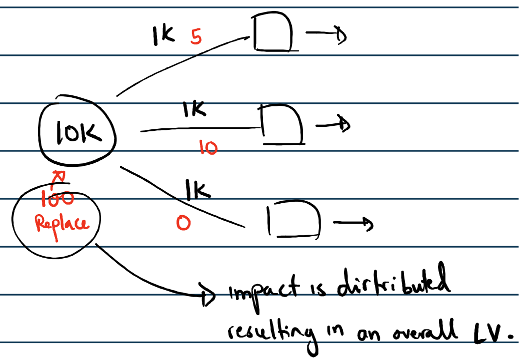
B(ootstrap)AGG(regation)ING

Saturday 4 May 2024 4:49 PM



→ Designed to improve the stability & accuracy of ML algos used in statistical classification & regression.

→ Helps avoid overfitting Low Bias High Variance (Deep trees) ⟶ Low Bias Low Variance

→ Core idea: generate multiple subsets of the original data (with replacement), train a separate model for each subset, & then combine the results.

→ Types of Bagging depends on the method chosen to create subsets of data

1) Bootstraping (row sampling with replacement)
2) Pasting (row sampling without replacement)
3) Random subspaces (column sampling with/without replacement)
4) Random patches (row & column sampling)

use when dealing with high dimensional data

class sklearn.ensemble.**BaggingClassifier**(*estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*) [source]

→ OOB score - out of bag samples. When we perform row sampling with replacement there is a chance that some rows have never been used during training. We can use these rows to check the performance of our model.

class sklearn.ensemble.**BaggingRegressor**(*estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0*) [source]

→ Parallel learning

Random Forest

Saturday 4 May 2024 5:24 PM

→ Bagging technique which involves training many individual decision trees & combining their outputs to make a final prediction.

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*) [source]

class sklearn.ensemble.**RandomForestRegressor**(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*) [source]

→ The difference b/w bagging & random forest:

1) Bagging with DT as Base model with col sampling

→ Sampling done at tree level

2) Random Forest with col sampling

↓ Bias

→ sampling done at node level (extra randomness) ("more wisdom", "more diverse crowd")

(at each node a random subset of columns are picked & we select the best feature from the subset to perform the split)

→ can be used for feature selection

Assumption.

→ Wisdom of the crowd [base models are independent of each other] [decorrelated models]

→ RF's success depends on the randomness of the system. [How decorrelated each base model is from each other]

→ Black Box model

→ Extremely Randomized Trees → Splitting is done at random without calculations

sklearn.ensemble.ExtraTreesClassifier

class sklearn.ensemble.**ExtraTreesClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=False, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None, monotonic_cst=None*) [source]

* feature importance reliable??

Pros & Cons

- → Robustness to overfitting
- → Handling large datasets
- → Less pre-processing
- → Variable importance
- → Parallelizable
- → Non-parametric

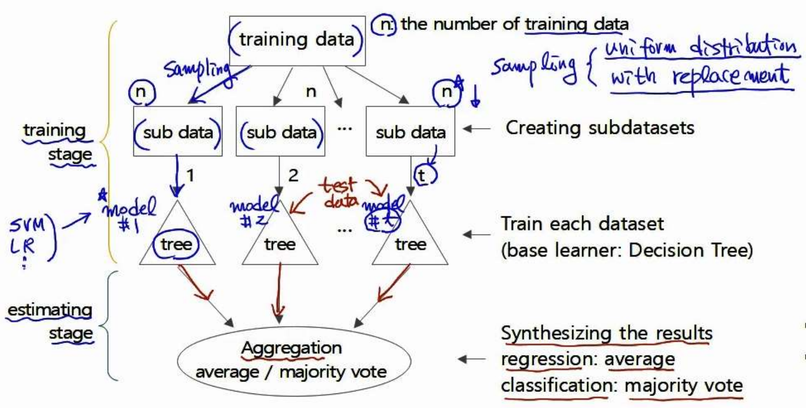
- → Model interpretability
- → Performance with Unbalanced Data
- → Predictive performance
- → Inefficiency with sparse data
- → Parameter tuning
- → Difficulty with High Cardinality Features
- → Can't Extrapolate

RF Revisited

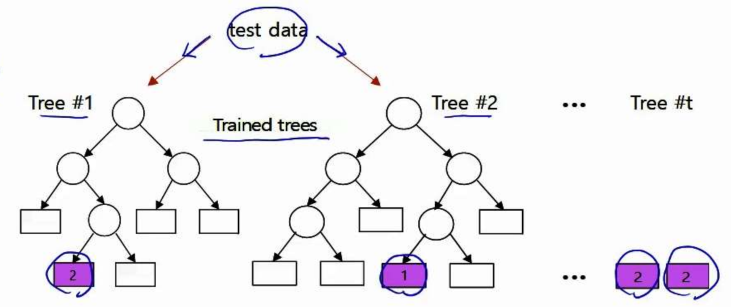
Friday 24 May 2024 3:47 AM

▪ Random Forest : Bootstrap Aggregation (Bagging)

- Random Forest uses multiple Decision Trees. Each tree is then built based on randomly and uniformly drawn samples with replacement for the training data.
- Subdatasets are created from samples extracted from the training data. And the size of each subdataset is equal to the size of the training data.
- In the figure below, the number of subdatasets and the number of trees is t, and the number of the training data is n.
- Samples are drawn from rows and columns of the training data. Column sampling is the random selection of features. This results in lower correlation between trees.
- Each tree is grown to the largest extent possible. There is no pruning. (Reference (2)). Random Forest uses multiple deep decision trees, but it is less prone to overfitting because it uses sample data, and average the results of the trees. This is why pruning is not necessary.
- After training, test data is inserted into each tree and the results are synthesized. For regression, it is estimated as the average of each tree's results, and for classification, it is estimated as the most frequent class of each tree's results. (majority voting).



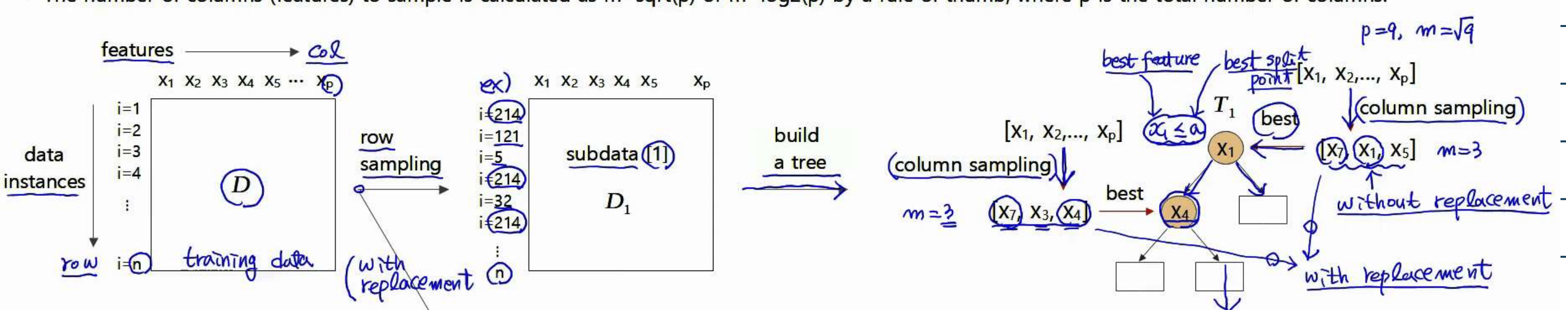
* General structure of bagging (parallel structure)

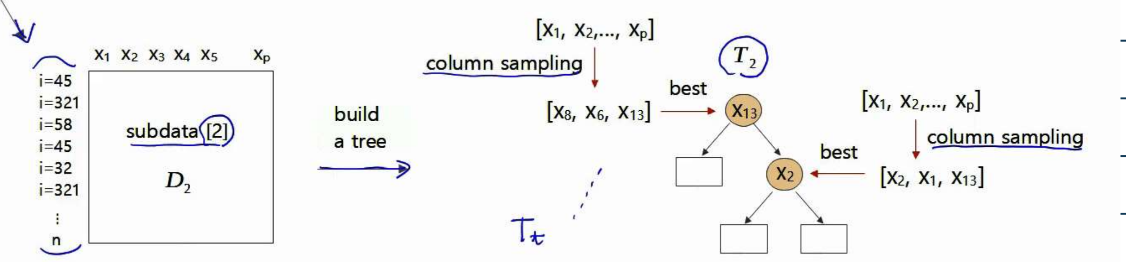


- If you insert one test data point into T trained trees, you can get t results.
- In classification, if the results are [2, 1, 2, 2, ...] as shown above, the class of the test data point is assumed to be 2 because 2 is the majority.
- In regression, it is estimated as the average value of the results.

▪ Data Sampling : row (data instances) and column (features) sampling

- The training data (D) is sampled row-wise and column-wise. The reason for sampling is to reduce the correlation between each tree, thus reducing the estimation variance.
- Row sampling is done with replacement and column sampling is done without replacement but after sampling, all are replaced for the next sampling. That is, column sampling for node splitting is done without replacement, but with replacement within an individual tree.
- The number of columns (features) to sample is calculated as m=sqrt(p) or m=log2(p) by a rule of thumb, where p is the total number of columns.





⊙ The number of features to sample

$m = \sqrt{p}$, $\log_2 p$ (rule of thumb)

where p is the total number of columns

- **Row sampling** with replacement (bootstrap samples) reduces the correlation between the decision trees. Without row sampling, the results of many trees can be similar, reducing ensemble effects. **Column sampling** (sampling features) can further reduce correlation between trees. Without column sampling, if there are a few key features, these features are selected from many trees, making the trees similar (correlated) again.

$$\hat{y} = \frac{1}{t}\sum_{i=1}^{t} T_i(x)$$ ~ for regression

output; input: test data point

Var

$$Var(\hat{y}) = Var\left(\frac{1}{t}\sum_{i=1}^{t} T_i(x)\right) = \frac{1}{t^2}Var\left(T_1(x) + T_2(x) + \cdots\right)$$

$$= \frac{1}{t^2}\sum_{i=1}^{t}\sum_{j=1}^{t} Cov\left(T_i(x),\ T_j(x)\right)$$

$$= \frac{1}{t^2}\sum_{i=1}^{t}\Big(\sum_{j\neq i}^{t} \underbrace{Cov\left(T_i(x),T_j(x)\right)}_{i\neq j} + \underbrace{Var\left(T_i(x)\right)}_{i=j}\Big)$$

assumption: σ^2 constant

$$Var(\hat{y}) = \frac{1}{t^2}\sum_{i=1}^{t}\left((t-1)\sigma^2\rho + \sigma^2\right) \qquad \rho = \frac{Cov(x,y)}{\sigma_x\sigma_y} \to \sigma^2$$

$Cov(\cdot) = \rho\,\sigma^2$

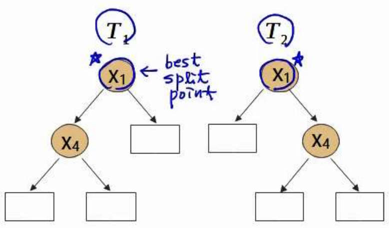
$$= \left(\frac{t(t-1)\rho\sigma^2 + t\sigma^2}{t^2}\right)$$

$$Var(\hat{y}) = \rho\sigma^2 + \sigma^2\frac{1-\rho}{t}$$

- Row/column sampling reduces ρ between trees, making Var(ŷ) smaller. Also, the larger the number of trees, t, the smaller Var(y) becomes.
- The smaller m is, the smaller ρ is.

⊙ **Without column sampling:**

- If x1 is the most important feature, it is likely to be used as the first split point for many trees, even if the data varies depending on row sampling. Then the trees will be similar.



Reference: https://stat.ethz.ch/education/semesters/ss2012/ams/slides/v10.2.pdf (Applied Multivariate Statistics – Spring 2012)

▪ Out-Of-Bag (OOB) score (or error rate)

Selected: In-Of-Bag (IOB)

- Data points that are not selected by row subsampling are called Out-of-Bag (OOB) data. OOB data can be used to evaluate the performance of the model.
- Using OOB score eliminates the need for cross-validation. No need to use a validation dataset.
- If you have a lot of data available, you can use a separate validation dataset to evaluate your model's performance, but if you have less data, you can use OOB data to evaluate its performance. OOB increases the efficiency of data use.

⟨Training data⟩

	x_1	x_2	x_3	x_4
i=1				
i=2				
i=3				
i=4				
i=5				

Data points selected for each tree

T_1	T_2	T_3	T_4	T_5	T_6
4	1	3	1	3	5
2	5	3	5	2	5
3	5	2	1	1	1
2	1	2	2	1	2
4	1	4	2	5	2

data point ID ← IOB data points

OOB data points

T_1	T_2	T_3	T_4	T_5	T_6
1	2	1	3	4	3
5	3	5	4		4
	4				

OOB tree list

	Tree list			
i=1	T_1	T_3		
i=2	T_2			
i=3	T_2	T_4	T_6	
i=4	T_2	T_4	T_5	T_6
i=5	T_1	T_3		

- i=1 data point is used to evaluate T_1 and T_3
- i=2 data point is used to evaluate T_2
- i=3 data point is used to evaluate T_2, T_4, T_6
- i=4 data point is used for T_2, T_4, T_5, T_6
- i=5 data point is used for T_1, T_3

- The probability that the subset does not contain the original data. (Out-Of-Bag probability)
- the number of subsets is n, and the number of data points in a subset is also n
- The probability that
  - data instance i will be selected when selecting a sample: $\frac{1}{n}$
  - data instance i will not be selected when selecting a sample: $\left(1-\frac{1}{n}\right)$
  - data instance i will not be selected at all while selecting n samples: $\left(1-\frac{1}{n}\right)^n$
- If n is large enough: $\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n=\frac{1}{e}=0.3679$ ← OOB probability

OOB tree map

count Σ		T_1	T_2	T_3	T_4	T_5	T_6
2	i=1	1		1			
1	i=2		1				
3	i=3		1		1		1
4	i=4		1		1	1	1
2	i=5	1		1			

2/5 3/5

OOB prediction map

	T_1	T_2	T_3	T_4	T_5	T_6
i=1	$T_1(x_1)$		$T_3(x_1)$			
i=2		$T_2(x_2)$	~~$T_3(x_2)$~~			
i=3		$T_2(x_3)$		$T_4(x_3)$		$T_6(x_3)$
i=4		$T_2(x_4)$		$T_4(x_4)$	$T_5(x_4)$	$T_6(x_4)$
i=5	$T_1(x_5)$		$T_3(x_5)$			

(final prediction)

$\frac{1}{2}\sum_{i\in\{1,3\}} T_i(x_1)$

$\frac{1}{3}\sum_{i\in\{2,4,6\}} T_i(x_3)$

▪ Missing value : Proximity Matrix

tree-1: 0 1 2 3

- If there are missing values in the data, they can be estimated through Random Forest. The idea is to estimate the missing values in data points by referring to the values in similar data points. Proximity matrix (PM) is used to measure the similarity. (1 – PM) is distance matrix.
- Proximity matrix (PM) is generated by the following procedure. When normalizing PM, it is divided by the number of trees. Here, it is normalized so that the sum of the columns is 1, as shown in the bottom right. This is to create weights for a weighted average to be used later.

x — 4 features (Categorical / Numeric); target y

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
i=1	No	No	No	125	No
i=2	Yes	Yes	Yes	180	Yes
i=3	Yes	Yes	No	210	No
i=4	Yes	Yes	No	167.5	No

The first round of training, leaf indices in each tree

tree 1	tree 2	tree 3	···	tree 10
3	3	1		
0	0	0		
1	0	2		
1	0	2		

missing values — most common value / median or average value → initial guess

Only data points where y=No are considered.

```
model=RandomForestClassifier(...)
model.fit(x, y)
leaf_id = model.apply(x)
```

* apply trees in the forest to x, return leaf indices.

The data point number 3 and 4 are in the same leaf node in 3 trees. This means that the 2 data points are similar.

* Reference: [StatQuest] Random Forests Part 2: Missing data and clustering

proximity matrix of tree-1

	i=1	i=2	i=3	i=4
i=1	0			
i=2		0		
i=3			0	1
i=4			1	0

```
t = np.array([3, 0, 1, 1])
pm = np.equal.outer(t, t) * 1
np.fill_diagonal(pm, 0)
```

proximity matrix of tree-2

	i=1	i=2	i=3	i=4
i=1				
i=2			1	1
i=3		1		1
i=4		1	1	

```
t = np.array([3, 0, 0, 0])
pm = np.equal.outer(t, t) * 1
np.fill_diagonal(pm, 0)
```

proximity matrix of tree-3

	i=1	i=2	i=3	i=4
i=1				
i=2				
i=3				1
i=4			1	

```
t = np.array([1, 0, 2, 2])
pm = np.equal.outer(t, t) * 1
np.fill_diagonal(pm, 0)
```

Σ

	i=1	i=2	i=3	i=4
i=1				
i=2			1	1
i=3		1		3
i=4		1	3	

element-wise sum of the proximity matrices from tree-1 to tree-3.

	i=1	i=2	i=3	i=4
i=1		2	1	1
i=2	2		1	1
i=3	1	1		8
i=4	1	1	8	

If element-wise sum of the proximity matrices from tree-1 to tree-10 is:

	i=1	i=2	i=3	i=4
i=1	0	0.5	0.1	0.1
i=2	0.5	0	0.1	0.1
i=3	0.25	0.25	0	0.8
i=4	0.25	0.25	0.8	0
Σ	1	1	1	1

normalization of column values

Proximity Matrix

▪ Missing value imputation – training data

- Missing values are estimated using the proximity matrix (PM).
- Continuous variable is estimated as a weighted average using PM, and categorical variable is estimated as a category with a large probability times weight.
- Update the proximity matrix using the estimated missing values and repeat this process until there are no changes.

x, y

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
i=1	No	No	No	125	No
i=2	Yes	Yes	Yes	180	Yes
i=3	Yes	Yes	No	210	No
i=4	Yes	Yes	No	167.5	No

missing values — initial guess

198.5 → train → PM; → train → PM; repeat

The first round. **Proximity Matrix (pm)**

	i=1	i=2	i=3	i=4
i=1	0	0.5	0.1	0.1
i=2	0.5	0	0.1	0.1
i=3	0.25	0.25	0	0.8
i=4	0.25	0.25	0.8	0
Σ	1	1	1	1

Continuous variable: Weight

np.dot(x[:, 3].reshape(1, -1), pm)

"weight" vector

$$[125\ \ 180\ \ 210\ \ 167.5]\begin{array}{c}\begin{array}{cccc}i{=}1 & i{=}2 & i{=}3 & i{=}4\end{array}\\ \begin{bmatrix}0.00 & 0.50 & 0.10 & 0.10\\ 0.50 & 0.00 & 0.10 & 0.10\\ 0.25 & 0.25 & 0.00 & 0.80\\ 0.25 & 0.25 & 0.80 & 0.00\end{bmatrix}\end{array}=[184.38\ \ 156.88\ \ 164.5\ \ 198.5]$$

PM

The "weight" value for data point number 4 is assumed to be this value.

125 * 0.1 + 180 * 0.1 + 210 * 0.8 + 167.5 * 0

Categorical variable: Blocked Arteries

For each category, calculate the probability times the weight. The probability is calculated by excluding missing values.

$\text{Yes}: \frac{1}{3}\times\frac{0.1}{0.1+0.1+0.8}=0.033$

$\text{No}: \frac{2}{3}\times\frac{0.1+0.8}{0.1+0.1+0.8}=0.6$

Since it is larger, the missing value is assumed to be No.

Because of normalization, the denominator is always 1.

▪ Missing value imputation – test data

Proximities

<skipped>

When a test set is present, the proximities of each case in the test set with each case in the training set can also be computed. The amount of additional computing is moderate.

Missing value replacement for the test set

When there is a test set, there are two different methods of replacement depending on whether labels exist for the test set.

If they do, then the fills derived from the training set are used as replacements. If labels no not exist, then each case in the test set is replicated nclass times (nclass= number of classes). The first replicate of a case is assumed to be class 1 and the class one fills used to replace missing values. The 2nd replicate is assumed class 2 and the class 2 fills used on it.

This augmented test set is run down the tree. In each set of replicates, the one receiving the most votes determines the class of the original case.

▪ **Continuous variable:**

training

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
i=1	No	No	No	125	No
i=2	Yes	Yes	Yes	180	Yes
i=3	Yes	Yes	No	210	No
i=4	Yes	Yes	No	198.5	No
test	Yes	No	No	178.4	?

missing value

use trained model

tree 1	tree 2	tree 3	...
3	3	1	
0	0	0	
1	0	2	
1	0	2	
1	1	2	

tree의 leaf-id matrix

	i=1	i=2	i=3	i=4	test
i=1	0	0.4	0.1	0.1	
i=2	0.4	0	0.1	0.1	
i=3	0.25	0.2	0	0.5	0.3
i=4	0.25	0.2	0.5	0	0.3
test	0.1	0.2	0.3	0.3	

Proximity Matrix (Normalized)

1. Initial guess. There is no y in the test data, so all data points, including the training data, are used.
2. Create a proximity matrix using all data points, including the training data.
3. Estimate the missing value using proximity matrix

* Please see the code in the following video for more details.

▪ **Categorical variable:**

label = [Yes, No]

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
test	Yes	No	?	125	?

Example from above reference

replicate

Since we have y, we estimate it in the same way as the train data.

(A)

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
test	Yes	No	Yes	125	Yes

NO

y_pred = [1, 1, 0, 0, 1] → Yes = 3/5

Y Y N

consider all cases

(B)

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
test	Yes	No	No	125	No

y_pred = [1, 1, 0, 1, 1] → No = 1/5

Use the model to predict replicated data, A and B, then select the one with higher accuracy in y_pred. In this example, the missing value is assumed to be "yes".

▪ Outlier Detection: Algorithm

- Random Forest's proximity matrix can be used to detect outliers.
- It is an outlier detection algorithm in the form of supervised learning.

Reference [2]: **Outliers**

Outliers are generally defined as cases that are removed from the main body of the data. Translate this as: outliers are cases whose proximities to all other cases in the data are generally small. A useful revision is to define outliers relative to their class. Thus, an outlier in class j is a case whose proximities to all other class j cases are small.

(1)

Define the **average proximity** from case n in class j to the rest of the training data class j as:

$$\bar{P}(n) = \sum_{class(k)=j} prox^2(n,k)$$

(2)

The **raw outlier measure** for case n is defined as

$$r(n) = \frac{nsample}{\bar{P}(n)} = 4$$

This will be large if the average proximity is small. Within each class find the median of these raw measures, and their absolute deviation from the median. Subtract the median from each raw measure, and divide by the absolute deviation to arrive at the **final outlier measure**. (3)

Proximity matrix (normalized by the number of trees)

case →

	n=1	n=2	n=3	n=4	class j: y
n=1		0.2	0.1	0.1	0
n=2	0.2		0.1	0.1	1
n=3	0.1	0.1		0.8	0
n=4	0.1	0.1	0.8		0

1) Average proximity

$$\bar{P}(n=1) = \sum_{class(k)=0} prox^2(1,k)$$

$$= \sum_{k=[3,4]} prox^2(1,k) = 0.1^2+0.1^2 = 0.02$$

$$\bar{P}(n=2) = \sum_{class(k)=1} prox^2(2,k) = 0$$

$$\bar{P}(n=3) = 0.1^2+0.8^2 = 0.65$$

$$\bar{P}(n=4) = 0.1^2+0.8^2 = 0.65$$

2) Raw outlier measure → **3) Final outlier measure**

$$r(n=1) = \frac{4}{0.02} = 200$$

$$r(n=2) = \frac{4}{0} = \infty$$

$$r(n=3) = \frac{4}{0.65} = 6.15$$

$$r(n=4) = \frac{4}{0.65} = 6.15$$

- Data with an excessively large r_f are considered outliers.

y=0: r(n) = [200, 6.15, 6.15] → median = m_0, absolute deviation = s_0

y=1: r(n) = [∞, ...] → median = m_1, absolute deviation = s_1

$$r_f(n=1) = \frac{r(n=1)-m_0}{s_0} \qquad r_f(n=3) = \frac{r(n=3)-m_0}{s_0}$$

$$r_f(n=2) = \frac{r(n=2)-m_1}{s_1} \qquad r_f(n=4) = \frac{r(n=4)-m_0}{s_0}$$

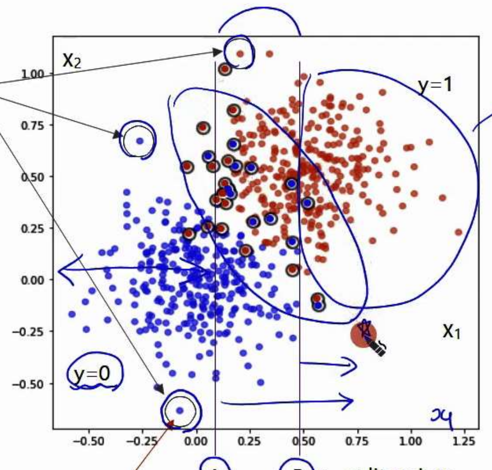
* Evaluate how far each data point is from the center of the normalized distribution.

▪ Outlier Detection: Interpretation of the result

- Let's consider why outliers are near the decision boundary.

Outlier detection algorithms in the form of unsupervised learning consider these data to be outliers. This is because they are far from the entire distribution.
(ex: Isolation Forest, Auto Encoder, etc)

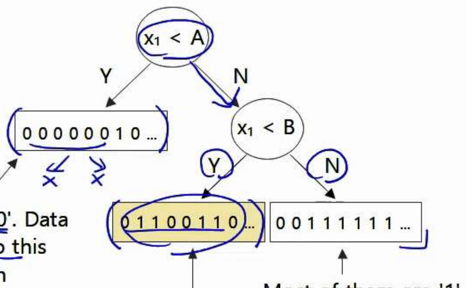
Random forest is a supervised learning method. Supervised learning type outlier detection algorithms take classes into account to determine outliers.



← split points

This data point is far from the y=0 cluster centroid, but belongs to the same leaf as the centroid, so it has high similarity to the centroid data. So it is not considered an outlier.

Within the decision boundary, no matter how far away from the center it is, it is not an error and therefore does not need to be treated as an outlier (in the "hinge loss" concept).



Most of them are '0'. Data points belonging to this leaf node have high similarity to each other. It doesn't matter how far they are from the cluster center.

Most of them are '1'.

Since there is a mix of '0' and '1', this node will be split again. Then these data points belong to different leaf nodes and their similarity (proximity) will be lowered. If the proximity becomes smaller, the outlier measure becomes larger, so they will be considered as outliers.