! , ch$, 1999, george, free, our, receive, …

Each node is labeled based on majority training samples reaching the node. Training samples of each class is indicated as x/y below each node.

Decision Tree

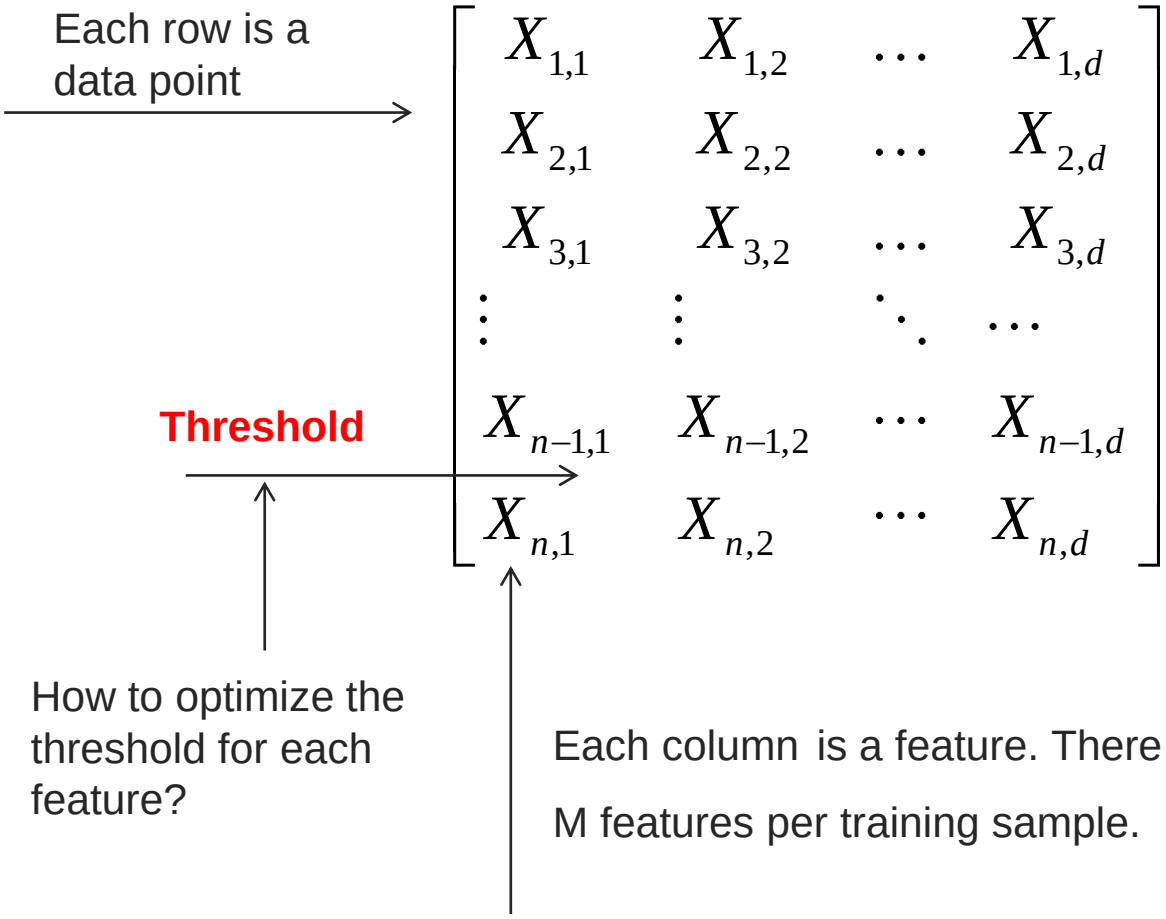
- Each internal node is associated with a test/split function defined as:

$$f(\mathrm{x},\theta) = \begin{cases} 1 \ \text{ if } \mathrm{x}(\theta_1) < \theta_2 \\ 0 \ \text{otherwise} \end{cases}$$

where $\theta_1 \epsilon \{1,2, \ldots d\}$ and $\theta_2 \epsilon \mathbb{R}$ is a threshold.

A node of decision Tree

Assume that n training samples (each with M features) reach a node.



Optimize threshold for each feature

- the threshold should be optimized so that Gini impurity is maximized: $Gini_{beforesplit} - Gini_{aftersplit}$

$$Gini(t)_{beforesplit} = 1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}}{n}\right)^2$$

Eqn (a)

$$Gini(t)_{aftersplit} = \frac{n^l}{n}\left[1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}{}^l}{n^l}\right)^2\right] + \frac{n^r}{n}\left[1 - \sum_{i=1}^{c} \left(\frac{n_{w_i}{}^r}{n^r}\right)^2\right]$$

Eqn (b)

C -total number of classes at the node
n – total samples
r - right branch
l – left branch
wi – the same as *i*.

There are other impurity criteria such as information gain, etc.

Example

Imagine that 50 training samples belonging to 5 classes (10 samples each) reaching a node. Pay attention to the squared terms in the summation in the previous slide.

Before the split: **1-** $\sum_{i=1}^{5}\left(\frac{10}{50}\right)^2$

Imagine 2 hypothetical cases after split:

(a) 40 samples belonging to first 4 classes on the right and 10 samples belonging to the last class on the left.

After Split: $\frac{10}{50}\left[1-\left(\frac{10}{10}\right)^2\right]+\frac{40}{50}\left[1-\sum_{i=1}^{4}\left(\frac{10}{40}\right)^2\right]$

(b) 5 samples belonging to each class on the left and right leaf nodes.

After Split: $\frac{25}{50}\left[1-\sum_{i=1}^{5}\left(\frac{5}{25}\right)^2\right]+\frac{25}{50}\left[1-\sum_{i=1}^{5}\left(\frac{5}{25}\right)^2\right]$

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Rotation Forest

Rotation Forest algorithm:

· Training phase:
Given:
$X := N \times m$ is the training dataset, where N is the number of the training data, m is the dimension of each data.
F is the feature set.
$Y : N \times 1$ is the labels of the training set.
L is the ensemble size, which means the number of trees in the forests.
T_i refers to each random tree in the RF, $i = 1...L$.
m is the number of subsets. $\{w_1, ..., w_c\}$ the set of class labels
For $i = 1...L$:

1. prepare the Rotation Matrix R_i
   - Split F into m subsets:$F_{i,j}$ (for $j = 1...m$)
   - For $j = 1...m$
     - Let $X_{i,j}$ be the data set X for the features in $F_{i,j}$
     - Eliminate from $X_{i,j}$ a random subset of classes
     - Select a bootstrap sample from $X_{i,j}$ of size 75% of the number of objects in $X_{i,j}$. Denote the new set by $X'_{i,j}$
     - Apply PCA on $X'_{i,j}$ to obtain the coefficients in a matrix $C_{i,j}$
   - Arrange the $C_{i,j}$ for $j = 1...m$ in a rotation matrix as in equation (1)
   - Construct ${R_i}^a$ by rearranging the columns of R_i so as to match the order of features in F.
2. Building classifier using $(X {R_i}^a, Y)$as the training set

· Classification phase:

1. For a given x, let $d_{i,j}(x{R_i}^a)$ be the probability assigned by classifier D_i to the hypothesis that x comes from class w_j. Calculate the confidence for each class, w_j, by the average combination method:

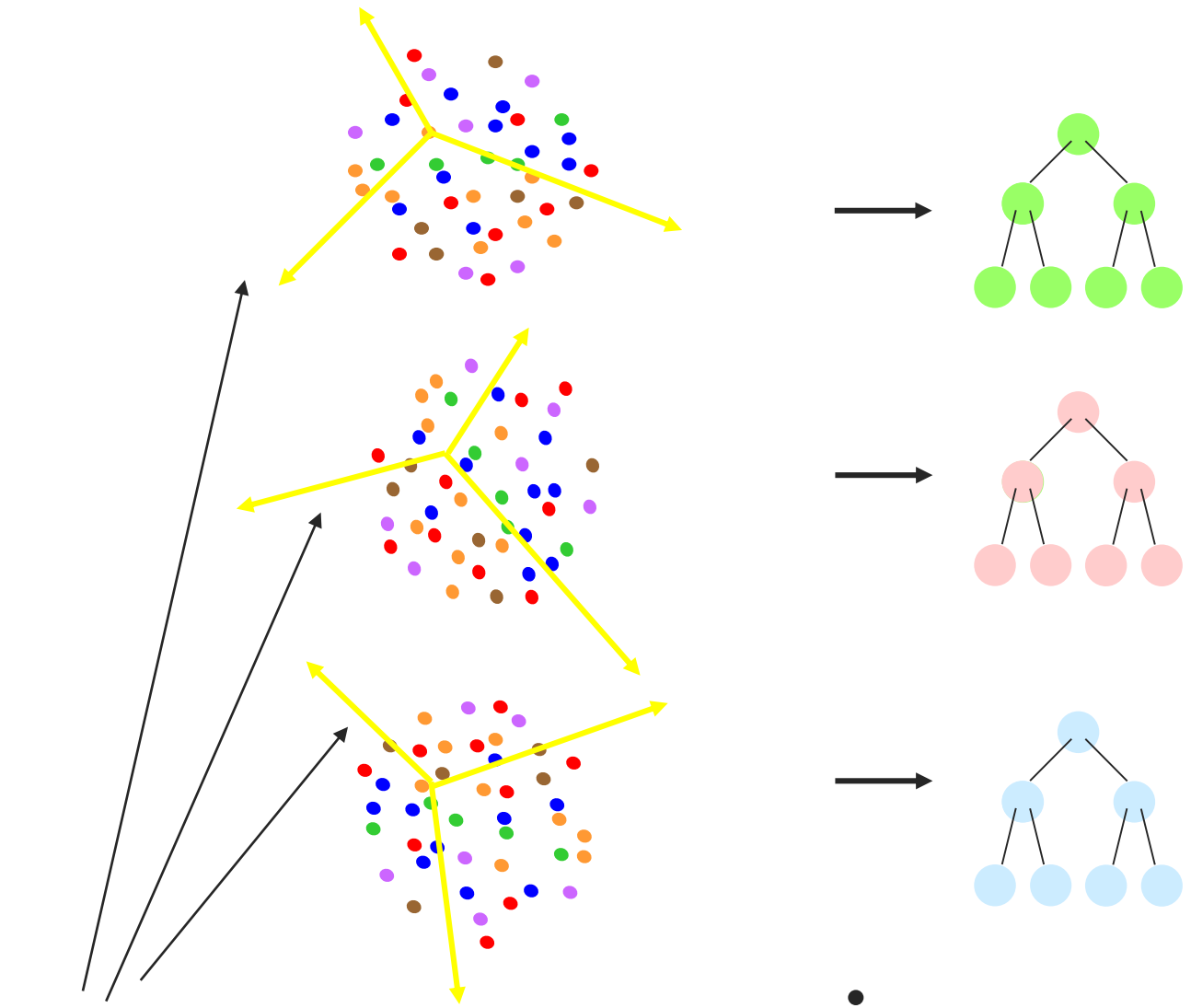
$$\mu_j(x) = \frac{1}{L}\sum_{i=1}^{n} d_{i,j}(x{R_i}^a), j = 1, ...c \tag{1.4}$$

2. Assign x to the class with the largest confidence.

Rotation Forest

- A distinct rotation matrix for each tree
- All features, all samples are rotated at the root level.
- In the default Rotation Forest version, all rotated features are used in each node.

Rotation Forest Illustrated



Different rotation applied

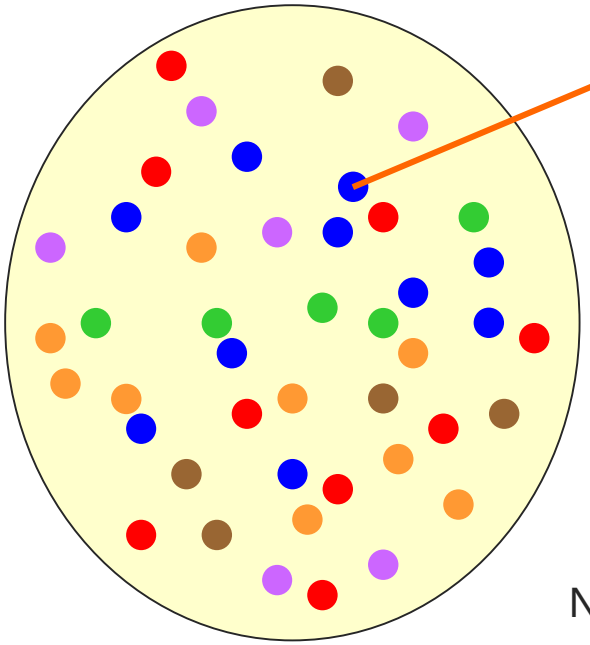
Rotation Forest

- Base classifiers: decision trees → Forest

  PCA is a simple rotation of the coordinate axes → Rotation Forest

Method for Constructing a Rotation Matrix

X: the objects in the training data set



x = [x1, x2, …, x_M]^T a data point with M features

N × M matrix

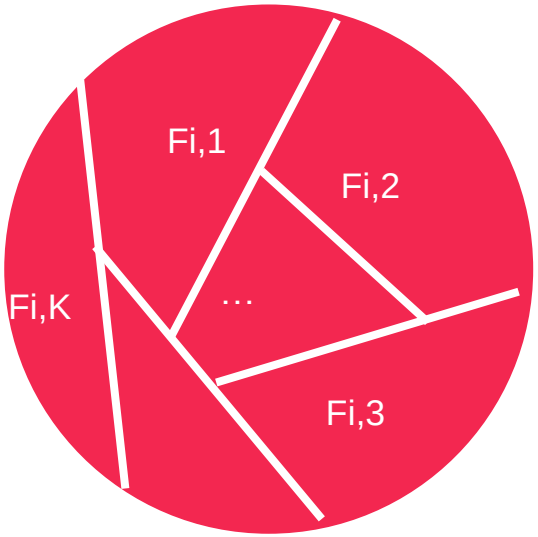
$$X = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & \cdots & x_n^1 \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ x_1^N & x_2^N & \cdots & \cdots & x_n^N \end{bmatrix}$$

Y = [y1, y2, …, y_N]^T : class labels with c classes

Method for Constructing a Rotation Matrix

- For i = 1 … L (to construct the training set for classifier D_i)

F : full feature set



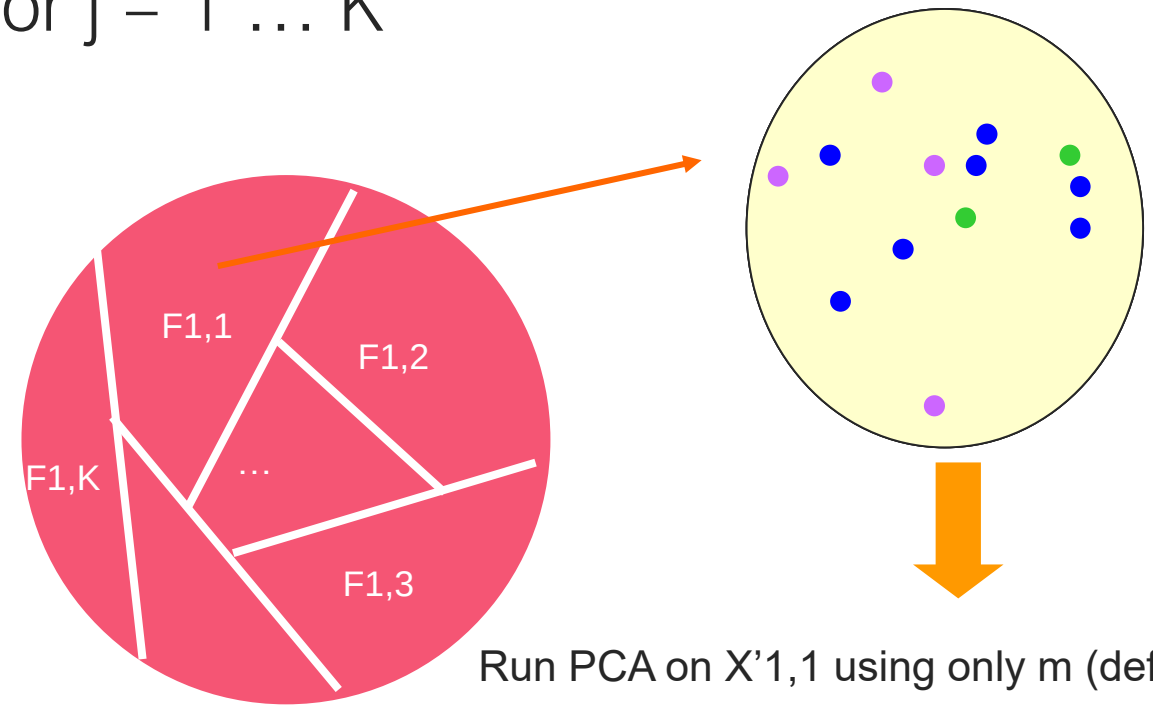
K subsets of features ($F_{i,j}$ j=1…K)

each has m = M/K features; m can be 3 as a default.

Method for Constructing Rotation Matrix

- For j = 1 … K

X1,1: data set X for the features in F1,1



Eliminate a random subset of data / classes

Select a bootstrap sample from X1,1 to obtain X'1,1

Run PCA on X'1,1 using only m (default m=3) features

Principal components $\mathbf{a}^{(1)}_{1,1},\ldots,\mathbf{a}^{(m1)}_{1,1}$

Method … (Cont'd)

$$X = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & \cdots & x_n^1 \\ \vdots & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ x_1^N & x_2^N & \cdots & \cdots & x_n^N \end{bmatrix}$$

- Arrange the principal components for all j to obtain a *rotation matrix*

$$R_1 = \begin{bmatrix} a_{1,1}^{(1)}, a_{1,1}^{(2)}, \ldots, a_{1,1}^{(M_1)} & [0] & \cdots & [0] \\ [0] & a_{1,2}^{(1)}, a_{1,2}^{(2)}, \ldots, a_{1,2}^{(M_2)} & \cdots & [0] \\ \vdots & \vdots & \ddots & \vdots \\ [0] & [0] & \cdots & a_{1,K}^{(1)}, a_{1,K}^{(2)}, \ldots, a_{1,K}^{(M_K)} \end{bmatrix}$$

- Rearrange the rows of R_1 so as to match the order of features in F to obtain R_1^a
- Build classifier D_1 using XR_1^a as a training set

Outline of RF

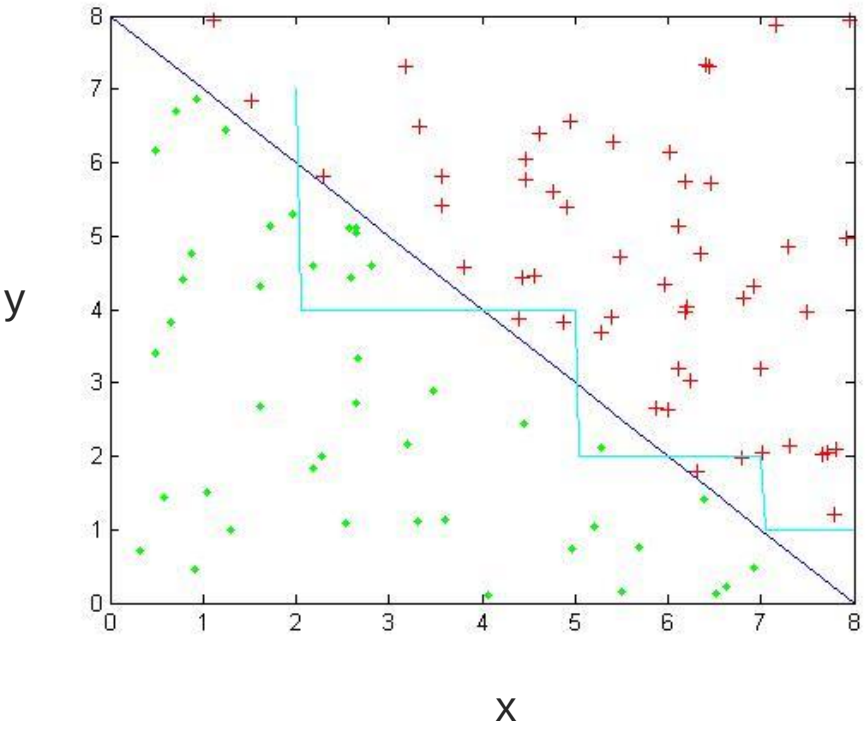
Random Forest and its Variants

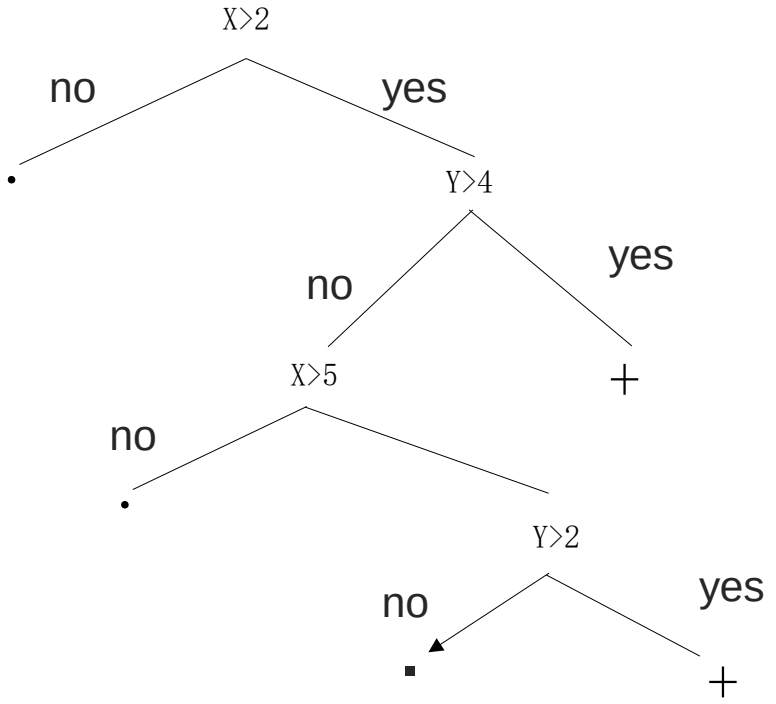
- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Hyperplane in a node of a Decision Tree

- In most case, axis-parallel hyperplane is employed in the decision tree.
- Oblique hyperplane can be better
- Axis-parallel or univariate or othorgonal
- Oblique or multi-variate

Oblique VS axis-parallel hyperplane





An example instance space,“+”“.”means different classes, the stage shows the decision boundary of univariate DT and the line shows the decision boundary of oblique DT

A Toy Example of Oblique and axis-Parallel RF

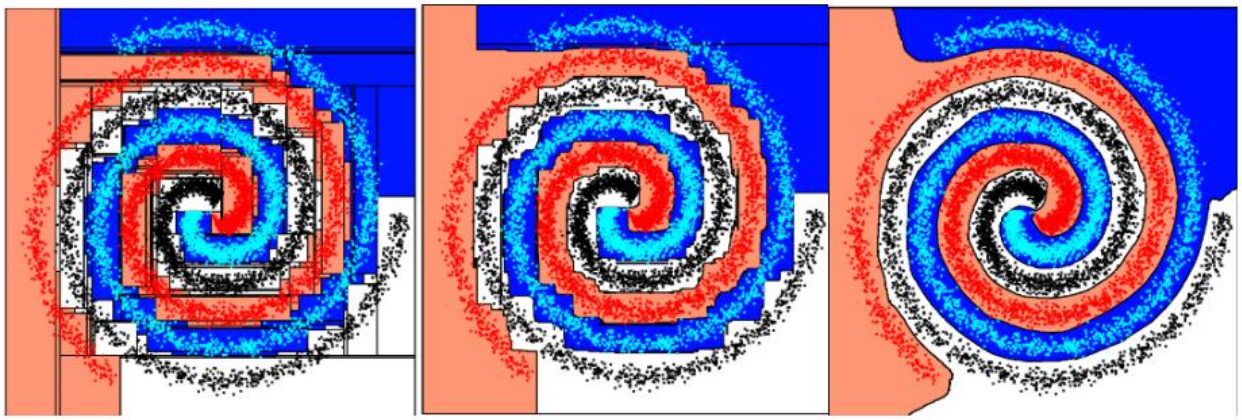


Figure: Toy example:spiral dataset. Hyperplane from left to right are generated by a single axis-parallel decision tree, conventional axis-parallel decision tree ensemble and oblique decision tree ensemble.

Oblique Decision Tree (DT) ensemble Via multisurface Proximal SVM

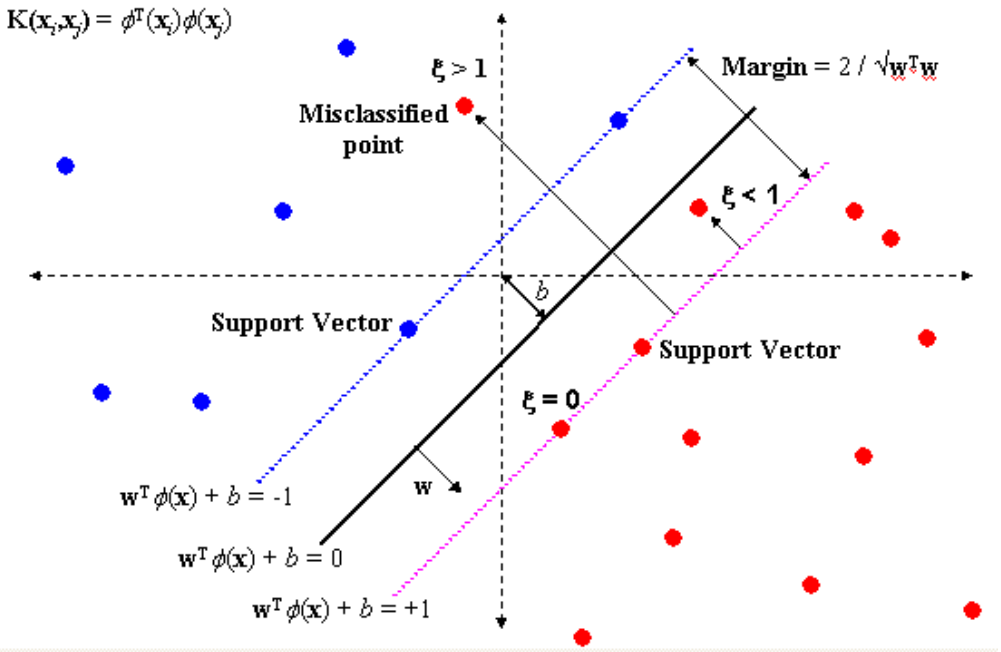
- Univariate (axis-parallel) DT: finding a split amounts to finding an feature which is the most ‘useful ‘ in discriminating the input data.
- Multivariate (oblique) DT: finding a ‘composite’ feature, a combination of the existing features that has good discriminatory power.
- Suppose there are n examples with d features. For Uni-DT, there exist only d(n-1) hyperplanes. However, the number of the oblique hyperplanes is $O(n^d)$.
- Exhaustive search which works quite good for uni-DT is impossible for Multi-DT.

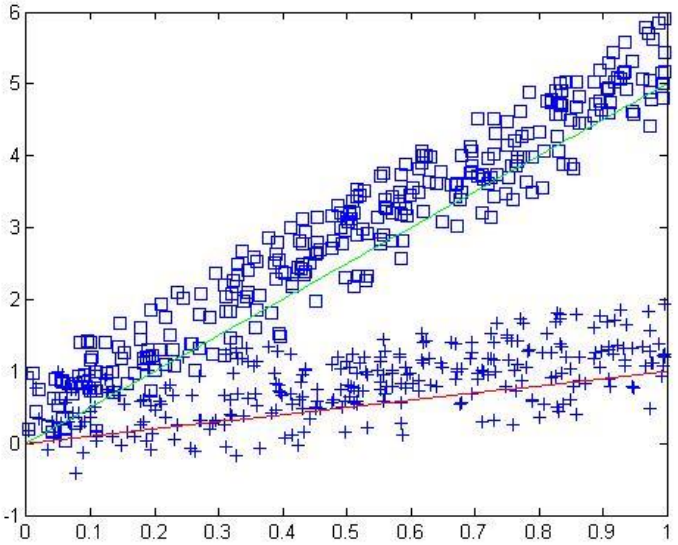
Le Zhang and P. N. Suganthan, “Oblique Decision Tree Ensemble via Multisurface Proximal Support Vector Machine,” IEEE Transactions on Cybernetics, 2015.
(Codes available online)

How to find the optimal hyperplane in each internal node

- Heuristic search: hill-climbing, simulated annealing, Genetic algorithm
- Time-consuming, usually lead to sub-optimal solutions.

SVM and its Variant: Multisurface Proximal SVM

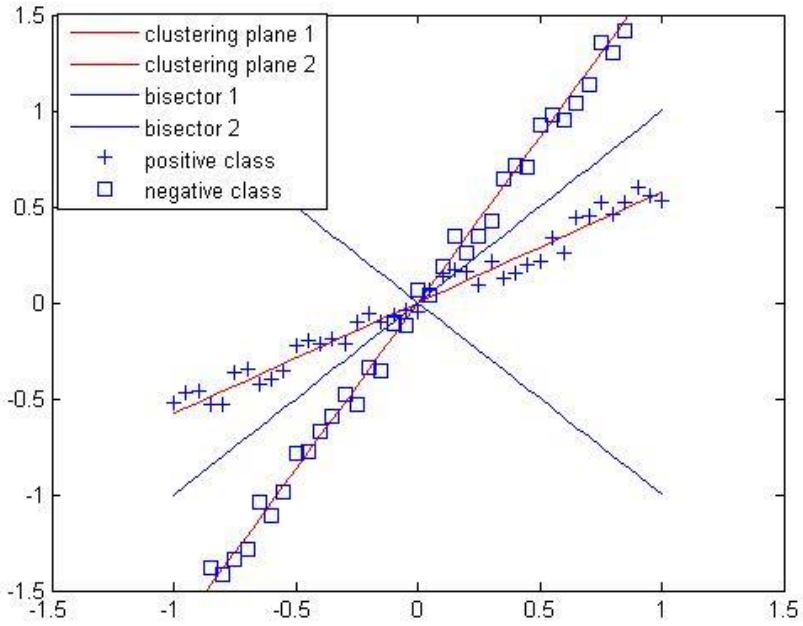




SVM finds two parallel hyperplanes that divide the feature spaces into three disjoint parts. The data lies in between these two hyperplanes are linearly inseparable. SVM classifies data by assigning them to one of the remaining two disjoint half-spaces.

MP-SVM aims to find two clustering hyperplanes, where the first plane is closest to the class 1 data & farthest from class 2 and the second plane is closest to class2 and furthest from class 1.

Obtain the optimal hyperplane from MP-SVM



- An example of how to get the optimal hyperplane from the two clustering hyperlanes. The two red planes are the clustering planes and the two blue planes are the two angle bisectors of the two clustering planes. Here we can choose the bisector to act as a test. The data above the plane goes to one child node and the others goes to the other child node. Since there are two bisectors, we can choose the one which has better discriminant ability. Here the discriminant ability can be defined as some criteria such as info-gain or GINI impurity.

MPSVM

- The first and the second clustering hyperplanes (*W* below) can be obtained by solving the following two equations, where A represents the matrix (each row is a data sample) of the first class and B stands for the matrix of the second class, e is a vector of ones with the same dimension as AW and BW.

$$\min_{(W,b)\neq 0} \frac{\| AW - eb \|^2 / \left| \begin{matrix} W \\ b \end{matrix} \right|^2}{\| BW - eb \|^2 / \left| \begin{matrix} W \\ b \end{matrix} \right|^2}$$

$$\min_{(W,b)\neq 0} \frac{\| BW - eb \|^2 / \left| \begin{matrix} W \\ b \end{matrix} \right|^2}{\| AW - eb \|^2 / \left| \begin{matrix} W \\ b \end{matrix} \right|^2}$$

MPSVM

- By defining

  G=[A -e]'[A -e], H=[B -e]'[B -e]

  Z=[W b]' the two problems become:

$$\min_{Z\neq 0} \frac{Z^T GZ}{Z^T HZ}, \qquad \min_{Z\neq 0} \frac{Z^T \mathrm{H}Z}{Z^T \mathrm{G}Z}$$

- Thus, the two clustering hyperplanes can be found by the eigenvectors corresponding to the smallest and largest eigenvalues of the following generalized eigenvalue problem:

$$Gz = \lambda Hz, z \neq 0$$

MPSVM based decision tree

- How to handle the multiclass problem since the MPSVM can only solve the binary classification problems.

- As the tree grows, the number of samples reaching lower nodes will be less and less but the matrix G and H have the size of $(M + 1) \times (M + 1)$ always, So, G and H may become singular.

- (One approach would be to use feature subspace to solve singularity problem and to reduce computation time)

MPSVM based decision tree

multi2binary algorithm

Input:

$X := N \times m$ is the training dataset, where N is the number of training samples, m is the dimension of each data.

$Y := N \times 1$ are the labels of the training set.

$\{w_1, ..., w_c\}$ the set of class labels

Output:

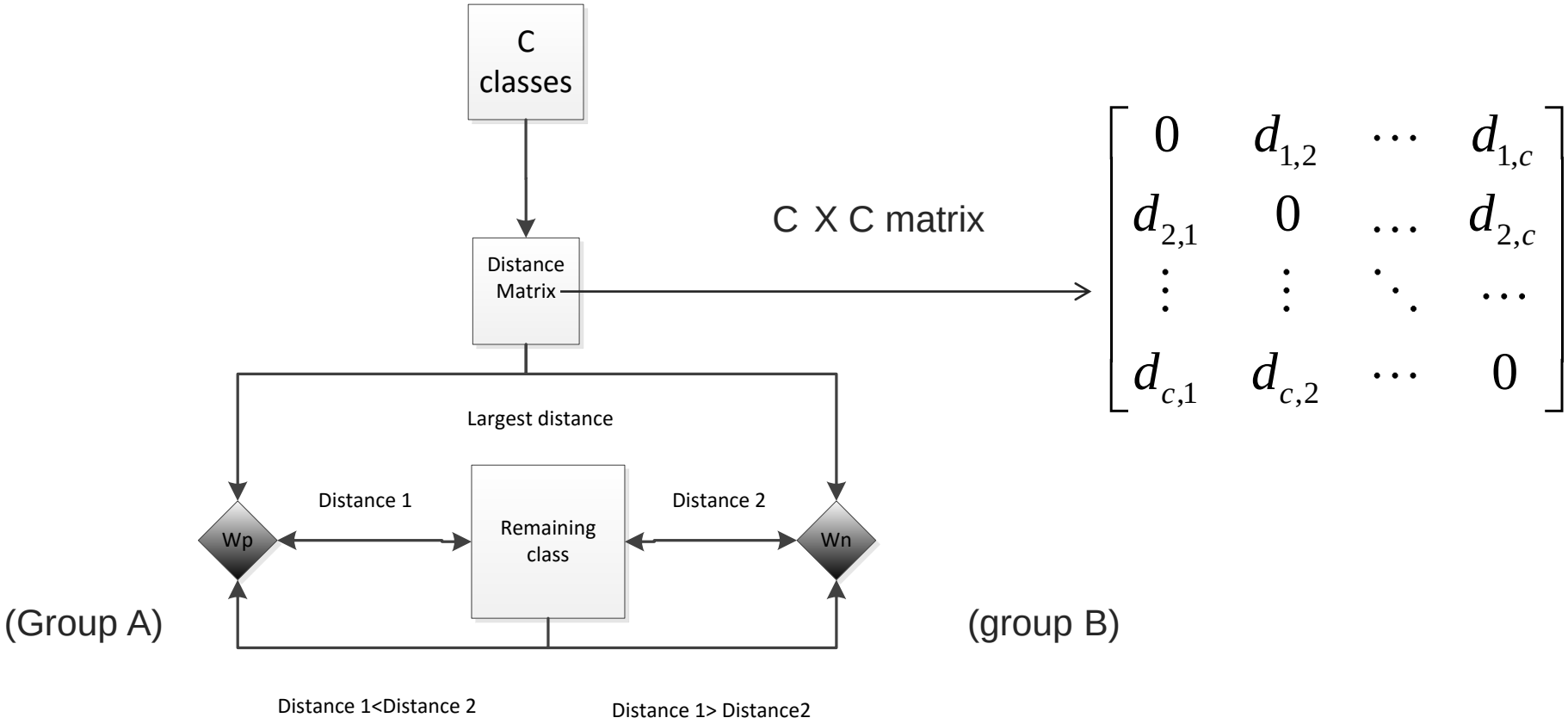
W_p and W_n are the two groups (hyper-classes).

- for $i = 1, ...c$
  - obtain the pairwise Bhattacharyya distance between class w_i and w_k for $k = i+1, ..., c$
- find the pair of classes with the largest Bhattacharyya distance, name them as w_p and w_n and put them in group W_p and W_n respectively.
- for every remaining class, say w_k assign it to the group W_p if $B(w_k, w_p) < B(w_k, w_n)$, otherwise assign it to the group W_n

$$B(w_1, w_2) = \frac{1}{8}(\mu_2 - \mu_1)^T \left(\frac{\Sigma_1 + \Sigma_2}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} \ln \frac{\left| (\Sigma_2 + \Sigma_1)/2 \right|}{\sqrt{|\Sigma_1|}\sqrt{|\Sigma_2|}}$$

the Bhattacharyya distance measures the similarity between two discrete or continuous probability distributions,which is considered a good measure for class separability between two normal classes $W_1 \sim N(\mu_1, \Sigma_1)$ and $W_2 \sim N(\mu_2, \Sigma_2)$.

Multiclass problem



Methods to handle S3 problem

- NULL space method[1-2] is very sensitive to the data perturbations.
- Here, we apply two regularization approaches: Tikhonov regularization and axis-parallel split regularization.

[1] X. Jiang, “Linear subspace learning-based dimensionality reduction,” Signal Processing Magazine, IEEE, vol. 28, no. 2, pp. 16–26, 2011.
[2] X. Jiang et al, “Eigenfeature regularization and extraction in face recognition,” TPAMI, vol. 30, no. 3, pp. 383–394, 2008.

Methods to handle S3

- Tikhonov regularization works by adding a constant term to the diagonal entries of the matrix to be regularized. In our case, suppose G becomes rank deficient, G is regularized by:

$$G' = G + \delta * I$$

Methods to handle S3

- If the matrix of G or H becomes singular at a node, we can always use the axis-parallel split to continue the tree induction process. Hence, the decision tree grows using heterogeneous test functions. From the root node to the current node, the decision tree uses the MPSVM to perform splits. From the current node to the leaf node, the decision tree switches to use the axis-parallel splitting method.

Experiment result

abbreviation	full name
RaF	Random Forest
RoF	Rotation Forest
RRoF	Random Rotation Forest
MPSVM	Multisurface Proximal SVM
MPRaF-T	MPSVM based RaF with Tikhonov Regularization
MPRaF-P	MPSVM based RaF with axis-parallel Regularization
MPRaF-N	MPSVM based RaF with NULL Space Regularization
MPRoF-T	MPSVM based RoF with Tikhonov Regularization
MPRoF-P	MPSVM based RoF with axis-parallel Regularization
MPRoF-N	MPSVM based RoF with NULL Space Regularization
MPRRoF-T	MPSVM based RRoF with Tikhonov Regularization
MPRRoF-P	MPSVM based RRoF with axis-parallel Regularization
MPRRoF-N	MPSVM based RRoF with NULL Space Regularization

Datasets

Le Zhang and P. N. Suganthan, "Oblique Decision Tree Ensemble via Multisurface Proximal Support Vector Machine," IEEE Transactions on Cybernetics, 2015. (Codes available online)

Datasets	Classes	Samples	Features
Adult	2	32561	123
AFP-Pred	2	9974	119
Australian	2	690	14
Balance scale	3	625	4
Banknote	2	1372	4
Biodeg	2	1055	41
Blood transfusion	2	748	4
BLProt	2	18943	545
Breast cancer	2	699	9
Breast tissue	6	106	9
Climate model	2	540	18
DNA	3	3186	180
Ecoli	8	336	7
Fertility	2	100	9
Glass	7	214	9
Harberman	2	306	3
Heart	2	217	13
Hepatitis	2	155	13
segment	7	2310	19
Ionosphere	2	351	34
Iris 3	150	4	
Lymphography	4	296	18
Mam-masses	2	961	5
Orl	40	400	1024
Ozone	2	2536	72
Page block	5	5473	10
Parkinsons	2	195	22
Pima-diabetes	2	768	8
Planning Relax	2	182	12
Seeds	3	210	7
Sonar	2	208	60
Spambase	2	4601	57
Statlog-segment	7	2310	19
Teaching assistant	3	151	5
Twonorm	2	7400	20
User Knowledge	4	258	5
Vehicle	4	846	18
Vertebral-2C	2	310	6
Vertebral-3C	3	310	6
Waveform1	3	5000	21
Waveform2	3	5000	40
Wine	3	178	13
Wine quality(red)	6	1599	11
Yale	15	165	1024

Experiments

- Experimental settings:

  1. Rotation Forest, and Random Forest were kept at their default values in WEKA

  2. for Rotation Forest, M is fixed to be 3

  3. all ensemble methods have the same ensemble size 50.

  4. base DT classifier: CART (Breiman), Classification and Regression Tree.

  5. database: UCI Machine Learning Repository, Bioinformatic dataset, Face recognition dataset

  6. 10 times 3-fold cross validation.

Experimental Result: RaF

Datasets	RaF	MPRaF-T	MPRaF-P	MPRaF-N
Adult	**84.37**	83.40	84.33	82.06
AFP-pred	74.00	73.43	72.78	**78.93**
Australian	**87.01**	86.42	86.90	86.25
Balance scale	84.77	89.02	88.19	**89.10**
Banknote	99.17	**99.91**	**99.91**	99.89
Biodeg	86.31	**86.68**	86.52	85.78
Blood transfusion	77.09	**78.05**	77.49	77.50
BLProt	81.22	81.27	**83.85**	70.40
W-breast cancer	96.45	96.72	96.72	**96.92**
Breast tissue	68.01	**68.87**	67.55	60.94
Climate model	**92.28**	92.06	91.85	91.59
DNA	**94.24**	91.30	93.92	80.07
Ecoli	84.55	**85.46**	84.76	84.40
Fertility	87.70	87.90	**88.20**	88.00
Glass	**96.92**	94.21	96.64	59.67
Harberman	70.39	**72.39**	70.29	71.34
Heart	81.93	**83.74**	82.96	83.07
Hepatitis	82.97	83.61	**83.94**	79.48
segment	95.33	94.42	**95.51**	89.55
Ionosphere	93.62	92.68	**93.96**	89.74
Iris	94.60	**97.60**	95.67	97.40
Lymphography	**93.10**	95.37	93.01	89.26
Mam-masses	82.09	82.24	81.93	**82.38**
Orl	77.15	**94.10**	88.20	93.45
Ozone	97.07	97.12	97.10	**97.13**
Page Block	97.33	97.28	**97.35**	95.37
Parkinsons	90.21	**91.23**	90.82	86.36
Pima-diabetes	75.81	75.91	75.98	**76.77**
Planning relax	69.62	70.49	70.66	**71.21**
Seeds	92.14	**94.19**	93.14	92.43
Sonar	79.13	**82.12**	78.99	81.44
Spambase	94.60	93.68	**94.68**	93.77
statlog-segment	97.38	97.13	**97.48**	94.89
Teaching assistant	54.37	55.10	**55.56**	53.38
Twonorm	96.51	97.47	**97.51**	97.45
User Knowledge	91.51	91.20	**91.82**	90.15
Vehicle	74.86	76.30	**76.34**	70.76
Vertebral-2C	82.58	**84.61**	84.58	84.10
Vertebral-3C	83.35	83.61	**83.77**	82.48
Waveform1	84.34	**85.70**	85.50	85.70
Waveform2	84.47	**85.44**	85.32	85.34
Wine	97.30	97.64	**98.26**	96.24
Wine quality(red)	66.62	**67.22**	67.10	59.42
Yale	56.44	**80.00**	64.56	78.22

	MPRaF-T	MPRaF-P	MPRaF-N
win-tie-lose	32-0-12	33-0-11	22-0-22

	RaF	MPRaF-T	MPRaF-P	MPRaF-N
mean accuracy	84.80	86.24	85.72	83.63

"MPRaF-T", "MPRaF-P", "MPRaF-N" means MPSVM based Oblique Random Forest with Tikhonov, axis-parallel split, and NULL space regularization, respectively. The "win-tie-lose" value means the number of times that this method wins, ties and loses to RaF, respectively. "mean accuracy" stands for the overall accuracy across all the datasets for each method.

MPRaF-P: Multisurface Proximal Random Forest with axis parallel regularization.

Experimental Result: RoF

Datasets	RoF	MPRoF-T	MPRoF-P	MPRoF-I
Adult	81.83	83.55	81.83	**83.81**
AFP-pred	79.79	64.16	**79.96**	61.09
Australian	**85.58**	80.71	83.30	81.91
Balance scale	86.90	**88.00**	87.81	83.98
Banknote	**99.92**	99.74	99.85	99.83
Biodeg	86.87	81.92	**87.08**	**86.03**
Blood transfusion	74.80	73.30	75.03	**76.32**
BLProt	81.20	68.70	**81.29**	70.81
W-breast cancer	**96.47**	93.73	96.01	96.05
Breast tissue	**67.36**	54.53	66.42	49.06
Climate model	92.24	90.91	**92.59**	91.98
DNA	93.72	90.81	93.68	**94.41**
Ecoli	84.55	79.32	**84.73**	82.59
Fertility	**86.10**	78.80	85.70	85.10
Glass	97.57	93.36	**97.76**	46.82
Harberman	**68.30**	67.75	66.96	67.81
Heart	**81.30**	76.26	78.78	79.96
Hepatitis	**80.90**	76.97	80.39	79.35
segment	**95.08**	86.00	94.98	66.72
Ionosphere	94.30	84.37	**94.47**	85.76
Iris	94.73	95.00	**95.33**	94.93
Lymphography	94.19	92.91	**94.32**	91.62
Mam-masses	77.39	76.87	76.74	**79.49**
Orl	89.90	84.50	**90.00**	81.00
Ozone	97.08	96.26	**97.13**	96.34
Page Block	**97.22**	95.98	97.08	95.76
Parkinsons	90.10	81.69	**90.41**	75.64
Pima-diabetes	**74.44**	71.05	72.02	73.26
Planning relax	67.86	58.35	63.90	**66.32**
Seeds	**92.86**	91.14	92.05	88.86
Sonar	**82.26**	73.89	77.16	71.49
Spambase	**94.94**	89.52	94.52	93.88
statlog-segment	98.12	92.63	**98.20**	81.27
Teaching assistant	**57.35**	55.63	57.22	57.28
Twonorm	**97.49**	95.93	97.39	97.17
User Knowledge	91.32	87.36	**92.64**	91.40
Vehicle	77.62	77.04	**79.69**	68.96
Vertebral-2C	85.23	79.81	**85.52**	77.23
Vertebral-3C	84.54	78.19	**84.65**	75.03
Waveform1	**85.61**	80.63	85.06	84.83
Waveform2	**85.30**	80.53	85.02	84.71
Wine	94.10	92.36	**94.55**	92.25
Wine quality(red)	66.74	57.10	**67.03**	53.42
Yale	73.56	**78.89**	74.78	76.67

	MPRoF-T	MPRoF-P	MPRoF-N
win-tie-lose	4-0-40	21-0-23	9-0-35

	RoF	MPRoF-T	MPRoF-P	MPRoF-N
mean accuracy	85.58	81.28	85.25	79.95

"MPRoF-T", "MPRoF-P", "MPRoF-N" means MPSVM based Oblique Rotation Forest with Tikhonov, axis-parallel split, and NULL space regularization, respectively. The "win-tie-lose" value means the number of times that this method wins, ties and loses to RoF, respectively. "mean accuracy" stands for the overall accuracy across all the datasets for each method.

MPRoF-T: Multisurface Proximal Rotation Forest with Tikhonov regularization

Discussion

- RaF

	MPRaF-T	MPRaF-P	MPRaF-N
win-tie-lose	32-0-12	33-0-11	22-0-22

- RoF

	MPRoF-T	MPRoF-P	MPRoF-N
win-tie-lose	4-0-40	21-0-23	9-0-35

$$\min_{Z \neq 0} \frac{Z^T G Z}{Z^T H Z}$$

Why MPSVM works for RaF, but not for RoF?

For RaF, H is (m+1)X(m+1)
At each root node.

For default RoF, H is (M+1)X(M+1) with m~=sqrt(M).

Random Rotation Forest

- In order to solve this problem, we propose to employ random feature subspace in the base learner of Rotation Forest and name it as Random Rotation Forest (RRoF). In each node, the test function is evaluated on a randomly selected sub feature set instead of the whole feature set.
- In this case, Random Forest and Rotation Forests differ in the way that they perturb the data: Random Forest uses bagging to create a data subset and Rotation Forests employs different rotation matrices for different tree.

Experiment Result: RRoF

Datasets	RRoF	MPRRoF-T	MPRRoF-P	MPRRoF-N
Adult	83.31	**83.95**	83.38	81.91
AFP-pred	76.41	73.16	74.11	**77.56**
Australian	86.42	86.42	**86.61**	86.49
Balance scale	86.66	89.30	88.70	**89.81**
Banknote	99.88	**100.00**	**100.00**	**100.00**
Biodeg	86.92	87.18	86.65	**87.55**
Blood transfusion	75.76	77.59	76.14	**77.63**
BLProt	82.19	81.92	**82.57**	66.94
W-breast cancer	96.74	96.39	96.67	**96.80**
Breast tissue	69.53	68.87	**70.19**	65.85
Climate model	92.15	**92.30**	91.26	91.76
DNA	**94.38**	91.29	93.68	90.88
Ecoli	85.51	85.24	**85.80**	84.38
Fertility	86.90	88.00	87.70	**88.10**
Glass	96.68	94.95	**97.43**	69.86
Harberman	68.86	**71.21**	70.13	69.67
Heart	81.85	83.11	82.37	**83.59**
Hepatitis	82.45	**84.71**	84.13	83.74
segment	95.75	95.62	**96.01**	94.31
Ionosphere	94.36	93.39	**94.70**	90.77
Iris	94.40	**96.47**	95.40	96.33
Lymphography	95.03	**95.41**	95.20	93.99
Mam-masses	78.68	**81.48**	78.61	81.38
Orl	89.40	**94.55**	91.95	85.85
Ozone	97.10	**97.12**	97.10	97.10
Page Block	97.22	97.20	**97.24**	96.78
Parkinsons	91.33	**92.15**	**92.15**	88.41
Pima-diabetes	75.51	74.87	75.26	**75.81**
Planning relax	70.04	**71.26**	71.10	71.14
Seeds	93.19	94.71	**95.00**	93.62
Sonar	81.54	**83.46**	80.77	83.17
Spambase	**95.08**	94.51	95.04	94.42
statlog-segment	97.93	97.88	**98.04**	97.24
Teaching assistant	56.42	54.70	56.29	**57.02**
Twonorm	97.34	97.52	**97.56**	97.47
User Knowledge	89.19	88.91	89.42	**89.65**
Vehicle	77.25	77.02	**77.81**	72.85
Vertebral-2C	85.19	85.26	**85.45**	84.97
Vertebral-3C	84.90	85.10	**85.06**	84.26
Waveform1	85.66	85.67	85.63	**86.00**
Waveform2	85.53	85.53	**85.74**	85.70
Wine	97.02	97.98	**98.09**	97.36
Wine quality(red)	67.08	66.90	**67.50**	60.46
Yale	72.11	**82.33**	75.11	78.89

	MPRRoF-T	MPRRoF-P	MPRRoF-N
win-tie-lose	25-2-17	33-0-11	24-0-20

	RRoF	MPRRoF-T	MPRRoF-P	MPRRoF-N
mean accuracy	85.84	86.42	86.27	84.72

"MPRRoF-T", "MPRRoF-P", "MPRRoF-N" means MPSVM based Random Rotation Forest (subspace involved in each node) with Tikhonov, axis-parallel split, and NULL space regularization, respectively. The "win-tie-lose" value means the number of times that this method wins, ties and loses to RRoF, respectively. "mean accuracy" stands for the overall accuracy across all the datasets for each method.

MPRRoF-N: Multisurface Proximal Random Rotation Forest with Null space regularization

RaF vs RoF vs RRoF

- In order to find out the statistical significance of the results, we carry out a Friedman test. It ranks the algorithms for each data set separately with the best performing algorithm getting the lowest rank. Let r_i^j be the rank of the jth of k algorithms on the ith of N data sets. The Friedman test compares the average ranks of algorithms R_j. Under the null-hypothesis, which states that all the algorithms are equivalent and so their ranks R_j should be equal, the Friedman statistic

$$X_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right]$$

is distributed according to X_F^2 with k − 1 degrees of freedom, when N and k are big enough. In that case, Friedman's X_F^2 is undesirably conservative and derived a better statistic:

$$F_F = \frac{(N-1)X_F^2}{N(k-1)-X_F^2}$$

which is the F-distribution with k − 1 and (k − 1)(N − 1) degrees of freedom [1].

[1]. J. Demˇsar, "Statistical comparisons of classifiers over multiple data sets," JMLR, vol. 7, pp. 1–30, 2006.

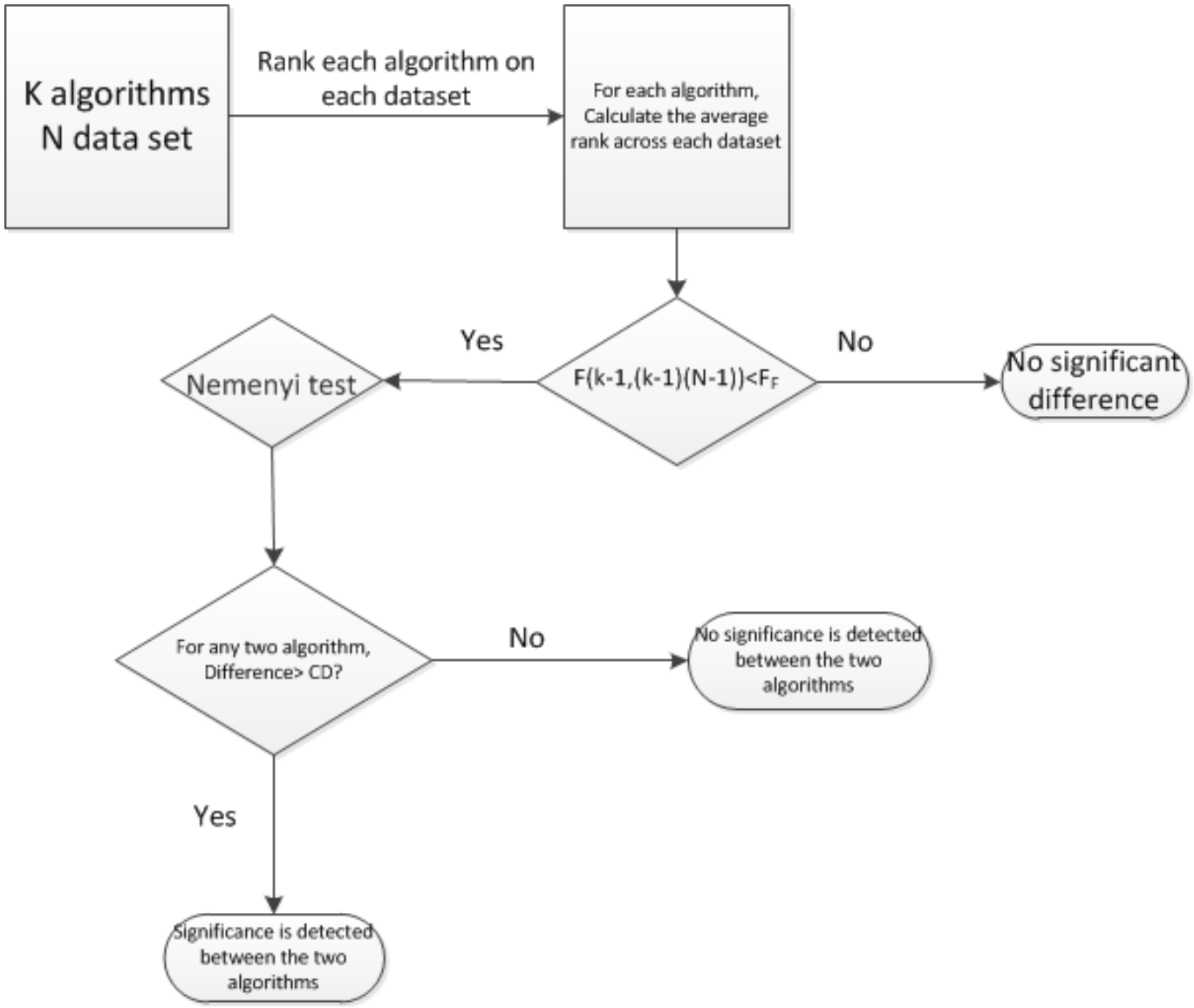
RaF vs RoF vs RRoF

- If the null-hypothesis is rejected, the Nemenyi test (Nemenyi,1963) can be used to check whether the performance of two among k classifiers is significantly different. If the corresponding average ranks differ by at least the critical difference

$$CD = q_\alpha \sqrt{\frac{(k+1)k}{6N}}$$

we say there is a significant difference between 2 classifiers.

RaF vs RoF vs RRoF



RaF vs RoF vs RRoF

- The average ranks for RaF, RoF and RRoF are 2.24, 2.07, 1.69, respectively. Then X_F^2 =6.98 F_F =3.71 .The critical value of F(2,86) for 0.05 is ~2.72, so we reject the null-hypothesis.
- In this case, CD=0.5.

Which regularization works better ?

Method	RaF(2.95)	MPRaF-T (1.97)	MRPaF-P(2.11)	MPRaF-N(2.97)
RaF(2.95)				
MPRaF-T(1.97)	√			√
MPRaF-P(2.11)	√			√
MPRaF-N(2.97)				

Method	RRoF(2.91)	MPRRoF-T (2.19)	MPRRoF-P(2.15)	MPRRoF-N(2.75)
RRoF(2.91)				
MPRRoF-T(2.19)	√			
MPRRoF-P(2.15)	√			
MPRRoF-N(2.75)				

The numbers in the bracket represent the average rank for the algorithm, √ means the method of the row is significantly better than the method of of the column. Empty entry means there is no significant difference between the method of row and the method of the column.

For a given regularization approach, which ensemble method is better?

- we use Sign Test to compare each pair of algorithms.
- If the two algorithms compared are, as assumed under the null-hypothesis, equivalent, each should win on approximately N/2 out of N data sets: if the number of wins is at least N/2 + 1.96sqrt(N)/2, the algorithm is significantly better with $p < 0.05$

For a given regularization approach, which ensemble method is better?

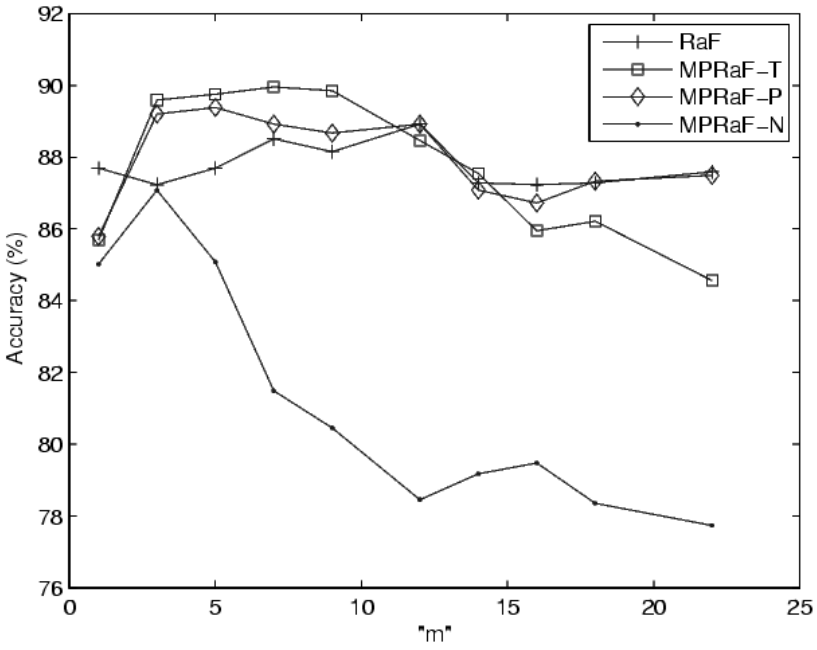
Method	Significance
(MPRaF-T,MPRRoF-T) (18,26)	
(MPRaF-P,MPRRoF-P) (16,28)	
(MPRaF-N,MPRRoF-N) (12,32)	√

The first number in the bracket represents the number of times RaF wins, the second number means the number of times RRoF wins. √ means there is significant difference between this pair of algorithms.

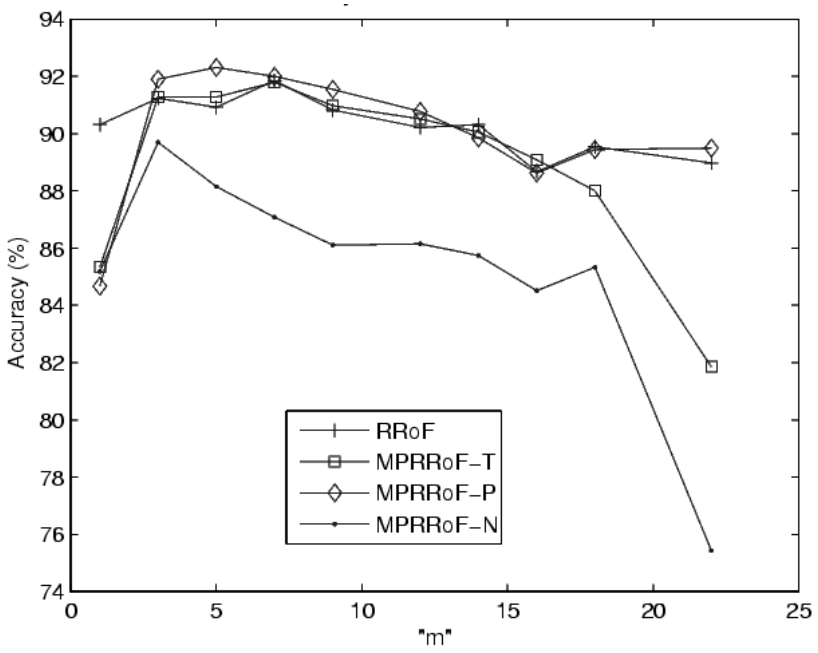
On the effect of m

- The parameter m denotes the number of features randomly selected at each node
- the smaller m is, the stronger the randomization of the trees and the weaker the dependence of their structures on the output
- However, if m is small, the features randomly selected at a node may fail to capture the geometry of the data samples

The effect of m for “ parkinsons” dataset

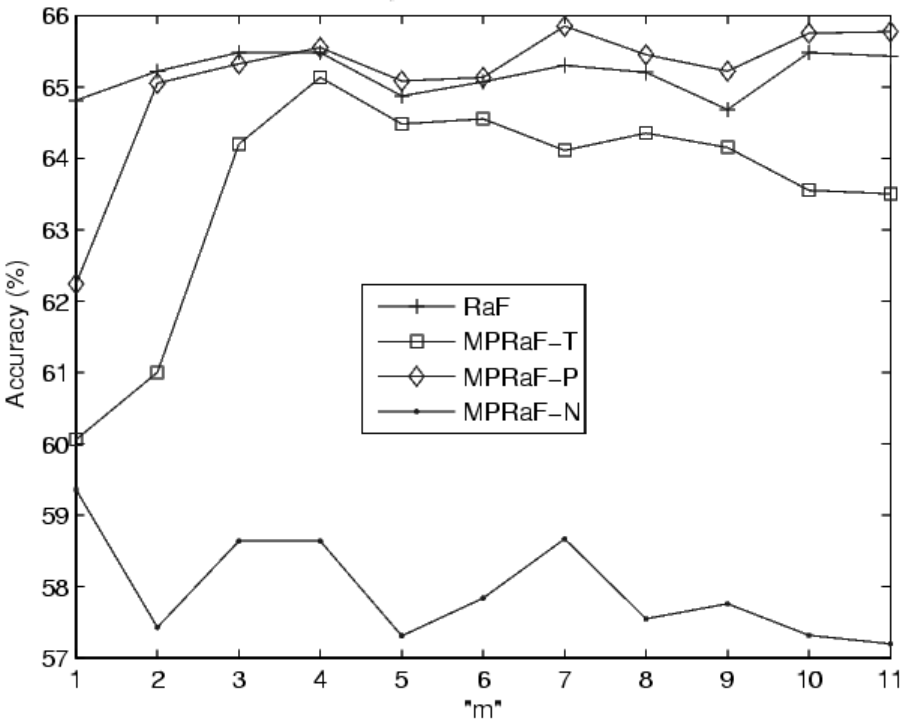


RaF

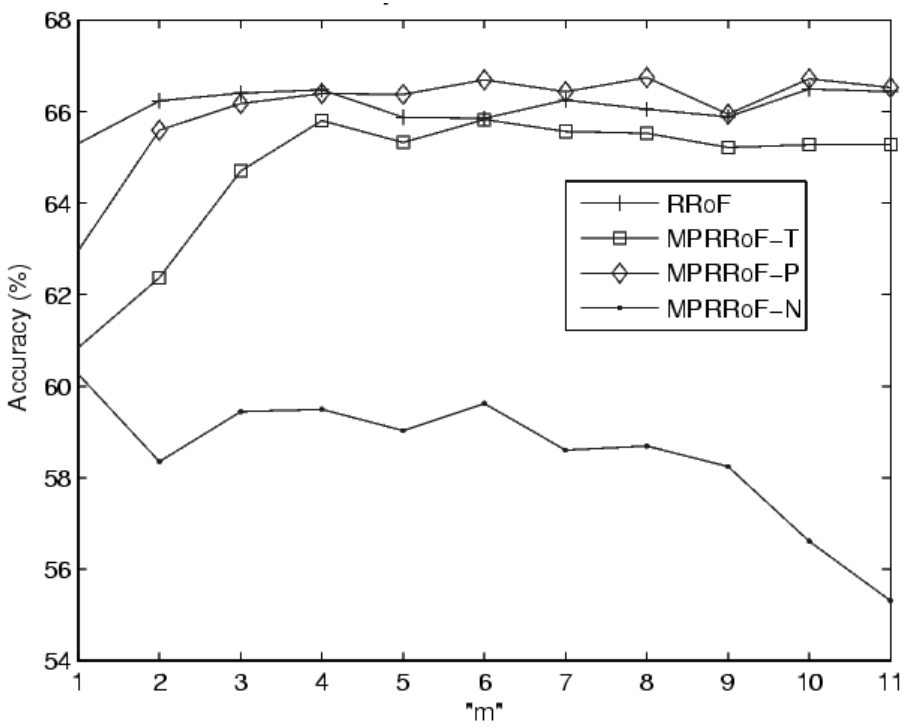


RRoF

The effect of m for “ wine quality (Wine)” dataset



RaF



RRoF

Discussion about the parameter m

- For very small value of m, the accuracy of all the ensemble methods are very low, especially for the MPSVM based ensembles

- However, as the m grows, the accuracies of all MPSVM based ensembles grow significantly and become stable very quickly except for the MPSVM with NULL space regularization.

Comprehensive Evaluation of MPSVM-Based Oblique RFs Using JMLR Protocol

- We choose MPRaF-P, MPRRoF-P and compare with other 179 classifier on 121 UCI dataset.
- 500 bags / trees in the forest.

Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The Journal of Machine Learning Research* 15.1 (2014): 3133-3181.

L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, Nov 2017

Overall performance compared with JMLR Results[1]

Top 10 classifiers

Method	Friedman Rank	Mean accuracy
MPRaF-P	33.10	82.05
CoKRR-max	34.13	81.79
MPRRoF-P	34.56	82.35
parRF_t	34.97	81.96
rf_t	35.16	82.30
svm_C	36.78	81.77
svmPoly_t	40.36	80.31
KRR / elm_kernel_m	41.65	81.52
Rforest_R	41.67	81.94
svmRadialCost_r	42.47	80.54

Top 11-20 classifier

Method	Friedman Rank	Mean Accuracy
RVFL Ensemble	44.27	80.94
svmRadial_t	44.78	80.17
C5.0_t	45.22	80.56
avNNet_t	46.54	79.37
nnet_t	47.99	79.52
BG_LibSVM_w	49.44	80.77
pcaNNet_t	49.61	78.65
mlp_t	49.82	80.25
RotationForest_w	50.20	80.61
RRF_t	52.64	80.92

Our proposed methods in blue bold. Other results copied from the JMLR reference:

Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *The Journal of Machine Learning Research* 15.1 (2014): 3133-3181.

Strengths & Weaknesses of RF

Strengths

- Hard to overfit as the whole training data is usually not used collectively and non-existence of a global objective function
- Easy to parallelize (except boosting)
- Excellent batch mode classification performance

Weakness

- Online learning is tough or not so competitive
- Not an effective feature extractor
- Not so friendly for transfer learning, deep learning, etc.
- Not the best for forecasting (based on our experiences)

Comparing with Deep NN: Self-Normalizing Neural Networks (SNNs)

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. In *Advances in neural information processing systems* (pp. 971-980).

- Unlike Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), success stories of Deep Learning with standard feed-forward neural networks (FNNs) are rare.

- The activation function of SNNs are "scaled exponential linear units" (SELUs), which induce self-normalizing properties.

$$\text{selu}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha e^x - \alpha & \text{if } x \leqslant 0 \end{cases}$$

α and λ are the hyperparameters.
Default values: α = 1.6733 and λ = 1.0507

- "MSRAinit": FNNs without normalization and with ReLU activations and "Microsoft weight initialization"
- "BatchNorm": FNNs with batch normalization
- "LayerNorm": FNNs with layer normalization
- "WeightNorm": FNNs with weight Normalization
- "Highway": Highway networks
- "ResNet": Residual networks adapted to FNNs using residual blocks with 2 or 3 layers with rectangular or diavolo shape.

Table 1: **Left:** Comparison of seven FNNs on 121 UCI tasks. We consider the average rank difference to rank 4, which is the average rank of seven methods with random predictions. The first column gives the method, the second the average rank difference, and the last the p-value of a paired Wilcoxon test whether the difference to the best performing method is significant. SNNs significantly outperform all other methods. **Right:** Comparison of 24 machine learning methods (ML) on the UCI datasets with more than 1000 data points. The first column gives the method, the second the average rank difference to rank 12.5, and the last the p-value of a paired Wilcoxon test whether the difference to the best performing method is significant. Methods that were significantly worse than the best method are marked with "*". SNNs outperform all competing methods.

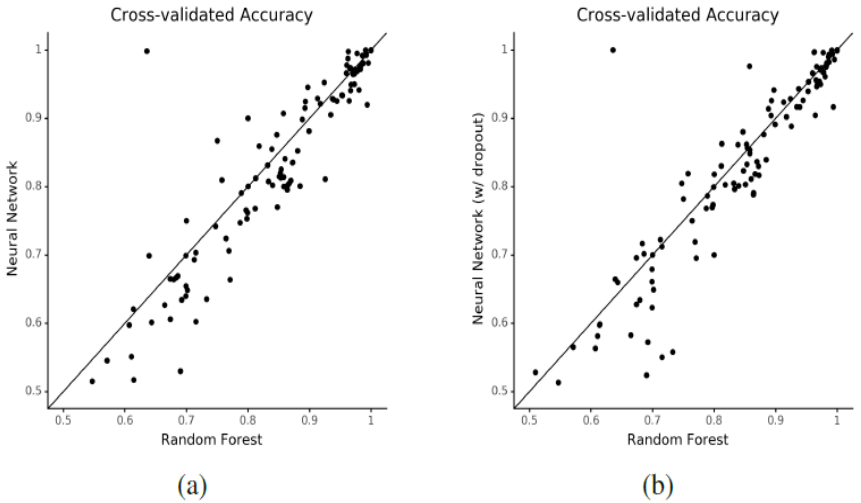
FNN method comparison			ML method comparison		
Method	avg. rank diff.	p-value	Method	avg. rank diff.	p-value
SNN	-0.756		SNN	-6.7	
MSRAinit	-0.240*	2.7e-02	SVM	-6.4	5.8e-01
LayerNorm	-0.198*	1.5e-02	RandomForest	-5.9	2.1e-01
Highway	0.021*	1.9e-03	MSRAinit	-5.4*	4.5e-03
ResNet	0.273*	5.4e-04	LayerNorm	-5.3	7.1e-02
WeightNorm	0.397*	7.8e-07	Highway	-4.6*	1.7e-03
BatchNorm	0.504*	3.5e-06	...	...	...

We further included 17 machine learning methods representing diverse method groups [9] in the comparison and the grouped the data sets into "small" and "large" data sets (for details see Supplementary Section S4.2). On 75 small datasets with less than 1000 data points, random forests and SVMs outperform SNNs and other FNNs. On 46 larger datasets with at least 1000 data points, SNNs show the highest performance followed by SVMs and random forests (see right panel of Table 1, for complete results see Supplementary Tables S9 and S10). Overall, SNNs have outperformed state of the art machine learning methods on UCI datasets with more than 1,000 data points.

Modern Neural Networks Generalize on Small Data Sets

Olson, M., Wyner, A., & Berk, R. (2018). Modern neural networks generalize on small data sets. In *Advances in Neural Information Processing Systems* (pp. 3619-3628).

Comparison on 116 UCI datasets



- 10 hidden layers
- 100 nodes per layer
- 200 epochs of gradient descent using Adam optimizer with a learning rate of 0.001 [15].
- He-initialization for each hidden layer [12]
- Elu activation function [8].

Each random forest was fit with 500 trees, using defaults known to work well in practice [6]. In particular, in each training instance, the number of randomly chosen splits to consider at each tree node was $\sqrt{p}$, where p is the number of input variables.

Figure 3: Plots of cross-validated accuracy. Each point corresponds to a data set.

Turning to the second plot in Figure 3, we see that dropout improves the performance of the neural network relative to the random forest. The mean difference between classifiers is now decreased to 1.5%, which is still significant at the 0.01 level.

We turn first to Figure 3, which plots the cross-validated accuracy of the neural network classifiers and the random forest for each data set. In the first figure, we see that a random forest outperforms an unregularized neural network on 72 out of 116 data sets, although by a small margin. The mean difference in accuracy is 2.4%, which is statistically significant at the 0.01 level by a Wilcoxon signed rank test.

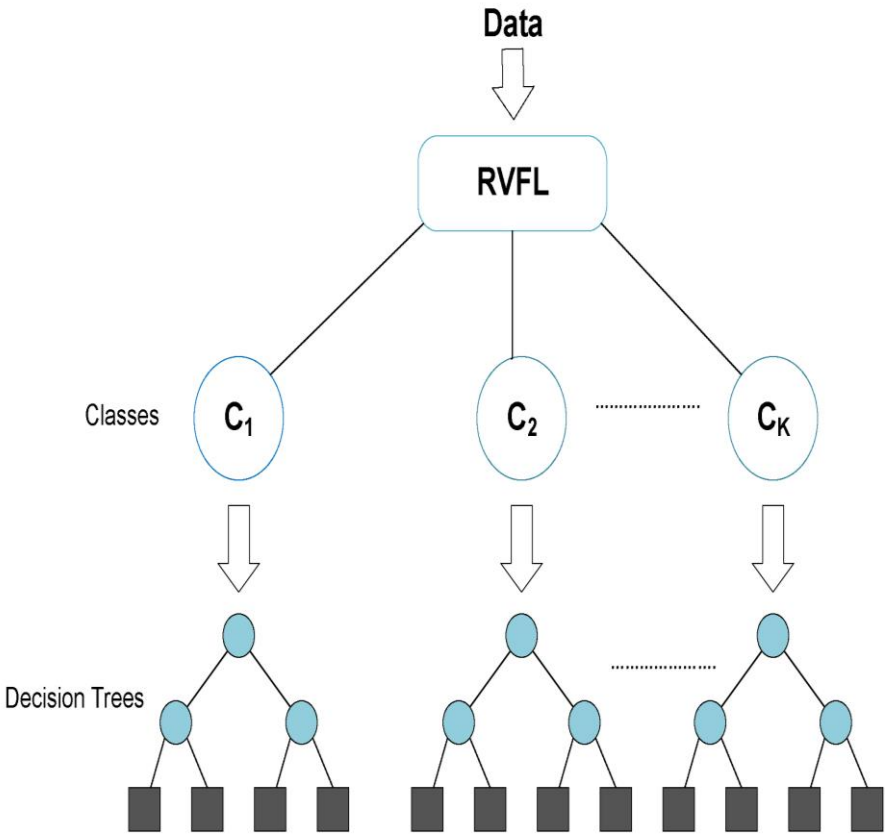
Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Ensemble of Decision Trees with RVFL for multi-class classification

- Decision trees & Random Forest: a 2-way (binary) split at each node
- Our proposed ensemble: A multi-way split at the root node using RVFL
  - ➢ Divide the data into 'K' partitions (K: the number of classes) utilizing the output scores of RVFL.
  - ➢ In each partition/subset, majority of the samples are from one class and the rest from other classes. The samples from other classes -> 'hard' to classify by RVFL
  - ➢ Grow a decision tree thereafter for each partition.



Algorithm:

Training phase:

1. For $j = 1, \ldots, M$
   (a) Generate T_j by drawing N bootstrapped samples from the training data.
   (b) Construct a RVFL network using T_j.
   (c) For $i = 1, \ldots, N$
      i. For each data x_i, select the classes with the highest and the second highest scores and store x_i as the training data for the two decision trees belonging to those two classes.
   (d) Grow a decision tree for each partition without pruning.

Testing phase:

In the testing phase, a test data is first passed to RVFL. Similar to the training phase, two classes with the highest and the second highest scores are selected based on the output of RVFL. The test data is pushed down in the decision trees of those classes. The final prediction is based on the majority votes of the decision trees.

Reference: Rakesh Katuwal, P.N. Suganthan, Le Zhang, An ensemble of decision trees with random vector functional link networks for multi-class classification, Applied Soft Computing, 2018.

69

Simulation

- 65 multi-class UCI data sets
- Ensemble size: $\mathbf{500}$
- RVFL Configuration
  - Number of hidden neurons, N = 3:203
  - λ $(= (1/2)^C)$ in ridge regression, C = -5:14
  - Activation Functions: radbas, sine and tribas
  - Range of the randomization for weights $[-S, +S]$ and bias $[0, S]$, where $S = 2^t$ with $t = -1.5: 0.5: 1.5$

- Decision Trees Configuration
  - m_{try} (number of features selected randomly at each node) → $\sqrt{n}$
  - minleaf = 1

Experimental Results

- The classifiers are ranked for each dataset separately, with the best performing algorithm getting the rank of 1, the second best rank 2,..,average ranks are assigned in case of ties.

Table: Average rank of each method

	RF	RFL (Proposed)	obRF	obRFL (Proposed)	RVFL
Rank	2.79	2.66	3.12	2.79	3.62

Lower rank reflects better performance.

RF -> Random Forest [1]
RFL -> RVFL+RF [3]
obRF -> Oblique Random Forest [2]
obRFL -> RVFL+obRF [3]
RVFL -> Random Vector Functional Link Network [4]

Table: Average rank values of 4 methods in datasets with $N > 500$

	RF	RFL (Proposed)	obRF	obRFL (Proposed)
Rank	2.55	2.09	3.01	2.33

Statistical comparison of classifiers

- Sign test: Out of N data sets, if the number of wins is at least $\frac{N}{2} + 1.96\sqrt{N}/2$, the algorithm is significantly better with $p < 0.05$.

Table: Significance of the difference between each pairs of algorithms

Method	Significance
(RF, RFL)(13,24)	✓
(RF, obRF)(23,14)	
(RF, obRFL)(17,19)	
(RFL, obRF)(25,12)	✓
(RFL, obRFL)(21,15)	
(obRF, obRFL)(10,26)	✓

The numbers in the bracket represent the number of times each method wins.

Conclusion

- Ideal for parallelization or distributed computation. Also, in agreement with Ockham's razor principle of building smaller trees.

- Improved the performance of random forest with multi-way splits. On 65 multi-class data sets from UCI repository, our proposed method RFL and obRFL achieved the best rank.

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Heterogeneous Oblique Random Forest

(submitted to Pattern Recognition)

- Decision trees in random forests (RaF) use a single feature in non-leaf nodes to split the data. Such splitting results in axis-parallel decision boundaries which may fail to exploit the geometric structure in the data.

- In oblique decision trees (obDT), an oblique hyperplane is employed instead of an axis-parallel hyperplane. Trees with such hyperplanes can better exploit the geometric structure to increase the accuracy of the trees and reduce the depth.

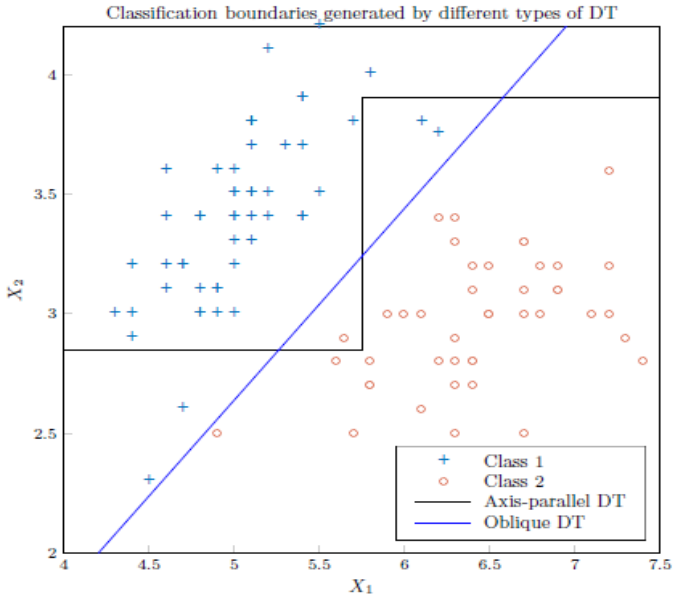


Fig: Classification boundaries generated by axis-parallel and oblique decision tree in a toy example for binary classification problem.

- A candidate hyperplane at each internal node of the decision tree is usually rated based on an impurity criterion. The impurity criterion gives higher rating to a hyperplane which results in higher purity of child nodes.

- Impurity criterion commonly used in decision trees such as gini index, information gain and towing rule are not differentiable with respect to hyperplane parameters. So, decision trees use some greedy search techniques for finding the best hyperplane at each node. The decision trees in random forest exhaustively search for a single feature or the best axis-parallel hyperplane.

- However, such exhaustive search for the best oblique hyperplane is computationally expensive. The present realizations of oblique decision trees do not evaluate many promising oblique splits to select the best.

- We propose a random forest of heterogeneous oblique decision trees that employ several linear classifiers at each non-leaf node on some top ranked partitions which are obtained via one-vs-all and two-hyperclasses based approaches and ranked based on ideal Gini scores and cluster separability.

- The oblique hyperplane that optimizes the impurity criterion is selected as the splitting hyperplane for each non-leaf node.

- The recursive partitioning of the training data while optimizing some impurity criterion aid random forest to generalize better and by the virtue of subspace and ensemble method, RaF is able to achieve state-of-the-art performances.

- Employing linear classifiers in each internal node can lead to both “accurate” and “diverse” decision trees which is of vital importance for the success of random forests. However, employing a single type of linear classifier may not always be optimal due to the No-Free-Lunch theorem.

- Such a limitation can be curbed by employing a simple strategy: selecting one best linear classifier from a pool of diversified linear classifiers. Depending on the dataset or subset of training samples at each node, one type of hyperplane may be best suited for partitioning the samples at that particular node. That means, such hyperplane can best optimize the impurity criterion compared to the others. Hence, selecting the one from a pool using an impurity criterion endows the random forest to generalize better with more accurate and diversified tree structure.

- We employ six linear classifiers namely Support Vector Machines (SVM), Multisurface Proximal SVM (MPSVM), Linear Discriminant Analysis (LDA), Least Squares SVM (LSSVM), Ridge Regression (RR) and Logistic Regression (LR). Extension of the proposed method to other linear classifiers is straightforward.

Oblique Random Forest (ObRaF) with linear hyperplanes based on one-vs-all approach

- Many popular binary classifiers such as SVM use “one-vs-all” approach to breakdown a multi-class classification problem into several binary classification problems. A caveat is that it may not always be the best method to deal with multi-class problems.

- Such methods can however, be integrated at the internal nodes of the decision trees. A linear classifier at each node doesn't need to be a perfect classifier but simply aid in further classification or partitioning.

- At each node of the decision tree a linear classifier is employed to separate one class from all other classes. That means, the best linear hyperplane at each node is obtained by searching amongst the *K (K = C* at the root node*)* hyperplanes obtained by transforming a multi-class classification problem into *K* binary classification problems.

- This idea was implemented in [1] using MPSVM as a linear classifier.

[1] R. Katuwal, P. N. Suganthan, Enhancing multi-class classification of random forest using random vector functional neural network and oblique decision surfaces, in: 2018 International Joint Conference on Neural Networks (IJCNN), 2018, pp. 1-8. doi:10.1109/IJCNN.2018.8489738

- At each node of the tree, *K* oblique hyperplanes generated using MPSVM are compared and the one that best optimizes Eq. (1) is selected as the splitting hyperplane. The optimization problem is given as:

$$\max(Gini_p - Gini_c)\,, \tag{1}$$

where

$$Gini_p = 1 - \sum_{i=1}^{K} \left(\frac{n_{w_i}}{n_t}\right)^2\,, \tag{2}$$

$$Gini_c = \frac{n_t^l}{n_t}\left[1 - \sum_{i=1}^{K}\left(\frac{n_{w_i}^l}{n_t^l}\right)^2\right] + \frac{n_t^r}{n_t}\left[1 - \sum_{i=1}^{K}\left(\frac{n_{w_i}^r}{n_t^r}\right)^2\right] \tag{3}$$

where $Gini_p$ and $Gini_c$ are the values of Gini impurity at the parent node and child nodes respectively, n_t is the number of data samples in the parent node, n_t^l, n_t^r is the number of data samples that reach the left and right child nodes of the current parent node, and $n_{w_i}^l, n_{w_i}^r$ are the number of samples of class w_i in the left and right child nodes respectively.

Algorithm 1 Oblique Decision Tree (obDT) Training

Require: Labeled training data (X, Y)
Require: Maximum tree depth $\mathcal{D}$
Output: Linear decision boundaries (w's and b's)
for d_T `from 1 to` $\mathcal{D}$ **do**
    Check stopping criteria at each node at depth d_T.
    Create K data partitions based on one-vs-all approach.
    Train a linear classifier on each partition.
    Compute Eqs. (2) and (3).
    return w and b of the hyperplane that best optimizes Eq. (1) at each node at depth d_T.
end for

- One of the issues with the "one-vs-all" technique is that it is computationally expensive when the number of classes is very large. Thus, generating *K* hyperplanes at each node can be computationally expensive in such cases.

- Since the linear hyperplane at each node is selected from a pool of hyperplanes based on the impurity criteria (more specifically Eq. (1)), it usually results in pure child nodes faster than previous exhaustive approaches (in case of RaF) and obRaF without impurity optimization approaches. Thus, the trees in such "one-vs-all" oblique random forest variants are generally shallow than the standard trees. This may negate the complexity associated with generating many hyperplanes at each node.

Heterogeneous Oblique Random Forest (ObRaF(H))

obRaF(H): Heterogeneous Oblique Random Forest, by Rakesh Katuwal, P.N. Suganthan, Le Zhang. R1 submitted to Pattern Recognition Journal

- Different linear classifiers employ different techniques to generate a linear decision boundary (maximum margin hyperplane in case of SVM, clustering hyperplane in case of MPSVM, etc.).

- One family of linear classifier may not always create the best partition or exploit the geometric structure in the data as the decision boundary depends on the training samples falling at each node.

- Based on our experimental results, we observe that, even though some of the linear classifier based oblique random forests consistently perform better (have high ranks compared to others), they are not always the best oblique random forest variants for every dataset. This is congruent with the famous No-Free-Lunch theorem. Thus, even though some of the linear classifier based variants have lower ranks compared to others, we can still integrate them to form a heterogeneous linear classifiers based oblique random forest.

- One-vs-all formulation requires evaluating n linear classifiers in K binary partitions thus, requiring nK evaluations at each node. An innate issue with one-vs-all based decision trees is that it leads to heavily unbalanced trees. Another issue with such one-vs-all based partitioning method is that it limits the search space of the optimal oblique splits to K choices. Even though the exhaustive search of the optimal oblique split is NP-hard, it is beneficial to increase the search space of the oblique splits to obtain the best split.

- Thus, to address or solve the above issues, we employ a hyperclasses based partitioning along with one-vs-all partitioning while using several linear classifiers at each node.

Generating several partitions and their ranking

- In standard decision trees, each axis-parallel split is rated based on an impurity criterion. Specifically, all the splits at each non-leaf nodes are associated with an impurity measure (we denote this by $g = Gini_p - Gini_c$ using Eqs. (2) and (3)) and the one with the maximum value (refer to Eq. (1)) is the selected split ('best' split) at that particular node.

- However, an exhaustive search for the optimal oblique split is NP-hard [1]. To extenuate an exhaustive search, we restricted the search space to limited oblique splits in one-vs-all based formulation. We observed that in case of one-vs-all partitions, partitions with higher positive training samples (large number of training samples for the one class) basically had higher g values compared to other partitions (We call the `one' class as positive class and the 'all' class as the negative class in case of one-vs-all partitioning.). This is intuitive since separating large number of samples of one class produces purer child nodes. In a nutshell, we can employ the information available at each node to obtain the best split.

- Generally, in case of oblique decision trees, a multi-class classification problem is treated as binary classification problem at each node. Thus, the purity of child nodes depends on the partitioning of the classes and the quality of the split. For the latter, we employ several linear classifiers as discussed in the subsequent slides. For the former, we combine the one-vs-all partitioning idea and hyperclasses based partitioning approach of [2]. In hyperclasses based partitioning, several classes are grouped in two hyperclasses (say c_1 and c_2) based on a criterion. For example, with 5 classes {1,2,3,4,5} at a node, they can be grouped into two hyperclasses c_1 = {1,2,5} and c_2 = {3,4} based on some criteria. [ ] uses the Bhattacharyya distance between the classes to group them into two classes.

[1] S. K. Murthy, S. Kasif, S. Salzberg, A system for induction of oblique decision trees, Journal of artificial intelligence research 2 (1994) 1-32.
[2] L. Zhang, P. N. Suganthan, Oblique decision tree ensemble via multisurface proximal support vector machine, IEEE Transactions on Cybernetics 595 45 (10) (2015) 2165-2176. doi:10.1109/TCYB.2014.2366468

- At each node of a decision tree, there are $2^{K-1}-1$ possible ways of partitioning K classes [1]. For example, if there are 10 classes at a node, these classes can be arranged in $2^{10-1}-1 = 511$ partitions. For large number of classes, the number of partitions is rampant. Thus, we selectively filter the partitions and use only the top ranked partitions to train the linear classifiers. Specifically, we use ideal gini score and clustering validity index to rank the partitions. Ranking the partitions enable us to use few significant partitions while avoiding the insignificant partitions (those likely to have small g). Before delving into the partitioning approach and ranking, we first introduce the two criteria used to rank the partitions.

Ideal Gini score

- The ideal gini score refers to the gini value obtained in the ideal case where an oblique split perfectly separates all the samples of one class from the other class.

- In one-vs-all partitioning, the ideal case is equivalent to separating all samples of the positive class from the rest of the classes while in hyperclasses based partitioning, it is equivalent to separating all the samples of one hyperclass from the other.

- Computing ideal gini score does not require generating a hyperplane but simply computing g assuming the scenario when all the samples of one class (or hyperclass) are sent to the right node and all others to the left node (We represent the ideal gini score as g_I and actual gini score as g respectively. We also use the terms partitions and hyperclasses interchangeably).

[1] A. K. Y. Truong, Fast growing and interpretable oblique trees via logistic regression models, Ph.D. thesis, University of Oxford (2009).

- A higher gini score means purer child nodes. If a split with high ideal gini score is able to replicate its g_I it is likely to have higher actual gini score g. Computing g_I beforehand, without actually training a linear classifier gives us an indication which partition the oblique split should focus on. By training the linear classifiers on the partitions with higher g_I which are likely to give higher g, we can ignore the partitions with lower g_I which are likely to give lower g.

Cluster separability

- g_I is a measure of the purity of the child nodes in an ideal setting with the assumption that the partitions are well-separated and a linear decision boundary can exactly classify or separate them. However, in the real settings, this may not always be true. Thus, we also need to consider the geometric structure of the data at each node when ranking the partitions. A partition may have high g_I but low g if the partitions are difficult or hard to separate. Thus, we use a cluster validity index to evaluate the separability of each partition or cluster.

- The cluster validity indices aim to identify the sets of clusters that are compact and well-separated. Since we are interested in binary decision trees or binary partitioning, it is highly desirable that the two partitions are well separated. Dunn index (DI) [1] is one of the popular cluster validity index. The Dunn index for k clusters is defined by:

[1] J. C. Dunn, Well-separated clusters and optimal fuzzy partitions, Journal of Cybernetics 4 (1) (1974) 95-104. arXiv:10.1080/01969727408546059

$$DI_k = \min_{i=1,\ldots,k} \left\{ \min_{j=1+i,\ldots,k} \left(\frac{diss(c_i, c_j)}{\max_{m=1,\ldots,k} diam(c_m)} \right) \right\} \quad (7)$$

where $diss(c_i, c_j) = \min_{x \in c_i, y \in c_j} ||x-y||$ is the dissimilarity between the clusters c_i and c_j (also known as inter-cluster distance) and $diam(C) = \max_{x,y \in C} ||x-y||$ is the intra-cluster function (diameter) of the cluster. A higher DI indicates better clustering. We use Bhattacharyya distance as the distance metric as it is considered a good measure for class separability between two classes [24]. The Bhattacharyya distance between two classes c_1 and c_2 with means and co-variances $(\mu_1, \sum_1)$ and $(\mu_2, \sum_2)$ respectively, is defined as:

$$B(c_1, c_2) = \frac{1}{8}(\mu_2 - \mu_1)^T \left(\frac{\sum_1 + \sum_2}{2} \right)^{-1} (\mu_2 - \mu_1) + \frac{1}{2} ln \frac{|(\sum_1 + \sum_2)/2|}{\sqrt{|\sum_1||\sum_2|}} \quad (8)$$

- Since we only consider binary partitions and their separability, the two partitions need to be well separated but not necessarily compact, thus, we can ignore the denominator part of DI i.e. the diameter of the cluster. The inter-cluster distance is the distance between the two closest classes, one in each partition.

[24] L. Zhang, P. N. Suganthan, Oblique decision tree ensemble via multisurface proximal support vector machine, IEEE Transactions on Cybernetics 595 45 (10) (2015) 2165-2176. doi:10.1109/TCYB.2014.2366468

Employing Exhaustive search or Genetic Algorithm

- Depending on the number of classes at each node, the partitions are obtained through either Genetic Algorithm (GA) or exhaustive search. We evaluate an equal number of one-vs-all partitions and hyperclasses based partition at each node. For example, if there are K one-vs-all partitions, we obtain another top K hyperclasses based partition from GA or exhaustive search and re-rank these $2K$ partitions. We do this because for K classes, there are 2^{K-1}-1-K possible hyperclasses based partitions. For large number of classes, the number of hyperclasses based partition drastically increases.

- In [1], several classes at each node are grouped into two hyperclasses based on the pairwise Bhattacharyya distance between the classes. Such partition may have a higher DI but not necessarily higher g_l .Thus, we introduce a slight variation in this approach. Specifically, keeping the two furthest classes obtained from the pairwise Bhattacharyya distance approach fixed, we randomly assign 50% (value used in our paper) of the classes mid-way between the two furthest classes to either of the partitions. We rank all these partitions based on g_l and DI and select the top K hyperclasses based partitions.

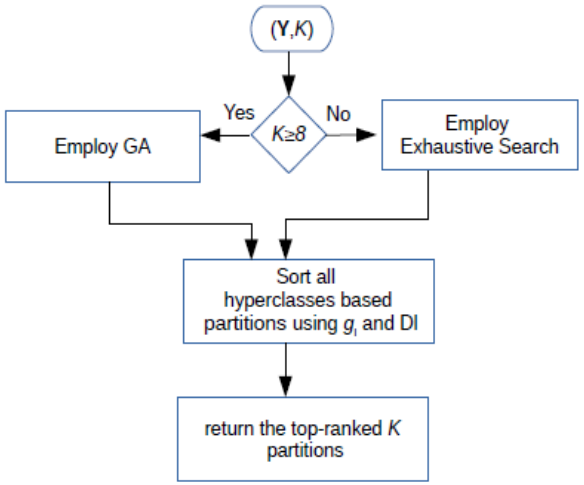


Figure 3: Flowchart explaining GAorExhSearch function of Algorithm 2.

[1] L. Zhang, P. N. Suganthan, Oblique decision tree ensemble via multisurface proximal support vector machine, IEEE Transactions on Cybernetics 595 45 (10) (2015) 2165-2176. doi:10.1109/TCYB.2014.2366468

Employing several linear classifiers at each node

- Employing a single type of linear classifier at a node may not yield the best partition. Such a limitation can however, be remedied by applying several diverse linear classifiers.

- Depending on the training samples at each node, one type of hyperplane (from many) may be best suited for partitioning the samples at that particular node. We employ several diverse linear classifiers at each node resulting in heterogeneous oblique decision tree.

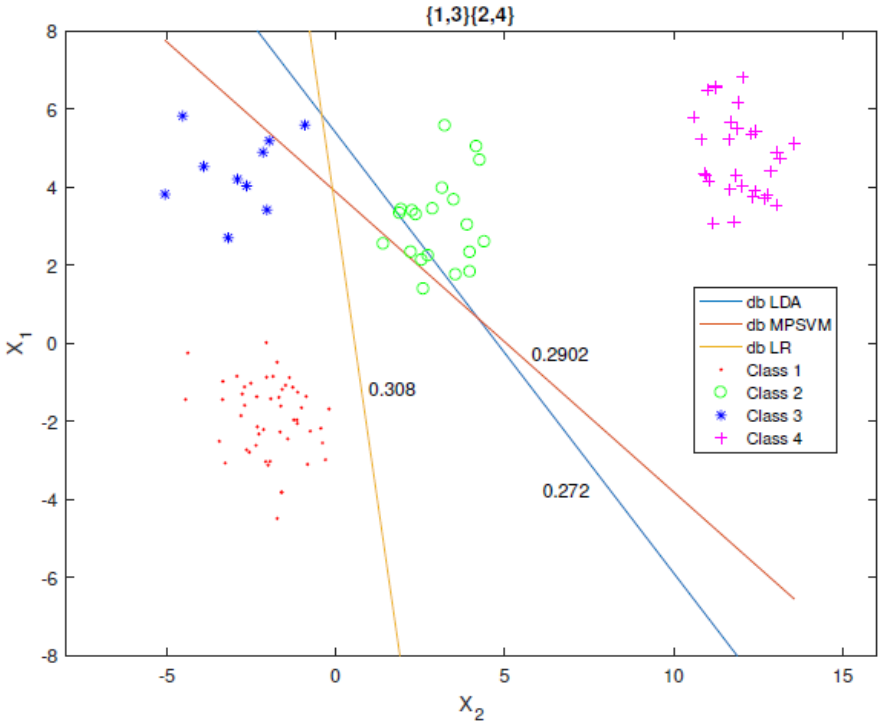


Fig: Decision boundaries generated by different linear classifiers and their associated g.

Algorithm 2 Heterogeneous Oblique Decision Tree Training

Require: Labelled training data (X, Y)
Require: Maximum tree depth $\mathcal{D}$
for d_T `from 1 to` D_{max} **do**
  Check the stopping criteria for all nodes at depth d_T.
  if $K \geq 5$ at a node **then**
    Use GAorExhSearch function to get the K hyperclasses based partitions and also create K one-vs-all partitions.
    Sort the $2K$ partitions obtained in line 6 using their average ranks obtained via both g_I and DI.
    Repeatedly divide the sorted partitions into halves and train the linear classifiers subsequently on the upper half to get several linear decision boundaries and compute g. (refer to Section 4.2.1)
    If the number of partitions in the upper half ≤ 5, train remaining linear classifiers on all remaining partitions.
  else
    Train all linear classifiers on all possible partitions and compute g.
  end if
  Select the decision boundary with the highest g as the splitting hyperplane at depth d_T.
end for

function GAorExhSearch ▷ refer to Sections 3.1.3 and 4.2.1
  if $K < 8$ **then** ▷ *Employ Exhaustive Search*
    Generate all possible hyperclasses based partitions.
    Sort all the partitions obtained in line 3 based on their average ranks obtained via g_I and DI.
  else ▷ *Employ Genetic Algorithm*
    Compute the pairwise Bhattacharyya distance between the classes.
    Find the two classes with largest distance and label them ext1 and ext2 respectively.
    Assign every remaining classes to ext1 and ext2 depending on their proximity to these classes.
    Create initial population by randomly assigning 50% of the classes mid-way between ext1 and ext2 to either one of the them.
    for i `from 1 to` *maxGeneration* **do**
      Employ GA and sort all the partitions at each generation using their average ranks.
    end for
  end if
  return the top-ranked K partitions
end function

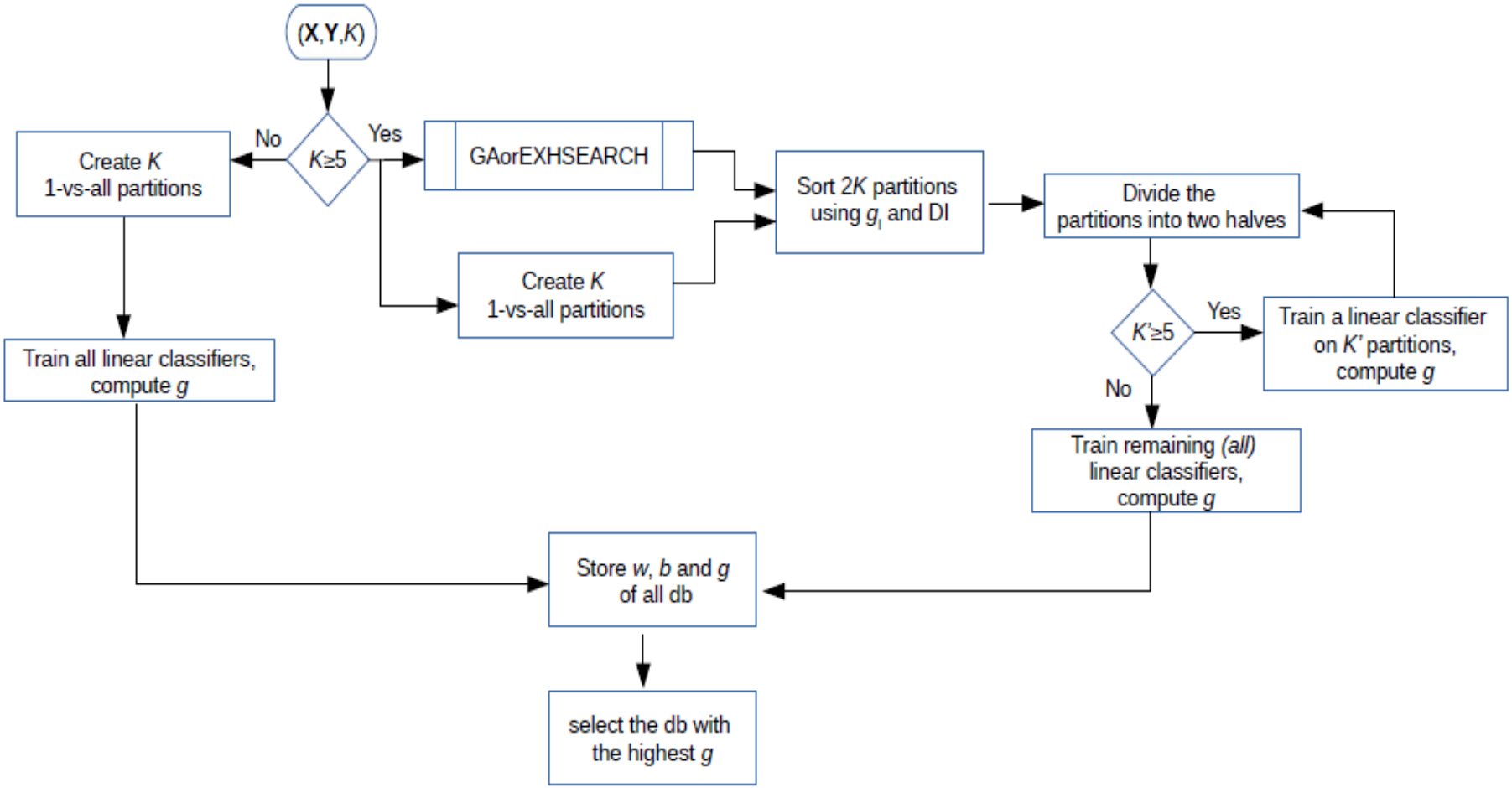


Figure 6: Flowchart depicting our proposed method at each node

Experiments

- We compare the classifiers on 121 datasets from the UCI repository as done in [1] and [15].
- For the random forest based methods, the number of trees is set to 500. Only the number of features randomly selected at each node, m_{try}, is tuned.
- For the heterogeneous oblique decision tree, we employ six linear classifiers in the order of LDA, MPSVM, LSSVM, LR, SVM and RR. These classifiers are ordered based on their performance. These ordered classifiers are subsequently used (beginning with LDA and followed by the rest) in the top ranked half obtained after repeated division of the sorted partitions (refer to line 8 of Algorithm 2). For the GA based partitioning, the population size is set to $2K$ while maximum generation is set to 5. We empirically found that setting these values leads to satisfactory results.

Table 1: Average Friedman Rank Based on Classification Accuracies of Each Method on 121 UCI Datasets

	RaF	obRaF[15]	obRaF(L)	obRaF(RR)	obRaF(S)	obRaF(LS)	obRaF(LR)	obRaF(M)	obRaF(H)
Rank	5.4	5.07	4.28	6.13	5.65	5.14	5.46	4.43	**3.44**

L, RR, S, LS, LR, M and H in the brackets along with obRaF refer to our proposed LDA, Ridge Regression, SVM, LSSVM, LR, MPSVM and heterogeneous oblique random forests respectively. Lower rank reflects better performance.

[1] M. Fernandez-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, Journal of Machine Learning Research 15 (2014) 3133-3181.
[15] L. Zhang, P. N. Suganthan, Oblique decision tree ensemble via multisurface proximal support vector machine, IEEE Transactions on Cybernetics 595 45 (10) (2015) 2165-2176. doi:10.1109/TCYB.2014.2366468

Complexity Analysis

- As the computational time required to grow an oblique tree is highly variable, we instead focus on the complexity of partitioning a single node.

- Consider a training set **X** which is a *N* x *d* matrix with *N* samples and *d* features. As each node is associated with different sizes of training data, for the sake of simplicity, here we consider the complexity at the root node.

- Let *K* denote the number of classes at the root node. The number of splits that need to be evaluated in the one-vs-all based oblique decision trees is upper bounded by *K*.

Regardless of the computational time for calculating the Gini scores, the complexities associated with each type of oblique decision trees at the root node are $\mathcal{O}(Nd^2 \cdot K)$ for obRaF(L), obRaF(RR), obRaF(LS), obRaF(LR), $\mathcal{O}(N^2 d \cdot K)$ for obRaF(S), and $\mathcal{O}(d^3 \cdot K)$ for obRaF(M) respectively. As the heterogeneous decision tree employs all the linear classifiers several times, the overall computational complexity for obRaF(H) is the aggregate of all the complexities associated with each linear classifier.

Algorithm	Complexity
LDA	$\mathcal{O}(Nd^2)$ [136]
RR	$\mathcal{O}(Nd^2)$
SVM	$\mathcal{O}(N^2 d)$ [136]
LS-SVM	$\mathcal{O}(Nd^2)$ [137]
LR	$\mathcal{O}(Nd^2)$ [136]
MPSVM	$\mathcal{O}(d^3)$ [69]

The complexity of an iterative algorithm such as LR is analyzed for one iteration, and so their actual running time may be slower.

Overall Comparisons Among 190 Classifiers

Table 2: Overall comparison of 20 top ranked classifiers out of 190 methods in 121 datasets

Method	Rank	Friedman Rank	Mean Accuracy
obRaF(H)*	1	29.22	83.39
obRaF(L)*	2	34.28	82.64
obRaF(M)*	3	35.28	82.3
MPRaF	4	36.73	82.05
CoKRR-max	5	37.84	81.79
MPRRoF	6	38.14	82.35
parRaF_t	7	38.6	81.96
RaF_t	8	38.73	82.3
obRaF(LS)*	9	39.37	82.19
obRaF(S)*	10	39.92	82.19
obRaF(LR)*	11	40.33	81.95
svm_C	12	42.29	81.77
svmPoly_t	13	44.57	80.31
KRR	14	45.55	81.52
rForest_R	15	45.65	81.94
svmRadialCost_t	16	46.32	80.54
obRaF(RR)*	17	46.83	81.51
svmRadial_t	18	48.98	80.17
RVFL ensemble	19	49.27	80.94
C5.0_t	20	49.38	80.56

"*" indicates the methods introduced in this work. We follow the naming conventions of [14, 15].

Table 3: Statistical comparison of obRaF(H) and each of the other top 20 ranked classifiers

Method	Rank	Wilcoxon signed-rank p, excluding errors
obRaF(L)*	2	2e-04
obRaF(M)*	3	4e-04
MPRaF	4	4e-05
CoKRR-max	5	2e-03
MPRRoF	6	1e-04
parRaF_t	7	4e-04
RaF_t	8	2e-04
obRaF(LS)*	9	1e-06
obRaF(S)*	10	1e-08
obRaF(LR)*	11	1e-08
svm_C	12	8e-04
svmPoly_t	13	3e-05
KRR	14	1e-03
rForest_R	15	4e-06
svmRadialCost_t	16	6e-04
obRaF(RR)*	17	1e-09
svmRadial_t	18	4e-07
RVFL ensemble	19	3e-07
C5.0_t	20	3e-07

The statistical comparison is done using a Wilcoxon signed-rank test. The test is performed by removing the datasets which failed to run for the top 20 ranked classifiers instead of incorrectly imputing them [43]

[14] M. Fernandez-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, Journal of Machine Learning Research 15 (2014) 3133-3181.

[15]] L. Zhang, P. N. Suganthan, Benchmarking Ensemble Classifiers with Novel Co-trained Kernel Ridge Regression and Random Vector Functional Link Ensembles, IEEE Computational Intelligence Magazine, Nov 2017.

[43] M. Wainberg, B. Alipanahi, B. J. Frey, Are random forests truly the best classifiers?, Journal of Machine Learning Research 17 (110) (2016) 1-5. URL http://jmlr.org/papers/v17/15-374.html

obRaF(H): Heterogeneous Oblique Random Forest, by Rakesh Katuwal, P.N. Suganthan, Le Zhang. R1 submitted to Pattern Recognition Journal

Table 4: Wilcoxon signed-rank test based statistical comparison of the 20 top ranked classifiers out of 190 Methods

	obRaF(H)	obRaF(L)	obRaF(M)	MPRaF	CoKRR-max	MPRRoF	parRaF_t	RaF_t	obRaF(LS)	obRaF(S)	obRaF(LR)	svm_C	svmPoly_t	KRR	rForest_R	svmRadialCost_t	obRaF(RR)	svmRadial_t	RVFL ensemble	C5.0
obRaF(H)		+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
obRaF(L)	-									+	+		+		+		+	+	+	+
obRaF(M)	-									+	+				+		+	+	+	+
MPRaF	-																			+
CoKRR-max	-																			
MPRRoF	-																			+
parRaF_t	-																			+
RaF_t	-																			+
obRaF(LS)	-																		+	
obRaF(S)	-	-	-																	
obRaF(LR)	-	-	-																	
svm_C	-																			
svmPoly_t	-	-																		
KRR	-																			
rForest_R	-	-	-																	
svmRadialCost_t	-																			
obRaF(RR)	-	-	-																	
svmRadial_t	-	-	-																	
RVFL ensemble	-	-	-		-				-											
C5.0_t	-	-	-	-		-	-	-												

+ means the method in the corresponding row is statistically significantly better than the method in the corresponding column. Similarly, - indicates that the method in the corresponding row is statistically significantly worse than the method in the corresponding column. The test is performed by removing the datasets which failed to run for the top 20 ranked classifiers instead of incorrectly imputing them [43] (refer to the Supplementary file for more details).

Comparison with stand-alone differential decision tree (sNDF) [1]

Table 4: Comparison in terms of accuracy(%)

Datasets	sNDF[1]	obRaF(H)
G50c	17.4±1.52	**11.44±1.35**
Letter	**2.92±0.17**	3.5±0.13
USPS	5.01±0.24	5.1±0.2
MNIST	2.8±0.12	2.68±0.17
Char74k	16.04±0.2	16.0±0.25

sNDF is the shallow neural decision forests of [1].

Table 5: Comparison in terms of accuracy(%)

Datasets	Patterns	Features	Classes	sNDF[1]	obRaF(H)
annealing	798	38	6	69	**96.23**
audiology-std	226	59	8	72	**87.21**
hill-valley	606	100	2	52.64	**69.14**
image-segmentation	210	19	7	**31.19**	17.29
monks-3	3190	6	2	**69.67**	55.84
optical	3823	62	10	96.99	**97.19**
pendigits	7494	16	10	96.97	**97.14**
soybean	307	35	18	**81.12**	72.34
st-landsat	4435	36	6	89.35	**91.96**
thyroid	3772	21	3	98.19	**98.5**

- sNDF is a stochastic, differentiable decision tree trained in an end-to-end manner.
- combines representational learning with decision trees.
- split node parameters are learned via back-propagation.

- Table 4 compares the sNDF and obRaF(H) in the datasets used in [1].
- Additional comparisons using UCI datasets in Table 5.

[1] P. Kontschieder, M. Fiterau, A. Criminisi, S. R. Bul, Deep neural decision forests, in: 2015 IEEE International Conference on Computer Vision 525 (ICCV), 2015, pp. 1467-1475. doi:10.1109/ICCV.2015.172

Summary

- We first presented several variants of oblique random forest employing several linear classifiers at the internal nodes of the tree. The oblique splits were generated based on one-vs-all partitioning.

- We also presented a heterogeneous variant employing diverse linear classifiers trained on some selective top ranked partitions. The partitions were created using both one-vs-all and two-hyperclasses based approaches and ranked using the ideal gini scores and cluster separability.

- This work also expands the comparison of classifiers on 121 UCI datasets and we hope that the top ranked classifiers could be considered as new baselines for the practitioners. Besides classification, such variants of oblique random forest can also be applied in other domains such as regression, time series forecasting and is an interesting research direction for future.

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Extremely Randomized (Extra) Trees[1]

- Aims to further reduce the variance of the model by incorporating stronger randomization technique.
- Increases bias. Therefore, uses the original dataset instead of bootstrapped versions to grow each tree unlike Bagging and Random Forest.
- Tree growing procedure is similar to Random Forest except at each node:

For each feature i in m_{try}, a random splitting value is picked between the minimum and the maximum values $a_s^{(i)} \in [a_{min}^{(i)}, a_{max}^{(i)}]$. The best split (feature and threshold) chosen at each node is the one which produces the highest reduction in the impurity.

[1]P. Geurts, D. Ernst, and L. Wehenkel, Extremely randomized trees, *Machine Learning*, vol. 63, no. 1, pp. 3-42, Apr 2006

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Boosted Trees

1. Ada-Boosted Trees
2. Gradient Boosted Trees

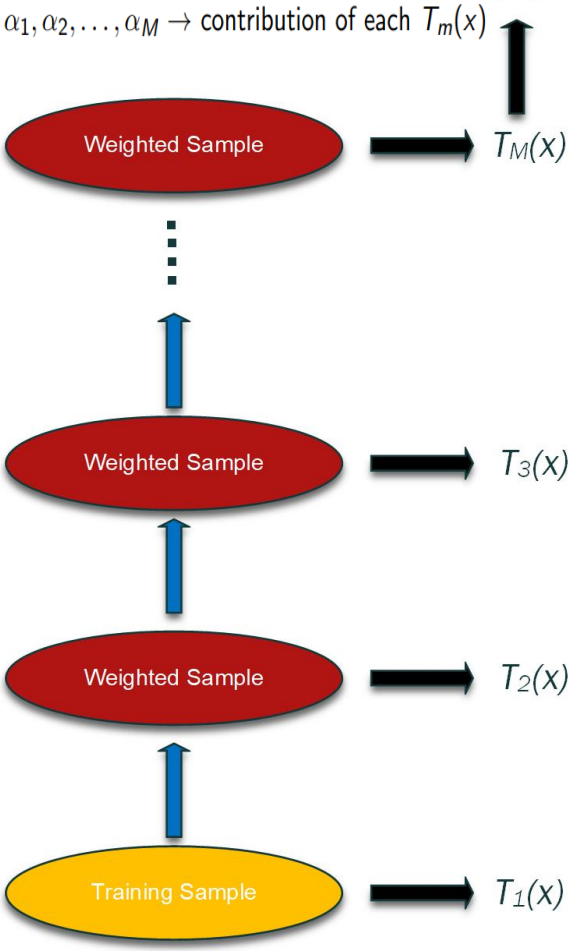
- The boosted trees are grown sequentially rather than in parallel as in Random Forest with bagging.
- Each tree is a weak classifier (shallow tree with high bias, low variance) built to improve upon the weakness of the previous tree.
- They can reduce both bias and variance.

Ada-boosted Trees[1]

$$T(\mathrm{x}) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$$

Adaboosted Trees

1. Initialize the observation weights $w_i = 1/N,\ \ i = 1, 2, \ldots, N$.
2. For $m = 1, \ldots, M$: $\rightarrow$# of iterations
   (a) Fit a shallow decision tree $T_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq T_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq T_m(x_i))],\ \ i = 1, 2, \ldots, N$
3. Output $T(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m T_m(x)\right]$.



[1] Freund, Y. and Schapire, R (1997), A decision-theoristic generalization of online learning and an application to boosting, *Journal of Computer and System Sciences* 55: 119-139

Ada-boosted Trees

- Initially, all of the weights are set to $w_i = 1/N$ and the tree is trained in the usual manner.
- For each successive iteration *m = 2,3…,M,* the observation weights are individually modified and successive trees are grown to the weighted observations.
- At step *m,* those observations that were misclassified by the classifier $G_{m-1}(x)$ have weights increased and vice versa. Thus, observations that are difficult to classify correctly receive ever-increasing influence.
- Each successive classifier is thereby forced to concentrate on those training observations that are missed by previous ones in the sequence.
- The weights can be either engraved in the architecture of the classifiers to reflect the importance of each observation or used as sampling distribution so those observations with higher weights have higher probability of being selected for the training of next classifier [1].

[1] Friedman,J., Hastie, T., & Tibshirani, R., The elements of statistical learning, Vol.1, No.10, New York, NY USA, Springer series in statistics.

Ada-boosted Trees

- Previous Ada-boosting algorithm is known as Adaboost.M1 (binary classification).
- It has the following variants
  - Adaboost.M2 (multiclass classification)
  - LogitBoost (binary classification for poorly separable classes)
  - Gentle AdaBoost or GentleBoost (for use with multilevel categorical problems)
  - RobustBoost (robust against label noise)
  - LSBoost (least squares boosting for regression ensembles)
  - LPBoost (multiclass classification using linear programming boosting)
  - RUSBoost (multiclass classification for skewed or imbalanced data)
  - TotalBoost (multiclass classification more robust than LPBoost)

Remember the computational advantage of bagging over boosting: parallelizable versus sequential.

Adaboosted Trees: ranked 20+ out of 180 classifiers based on a comprehensive evaluation on 121 datasets. Random Forest is the 1st ranked classification algorithm[1]. Hence, bagging based RF seems to have performance advantage too.

[1] Fernández-Delgado, Manuel, et al. "Do we need hundreds of classifiers to solve real world classification problems?." *JMLR* (2014): 3133-3181.

Gradient Boosted Trees[1]

Induce trees *Tm(x)*, whose predictions are as close as possible to the negative gradient, r_{im}.

For regression, $r_{im} = yi - f(xi)$ when the loss function is squared error.

For classification, the loss function is the multinomial deviance:

$$L(y, p(x)) = -\sum_{k=1}^{K} I(y = G_k) \log p_k(x)$$

$$= -\sum_{k=1}^{K} I(y = G_k) f_k(x) + \log\left(\sum_{k=1}^{K} e^{f_l(x)}\right)$$

where K is the number of classes and $G(x)$ is the predicted class.

[1] Friedman,J., Hastie, T., & Tibshirani, R., The elements of statistical learning, Vol.1, No.10, New York, NY USA, Springer series in statistics.

Gradient Boosted Trees for Regression[1]

Gradient Tree Boosting Algorithm for Regression

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1, \ldots, M$: $\rightarrow$# of iterations
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions $R_{jm}, j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

A constant γ_j is assigned to each such region and the predictive rule is

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

For Trees, γ parameterizes the split variables and split points.

[1] Friedman,J., Hastie, T., & Tibshirani, R., The elements of statistical learning, Vol.1, No.10, New York, NY USA, Springer series in statistics.

Gradient Boosted Trees

- Thus, for classification the negative gradient of the loss function is:

$-r_{ikm} = I(y_i = G_k) - p_k(x_i)$ where $p_k(x_i)$ is the class conditional probability calculated as:

$$p_k(x_i) = \frac{e^{f_k(x_i)}}{\sum_{l=1}^{K} e^{f_l(x_i)}}$$

- The algorithm for classification is similar to that shown in previous slide. Lines 2(a)-2(d) are repeated *K* times at each iteration *m,* once for each class.

Gradient Boosting for K-class Classification

1. Initialize $f_{k0}(x) = 0, \quad k = 1, 2, \ldots, K.$
2. For $m = 1$ to M:
   (a) Set

$$p_k(x) = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, k = 1, 2, \ldots, K$$

   (b) For $k = 1$ to K:
      i. Compute $r_{ikm} = y_{ik} - p_k(x_i), \quad i = 1, 2, \ldots, N$
      ii. Fit a regression tree to the targets $r_{ikm}, \quad i = 1, 2, \ldots, N$, giving terminal regions $R_{jkm}, \quad j = 1, 2, \ldots, J_m$.
      iii. Compute

$$\gamma_{jkm} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jkm}} r_{ikm}}{\sum_{x_i \in R_{jkm}} |r_{ikm}|(1 - |r_{ikm}|)}, \quad j = 1, 2, \ldots, J_m$$

      iv. Update $f_{km}(x) = f_{k,m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jkm} I(x \in R_{jkm})$.
3. Output $\hat{f}(x) = f_{kM}(x), \quad k = 1, 2, \ldots, K.$

Gradient Boosted Trees

- The tunable parameters are:
  1. Number of iterations, *M (number of trees).*
  2. The depth of each of the constituent trees J_m, $m = 1,2,\ldots,M$.
  3. Shrinkage parameter, v (0-1 range) (trade-off between M)
  4. Subsampling ratio, η (stochastic gradient boosting) for huge training datasets.

A nice package for Gradient Boosting widely used by researches and practitioners in Kaggle competitions:

XGBoost (https://github.com/dmlc/xgboost).

With patient parameter tuning, Gradient Boosted Tree usually becomes the best method in Kaggle competitions …

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Deep Random Forest[1]

- An alternative to Deep Neural Networks
- *Representation learning* ability → crucial for deep neural networks.

- Known as gcForest, it consists of two blocks.
  1. Multi-Grained Scanning → extracts feature relationships
  2. Cascade Forest Structure → layer-by-layer processing of raw features

[1] Zhi-Hua Zhou, and Ji Feng, Deep Forest: Towards an Alternative to Deep Neural Networks, *International Joint Conference on Artificial Intelligence (IJCAI-17)*

Deep Random Forest

1. Multi-Grained Scanning

- Sliding windows are used to scan the raw features.
- The feature vector generated by sliding window is then used to train a completely-random Forest (Extra Trees) and a Random Forest.
- The class vectors (probability distribution) generated by the Forest are concatenated to form transformed features.

2. Cascade Forest Structure

- It employs a cascade structure of Random Forest as shown in the previous slide.
- Each level of cascade receives feature information processed by its preceding level, and outputs its processing result to the next level.

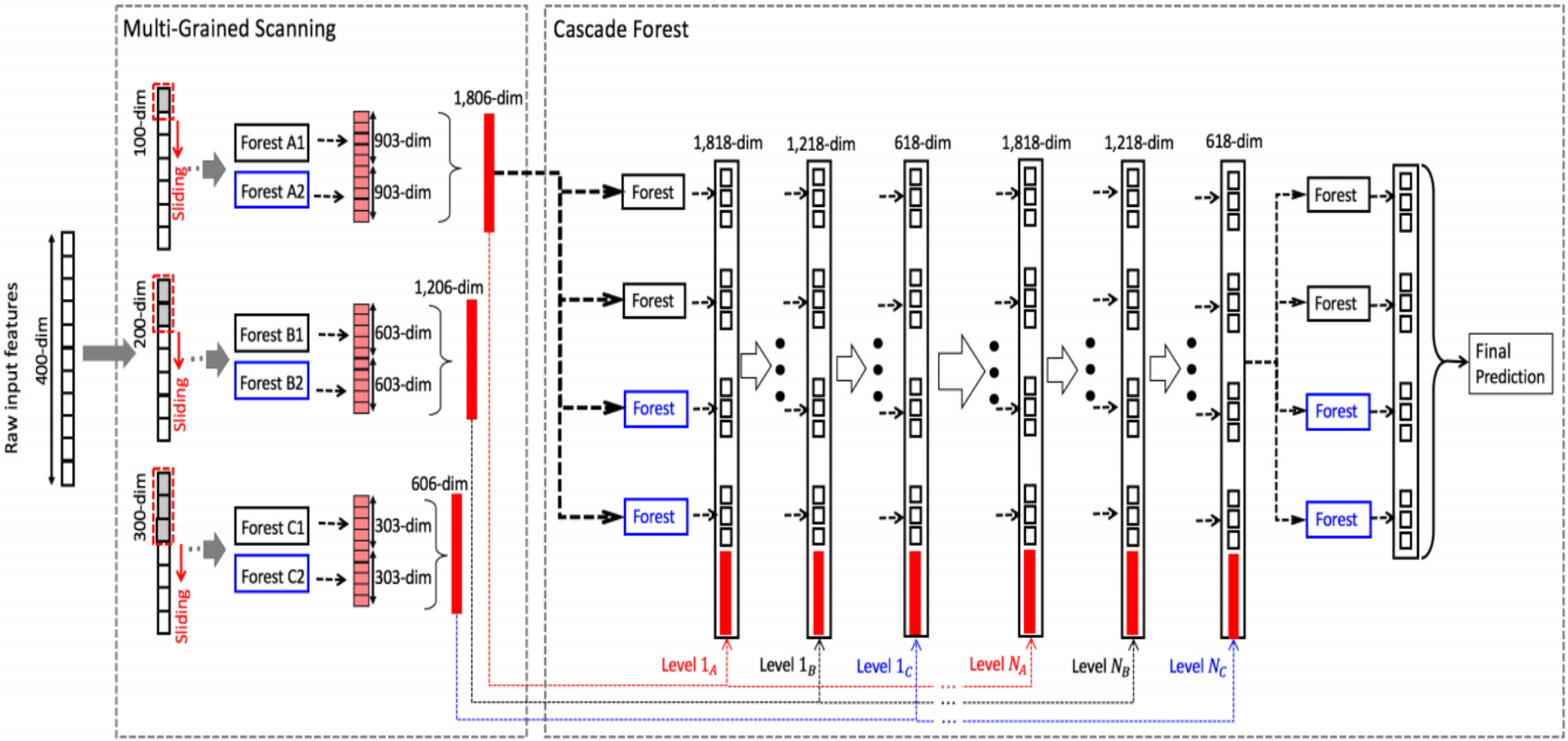


Figure: gcForest Architecture for 3 classes.

Deep Random Forest

Overall Procedures:

- Suppose, the original input is 400 raw features. 3 window sizes (100dim, 200dim, 300dim) are used.
- For *m* training examples, a window with size of 100 features will generate a dataset of 301 x *m* 100-dimensional training examples. These data will be used to train a completely-random tree forest and a random forest, each containing 500 trees. If there are three classes to be predicted, a 1,806-dimensional feature vector will be obtained. The transformed training set will then be used to train the 1st-grade of cascade forest.
- Similarly, sliding windows with sizes of 200 and 300 features will generate 1,206-dimensional and 606-dimensional feature vector, respectively, for each original training example. The transformed feature vectors, augmented with the class vector generated by the previous grade, will then be used to train the 2nd-grade and 3rd-grade of cascade forests, respectively.
- This procedure will be repeated till convergence of validation performance. In other words, the final model is actually a cascade of cascade forests, where each level in the cascade consists of multiple grades (of cascade forests), each corresponding to a grain of scanning.

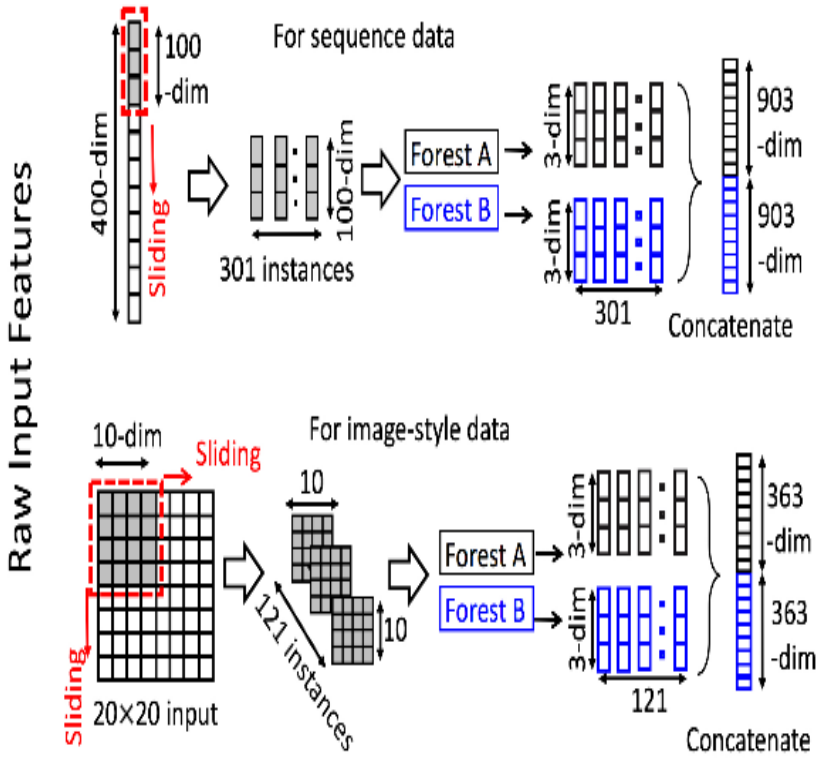


Figure : Illustration of feature re-representation using sliding window scanning. Suppose there are three classes, raw features are 400-dim, and sliding window is 100-dim.

Deep Random Forest (gcForest)

- Results on some datasets from [1].

Table 2: Comparison of test accuracy on MNIST

gcForest	**99.26%**
LeNet-5	99.05%
Deep Belief Net	98.75% [Hinton *et al.*, 2006]
SVM (rbf kernel)	98.60%
Random Forest	96.80%

Table 3: Comparison of test accuracy on ORL

	5 image	7 images	9 images
gcForest	**91.00%**	**96.67%**	**97.50%**
Random Forest	91.00%	93.33%	95.00%
CNN	86.50%	91.67%	95.00%
SVM (rbf kernel)	80.50%	82.50%	85.00%
kNN	76.00%	83.33%	92.50%

Table 6: Comparison of test accuracy on IMDB

gcForest	**89.16%**
CNN	89.02% [Kim, 2014]
MLP	88.04%
Logistic Regression	88.62%
SVM (linear kernel)	87.56%
Random Forest	85.32%

[1] Zhi-Hua Zhou, and Ji Feng, Deep Forest: Towards an Alternative to Deep Neural Networks, *International Joint Conference on Artificial Intelligence (IJCAI-17)*

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Transfer Learning with Random Forest [1]

- A model *M* trained using a very large and expensive dataset *D*, is sold to a consumer with dataset *D'*. Due to memory constraints or data privacy or any other reasons, *D* cannot be shared. Even though *M* is very accurate over source *D*, it might not be that much reliable for target *D'*. One viable solution is Model Transfer or domain adaptation.

- Presented here in the case of Decision Trees (Random Forest). A trained RF using *D* is optimized using *D'*.

- Two algorithms:
  1. Structure Expansion/Reduction (SER)
  2. Structure Transfer (STRUT)

[1] Segev, N., Harel, M., Mannor, S., Crammer, K., & El-Yaniv, R. (2017). Learn on source, refine on target: a model transfer learning framework with random forests. *IEEE transactions on pattern analysis and machine intelligence*, 39(9), 1811-1824.

Transfer Learning with Random Forest

Structure Expansion/Reduction (SER)

- In the expansion transformation, specialize rules induced over the source data to the target data. That means, expand the leaves of the trained RF with respect to the samples of the target data D'.

- In the reduction transformation, perform the opposite operation, i.e. generalize rules induced over the source data. Working bottom-up, delete the node for whose *leaf error*<*subtree error* where *leaf error* is the empirical error if the node were to be pruned into a leaf and *subtree error* is the empirical error of the subtree whose root is the current node.

- Such design serves as a kind of regularization that keeps the resulting target model closer to the source model than it would if reduction were to precede expansion.

Transfer Learning with Random Forest

We use the following notations for a tree in the forest: Each tree node v has an out-degree $d(v)$ and its children are denoted $v_1, \ldots, v_{d(v)}$. A leaf node v is associated with a single decision value in $\mathcal{Y}$, denoted $y(v)$. An internal (non-leaf) node v is associated with a single feature $\phi(v)$, and for a numeric feature it is also associated with a numeric threshold $\tau(v)$. Classification of a sample is done based on the leaf that sample reaches, i.e., the leaf at the end of the path in the tree the sample will follow, and for each node u along this path we say that x "arrives" at u.

Algorithm 1. Structure Expansion Reduction (SER)

Input: Node v, labeled samples S_v
Output: Node v
% Expand leaves and Recurse over child nodes:
if $d(v) = 0$ **then**
  $v \leftarrow$ Build Tree (S_v)
  return v
end if
for $v_i \in \{v_1, \ldots, v_n\}$ **do**
  Structure Expansion Reduction (v_i, S_{v_i})
end for
% Reduce current node:
if leafError$(v, (S_v))$ < subtreeError$(v, (S_v))$ **then**
  for $i \in d(v)$ **do**
    deleteNode(v_i)
  end for
  $d(v) \leftarrow 0$; $y(v) \leftarrow \arg\max_y |\{(\cdot, y) \in S_v\}|$
end if

Structure Transfer (STRUT)

- Decision trees for similar problems should exhibit structural similarity.
- The STRUT algorithm adapts a DT trained on the source samples to the target samples by discarding all numeric threshold values of a node in the tree and working top-down, selecting a new threshold for a node using the subset of target examples that reach a particular node.
- Divergence Gain (DG) is used in conjunction with Information Gain (IG) to select the best feature and threshold. DG is used to ensure that the new label distribution in the left and right child nodes Q'_L and Q'_R are similar to the original distributions Q_L and Q_R.

DG relies on the (symmetric) Jensen-Shannon divergence given in Equation (2), where $D_{KL}(\cdot)$ is the familiar Kullback-Leibler divergence and M is the mean distribution, $M = \frac{1}{2}(P+Q)$

The choice of the Jensen-Shannon divergence is justified by its frequent use as an effective statistic for the two-sample problem.

$$DG\left(S_v^T, \phi(v), \tau(v), Q_L, Q_R\right) = 1 - \frac{|S_{L,\tau}|}{|S_v^T|} JSD(Q'_L, Q_L) - \frac{|S_{R,\tau}|}{|S_v^T|} JSD(Q'_R, Q_R). \quad (1)$$

$$2JSD(P,Q) = D_{KL}(P||M) + D_{KL}(Q||M). \quad (2)$$

Any feature ϕ and threshold τ split any set of (labeled) examples, S, into two subsets, denoted $S_{L,\tau}$ and $S_{R,\tau}$. The label distributions over these subsets are denoted Q_L and Q_R, respectively.

- The final optimization problem is:

$$\begin{aligned} \underset{x}{\text{Maximize}} \quad & DG\left(S_v^T, \phi, x, Q_L, Q_R\right) \\ \text{s.t.} \quad & x \in \mathbb{R} \\ & \forall x' \in (x-\epsilon, x+\epsilon): \\ & \quad IG\left(S_v^T, \phi, x\right) \geq IG\left(S_v^T, \phi, x'\right). \end{aligned}$$

Transfer Learning with Random Forest

Generate two forests using both SER and STRUT and then use a voting ensemble.

Results show that transfer learning is better than using just the small target data to train an RF.

Algorithm 2. Structure Transfer (STRUT)

Input: Node v, labeled samples S
Output: Node v
% Prune unreachable subtree and update leaf distribution:
if $(|S| = 0)$ **then**
  $d(v) \leftarrow 0$
  return v
else if $(d(v) = 0)$ **then**
  $y(v) \leftarrow \arg\max_y |\{(\cdot, y) \in S\}|$
  return v
end if
% Refit thresholds for numeric features:
if ($\phi(v)$ is numeric) **then**
  $\tau_1 \leftarrow$ Threshold Selection$(S, \phi(v), Q_L(v), Q_R(v))$
  $\mathrm{DG}_1 = \mathrm{DG}\ (S, \phi(v), \tau_1, Q_L(v), Q_R(v))$
  $\tau_2 \leftarrow$ Threshold Selection$(S, \phi(v), Q_R(v), Q_L(v))$
  $\mathrm{DG}_2 = \mathrm{DG}\ (S, \phi(v), \tau_2, Q_R(v), Q_L(v))$
  if $(\mathrm{DG}_1 \geq \mathrm{DG}_2)$ **then**
    $\tau(v) \leftarrow \tau_1$
  else
    $\tau(v) \leftarrow \tau_2$
    swap(v_1, v_2)
  end if
end if
% Run STRUT on sons:
for ($i \in d(v)$) **do**
  STRUT (v_i, S_{v_i})
end for
return v

Transfer Learning with Random Forest

- **STRUT**, **SER** and **MIX** (combination of STRUT and SER) are the algorithms proposed in [1]. For details on other benchmarks and datasets, refer to [1].
- SrcOnly and TgtOnly are RF trained on source data *D* and target data *D'* respectively.

TABLE 4
Test Error Rates Compared to Benchmarks and Competing Algorithms—Lowest Error in Boldface

DATASET	SrcOnly	TgtOnly	relabeling	bias	pruning	ASVM	consensus	STRUT	SER	MIX
mushroom	15.2 ± 0.3	0.5 ± 0.1	2.1 ± 0.2	12.6 ± 0.5	14.1 ± 0.6	2.3 ± 0.2	0.6 ± 0.1	1.9 ± 0.2	$\mathbf{0.4 \pm 0.07}$	0.5 ± 0.08
letter	66.5 ± 0.4	19.3 ± 0.2	20.7 ± 0.3	63.4 ± 0.6	22.5 ± 0.4	33.8 ± 0.2	24.1 ± 0.4	21.0 ± 0.4	18.9 ± 0.2	$\mathbf{16.7 \pm 0.2}$
wine	66.6 ± 0.6	45.5 ± 0.3	44.9 ± 0.2	55.3 ± 0.2	$\mathbf{44.6 \pm 0.2}$	54.7 ± 0.4	44.3 ± 0.2	46.6 ± 0.3	45.8 ± 0.2	45.0 ± 0.3
digits	19.9 ± 0.05	3.0 ± 0.2	10.0 ± 0.0	19.9 ± 0.01	10.0 ± 0.0	10.0 ± 0.0	14.2 ± 0.9	5.4 ± 0.3	$\mathbf{2.9 \pm 0.2}$	3.8 ± 0.3
USPS	13.5 ± 0.0	14.9 ± 0.2	13.7 ± 0.1	13.7 ± 0.0	13.3 ± 0.1	78.2 ± 3.5	$\mathbf{11.6 \pm 0.1}$	15.6 ± 0.2	13.5 ± 0.1	13.3 ± 0.1
landmines	52.4 ± 0.1	41.0 ± 0.7	39.2 ± 0.5	52.1 ± 0.1	$\mathbf{38.2 \pm 0.2}$	43.4 ± 0.7	41.6 ± 0.7	40.4 ± 0.6	40.7 ± 0.4	40.4 ± 0.5
amazon-webcam	62.2 ± 0.1	74.6 ± 0.8	66.7 ± 0.7	65.4 ± 0.4	64.6 ± 0.7	88.0 ± 0.5	67.4 ± 0.7	71.3 ± 0.6	$\mathbf{62.0 \pm 0.6}$	64.6 ± 0.7
caltech-webcam	65.1 ± 0.4	74.6 ± 0.6	66.6 ± 0.4	67.6 ± 0.4	88.2 ± 0.3	68.6 ± 0.5	64.3 ± 0.4	71.3 ± 0.4	$\mathbf{63.3 \pm 0.4}$	64.6 ± 0.5
inversion(1%)	98.7 ± 0.2	54.2 ± 0.1	54.8 ± 5.5	96.2 ± 3.8	57.3 ± 5.8	92.4 ± 0.3	76.0 ± 0.3	$\mathbf{41.8 \pm 0.5}$	58.5 ± 0.3	44.1 ± 0.3
inversion(5%)	98.7 ± 0.2	28.2 ± 0.2	36.4 ± 3.7	96.9 ± 3.1	39.5 ± 4.0	92.4 ± 0.2	44.8 ± 0.3	$\mathbf{21.2 \pm 0.3}$	36.5 ± 0.2	22.2 ± 0.1
inversion(10%)	98.7 ± 0.2	20.5 ± 0.1	30.7 ± 3.1	97.0 ± 3.0	34.3 ± 3.5	92.4 ± 0.2	32.7 ± 0.2	$\mathbf{15.9 \pm 0.4}$	24.1 ± 0.3	16.4 ± 0.2
high-res(1%)	32.7 ± 0.2	54.2 ± 0.1	44.6 ± 4.5	$\mathbf{33.7 \pm 3.4}$	42.4 ± 4.3	90.2 ± 0.3	35.7 ± 0.5	48.2 ± 0.5	37.2 ± 0.3	38.5 ± 0.3
high-res(5%)	32.7 ± 0.2	28.2 ± 0.2	24.8 ± 2.5	32.5 ± 3.3	24.4 ± 2.5	90.2 ± 0.2	23.7 ± 0.3	28.3 ± 0.3	22.5 ± 0.2	$\mathbf{21.8 \pm 0.1}$
high-res(10%)	32.7 ± 0.2	20.5 ± 0.1	19.4 ± 2.0	32.3 ± 3.3	20.3 ± 2.0	90.2 ± 0.2	19.4 ± 0.2	22.9 ± 0.4	18.0 ± 0.3	$\mathbf{17.3 \pm 0.2}$
Activity(min)	11.5 ± 0.2	13.5 ± 0.2	11.4 ± 0.1	11.8 ± 0.02	11.2 ± 0.1	76.0 ± 0.2	14.4 ± 0.3	11.6 ± 0.2	11.2 ± 0.3	$\mathbf{11.1 \pm 0.2}$
Activity(median)	15.1 ± 0.3	16.2 ± 0.2	14.7 ± 0.1	15.6 ± 0.03	14.6 ± 0.1	76.9 ± 0.2	16.8 ± 0.1	14.6 ± 0.2	13.9 ± 0.1	$\mathbf{13.8 \pm 0.3}$
Activity(max)	15.4 ± 0.1	18.5 ± 0.3	14.7 ± 0.2	15.1 ± 0.04	14.4 ± 0.2	74.0 ± 0.1	18.1 ± 0.2	15.0 ± 0.1	15.0 ± 0.2	$\mathbf{14.2 \pm 0.2}$

[1] Segev, N., Harel, M., Mannor, S., Crammer, K., & El-Yaniv, R. (2017). Learn on source, refine on target: a model transfer learning framework with random forests. *IEEE transactions on pattern analysis and machine intelligence*, 39(9), 1811-1824.

Outline of RF

Random Forest and its Variants

- Random Forest (RaF)
- Rotation Forest
- Oblique Random Forest (obRaF)
- Hybridizing RVFL and RF
- Heterogeneous ObRaF
- Extra Trees
- Boosted Trees (AdaBoost, Gradient Boost)
- Deep Random Forest
- Random Forest with Transfer Learning
- Online Random Forest

Online Random Forest

- Online Random Forest is used in various computer vision tasks such as tracking. Here, we focus on the online Random Forest algorithm without delving deep into their applications.

- Online Training of the Decision Tree[1]
- Each node of the tree is associated with a split function that partitions the parent node into two child nodes. The split function and threshold is determined via a gain function defined as:

$$\Delta E_{j,p} = E(X_j) - \frac{|X_{j,p,l}|}{|X_j|} \cdot E(X_{j,p,l}) - \frac{|X_{j,p,r}|}{|X_j|} \cdot E(X_{j,p,r})$$

where $E(\cdot)$ denotes the Shannon entropy, X_j denotes the training samples arriving at node j, $X_{j,p,l}$ and $X_{j,p,r}$ are the left and right partitions made by split $(g_p(x), \theta_p)$ and $|\cdot|$ denotes the number of samples in it. Note that $\Delta E_{j,s} \geq 0$.

[1] W. Wang, C. Wang, S. Liu, T. Zhang and X. Cao, Robust Target Tracking by Online Random Forests and Superpixels, in *IEEE Transactions on Circuits and Systems for Video Technology*, July 2018.

Online Random Forest

- The Shannon entropy is defined as:

$$E(X) = \frac{1}{|X|} \sum_i (y_i - \mu)^2$$

where $\mu = \frac{1}{|X|} \sum_i (y_i)$ is the average value of labels in X.

- Training samples arrive one-by-one and the tree must be updated as soon as data arrive.
- For each leaf node, a triplet $\{n, \mu_n, E_n\}$ is maintained. When a new sample arrives, the average value and Shannon entropy of a leaf node or a partition are updated as shown in the next slide.

Online Random Forest

$$\mu_{n+1} = (n \cdot \mu_n + y_{n+1})/(n+1) \tag{5}$$

$$E_{n+1} = \frac{1}{n+1}\sum_{i=1}^{n+1}(y_i - \mu_{n+1})^2 \tag{6}$$

where the sum item in Eq. 6 is rewritten as follows,

$$\sum_{i=1}^{n+1}(y_i - \mu_{n+1})^2 = \sum_{i=1}^{n}(y_i - \mu_{n+1})^2 + (y_{n+1} - \mu_{n+1})^2 \tag{7}$$

$$(y_i - \mu_{n+1})^2 = (y_i - \mu_n)^2 - 2\triangle\mu \cdot (y_i - \mu_n) + (\triangle\mu)^2 \tag{8}$$

where $\triangle\mu = \mu_{n+1} - \mu_n$. We obtain,

$$E_{n+1} = \frac{1}{n+1}\{\sum_{i=1}^{n}(y_i - \mu_n)^2 + 2\triangle\mu \cdot \sum_{i=1}^{n}(y_i - \mu_n) + n \cdot (\triangle\mu)^2 + (y_{n+1} - \mu_{n+1})^2\} \tag{9}$$

the first and second items in the brace are equal to nE_n and zero, respectively. We employ Eq. 5 and Eq. 9 to update the node triples and their splits states.

Following Eq. 5-9, we maintain a triple set that consists of $\{n, \mu_n, E_n\}_j$ for leaf node j, and several $\{n, \mu_n, E_n\}_{j,p,l}$ and $\{n, \mu_n, E_n\}_{j,p,r}$ for each candidate split of node j. Gain function $\Delta E_{j,p}$ is conveniently computed. We then introduce two parameters: 1) n_{split}, which is the minimum number of samples a leaf node has to see before splitting, and 2) δ_{split}, which is the minimum gain a split has to achieve before splitting, as [44]. When $|X_j| > n_{split}$ and existing split $(g_p(x), \theta_p)$ satisfies $\Delta E_{j,p} > \delta_{split}$, the leaf node j is splited using $(g_p(x), \theta_p)$.

Online Random Forest

Algorithm 1 Online Random Forest Regressor

Initialization: Size of the forest: M; minimum number of samples: n_{split}; the minimum gain: δ_{split}.
Input: Random Forest; sample (x, y).
01. **for** $m = 1$ to M **do** //update each tree
02. $d = \text{Possion}(1)$
03. **Repeat** d times:
04. spread x to a leaf node j of tree m
05. update $\{n, \mu_n, E_n\}_j$ for leaf node j and $\{n, \mu_n, E_n\}_{j,p,l}$ or $\{n, \mu_n, E_n\}_{j,p,r}$ for each split using Eq. 5-9
06. **if** $|X_j| > n_{split}$ and $\exists$ split *s.t.* $\Delta E_{j,p} > \delta_{split}$ **then**
07. split node j, use the split $(g_p(x), \theta_p) = \arg\max_p(\Delta E_{j,p})$
08. create left and right children as new leaf node.
09. **end if**
10. **end Repeat**
11. **end for**
Output: Random Forest.

Thank You !

Questions?