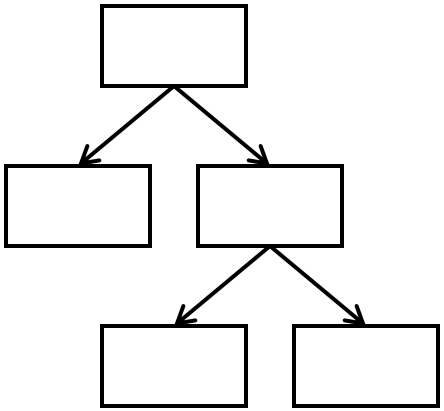
COMP 6321 Machine Learning

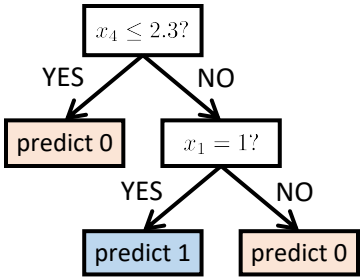
Decision Trees & Random Forests

Computer Science & Software Engineering
Concordia University, Fall 2020

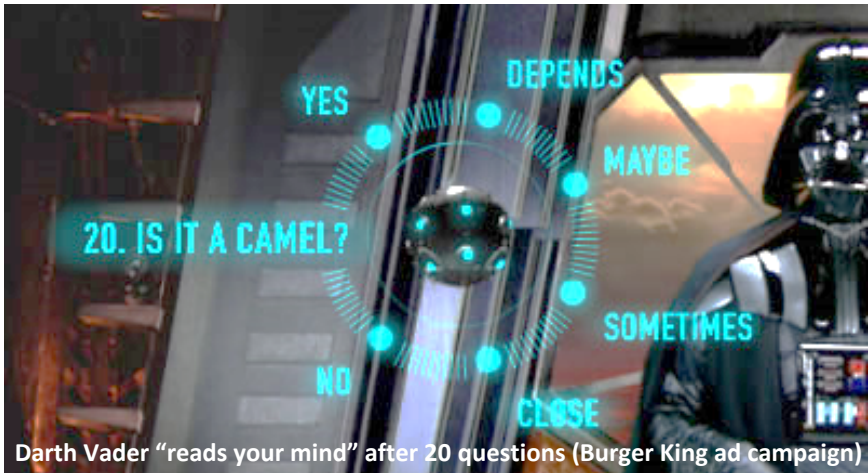
Decision Trees



Decision tree classifiers



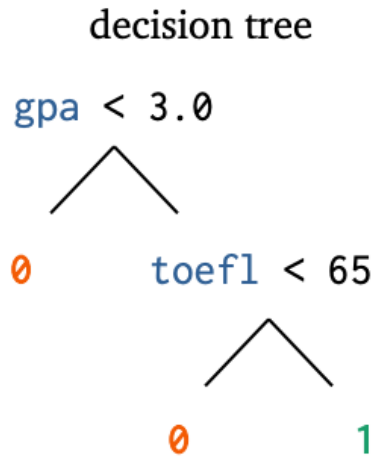
- **Idea:** Given a set of features $\mathbf{x}$, choose a feature x_j and perform a YES/NO test on it.
  - Each test provides information that will help us classify.
  - Each answer should influence the next question we ask!!
  - After using up our "budget" of tests, predict a class label.
- If tests are YES/NO, corresponds to a *binary tree*
- Prediction follows path to a leaf; each leaf has a class
- Basically plays a game of "20 questions" with the query point $\mathbf{x}$!

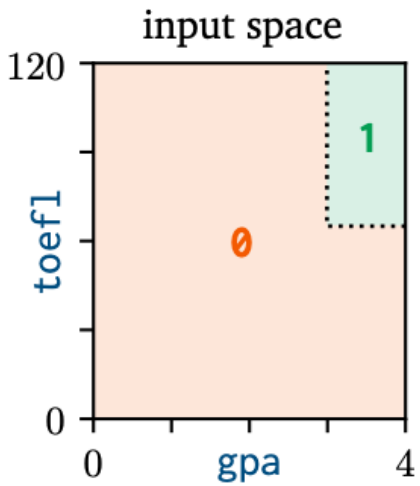


Darth Vader "reads your mind" after 20 questions (Burger King ad campaign)

- A decision tree is a program comprising nested *if-else* and *return* statements.
- The configuration of *if-else* and return statements is learned from data.
- Each *if* statement has a threshold parameter that must be learned from data.
- Each *if* statement typically tests a single input value.

```
int accept(float gpa, int toefl) {
    if (gpa < 3.0)
        return 0;          // REJECT
    else if (toefl < 65)
        return 0;          // REJECT
    else
        return 1;          // ACCEPT
}
```





Example tree

Some paths may not even require testing all the features, such as:
x = (yellow, thin, medium, sweet)

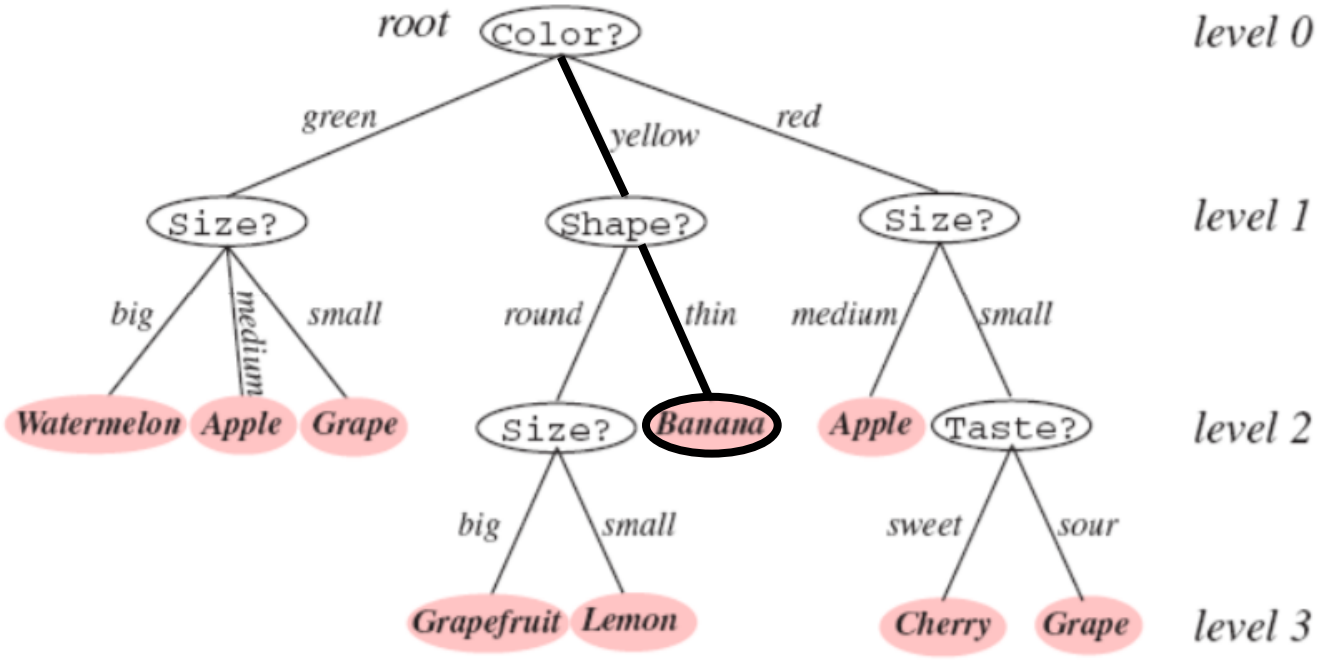
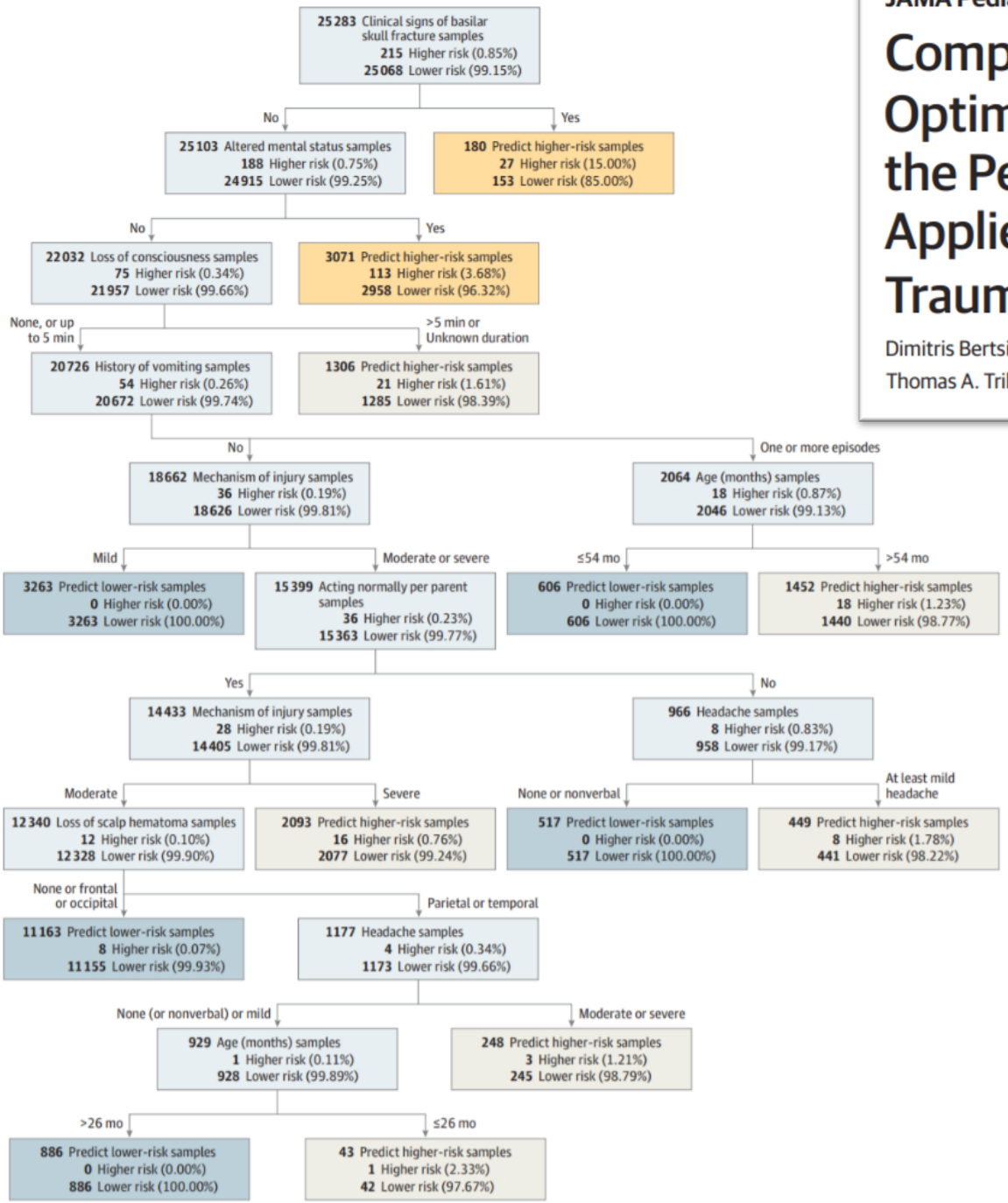


FIGURE 8.1. Classification in a basic decision tree proceeds from top to bottom. The questions asked at each node concern a particular property of the pattern, and the downward links correspond to the possible values. Successive nodes are visited until a terminal or leaf node is reached, where the category label is read. Note that the same question, `Size?`, appears in different places in the tree and that different questions can have different numbers of branches. Moreover, different leaf nodes, shown in pink, can be labeled by the same category (e.g., **Apple**). From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*.

Figure 2. Optimal Classification Tree for Clinically Important Traumatic Brain Injury in Children 2 Years or Older



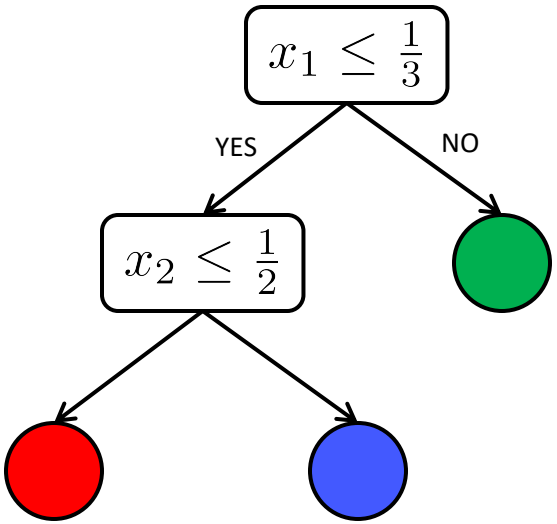
JAMA Pediatrics | Original Investigation

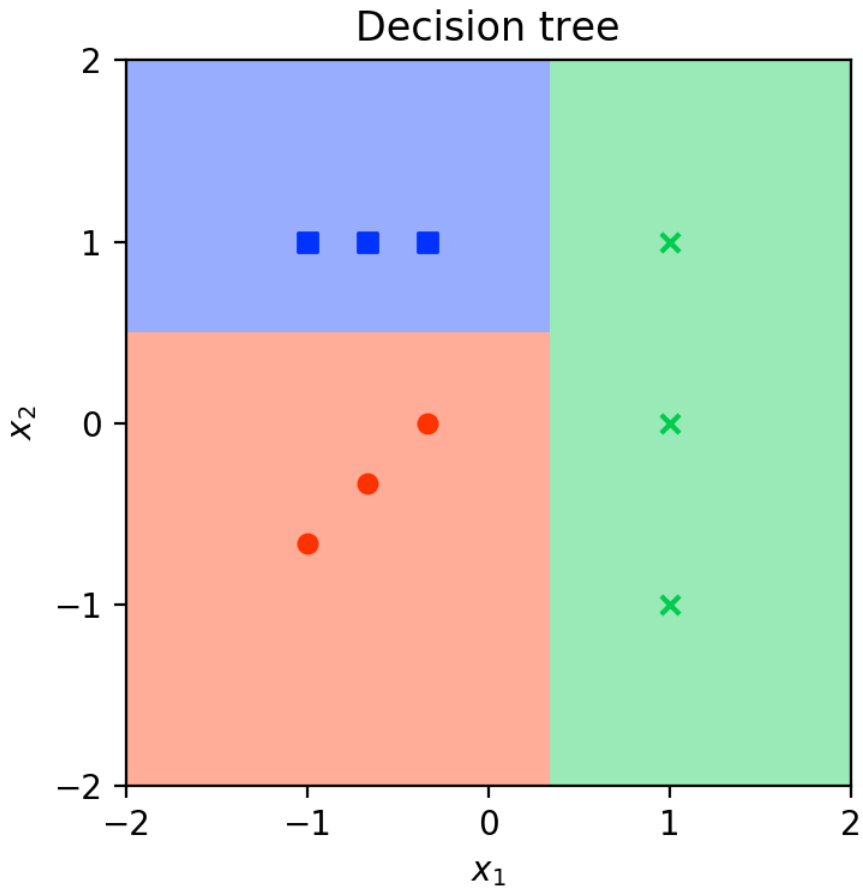
Comparison of Machine Learning Optimal Classification Trees With the Pediatric Emergency Care Applied Research Network Head Trauma Decision Rules

Dimitris Bertsimas, PhD; Jack Dunn, PhD; Dale W. Steele, MD, MSc; Thomas A. Trikalinos, MD; Yuchen Wang, BS

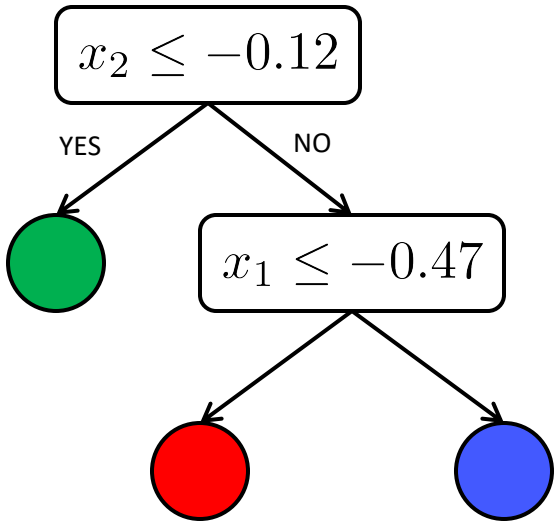

Image credit: Jim Araos

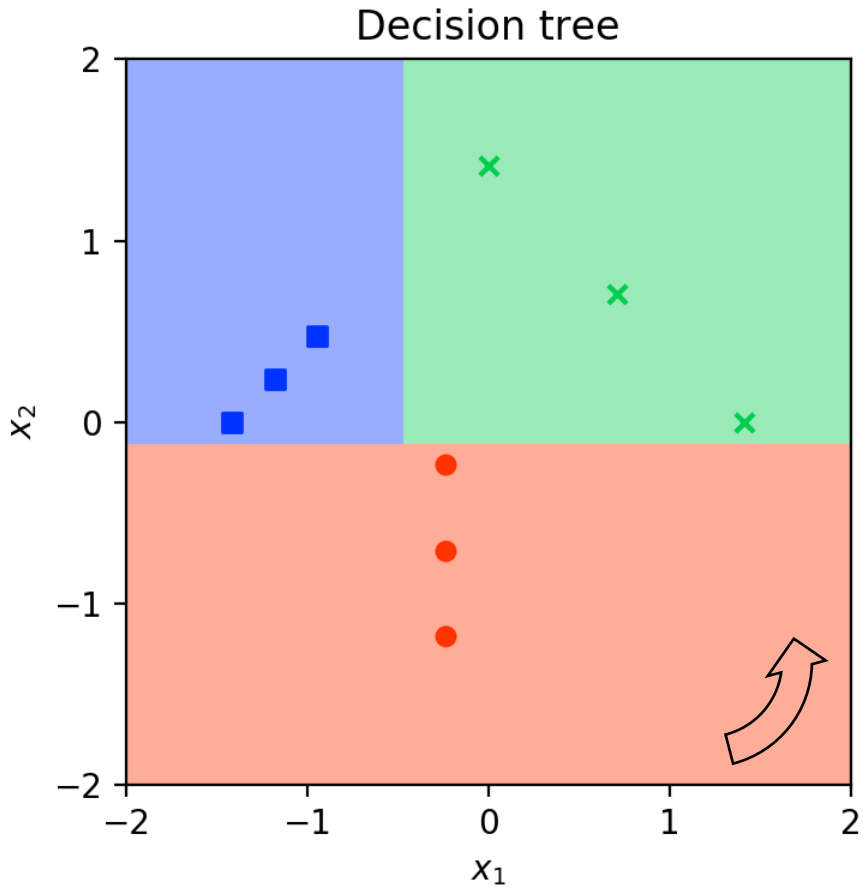
Example predictions





Example predictions





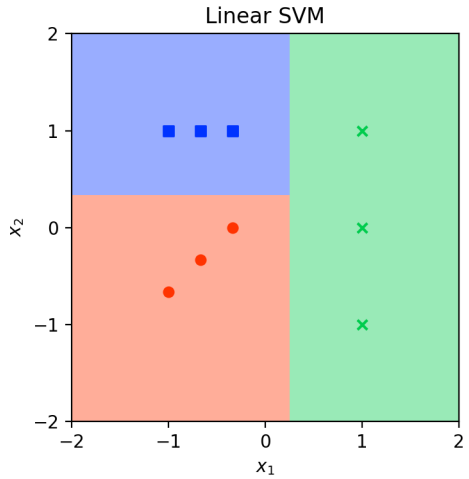
Decision tree classifiers

- **Intuitive predictions.** Small decision trees are interpretable, similar to how humans ask questions.
- **Fast predictions.** Each test is simple. Each prediction follows only one path in the tree.
- **Flexible.** Multi-class is easy. Handles missing features. Features x_j and $x_{j'}$ don't need to be comparable.
  - e.g. {teacher, police, counselor, ...} vs {$40000, $60000, ...}
- **Works well in practice.** The basis of several state of the art methods, including Random Forests.
- Considered **non-parametric**, unless we place hard limits on the complexity of the tree.

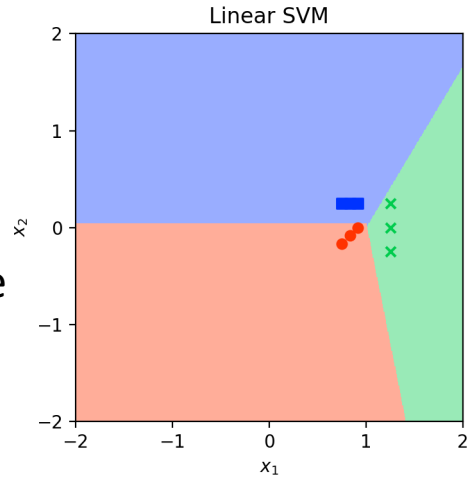
Decision tree classifiers

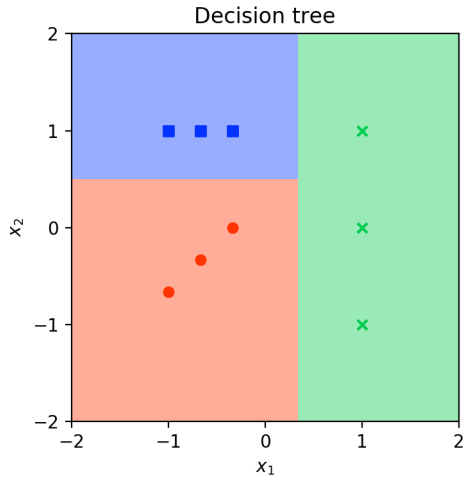
- **Insensitive to feature shifting/scaling.**

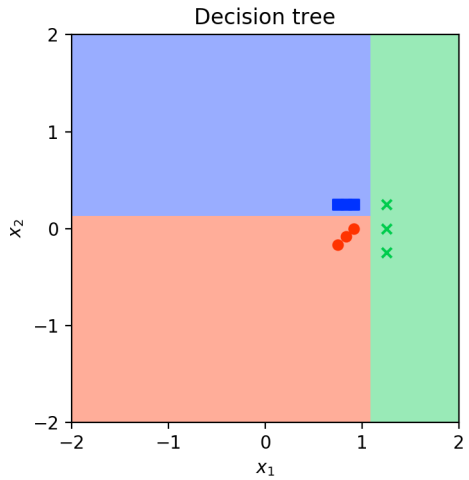
Why? Because tests one x_j at a time.



shift and scale training data



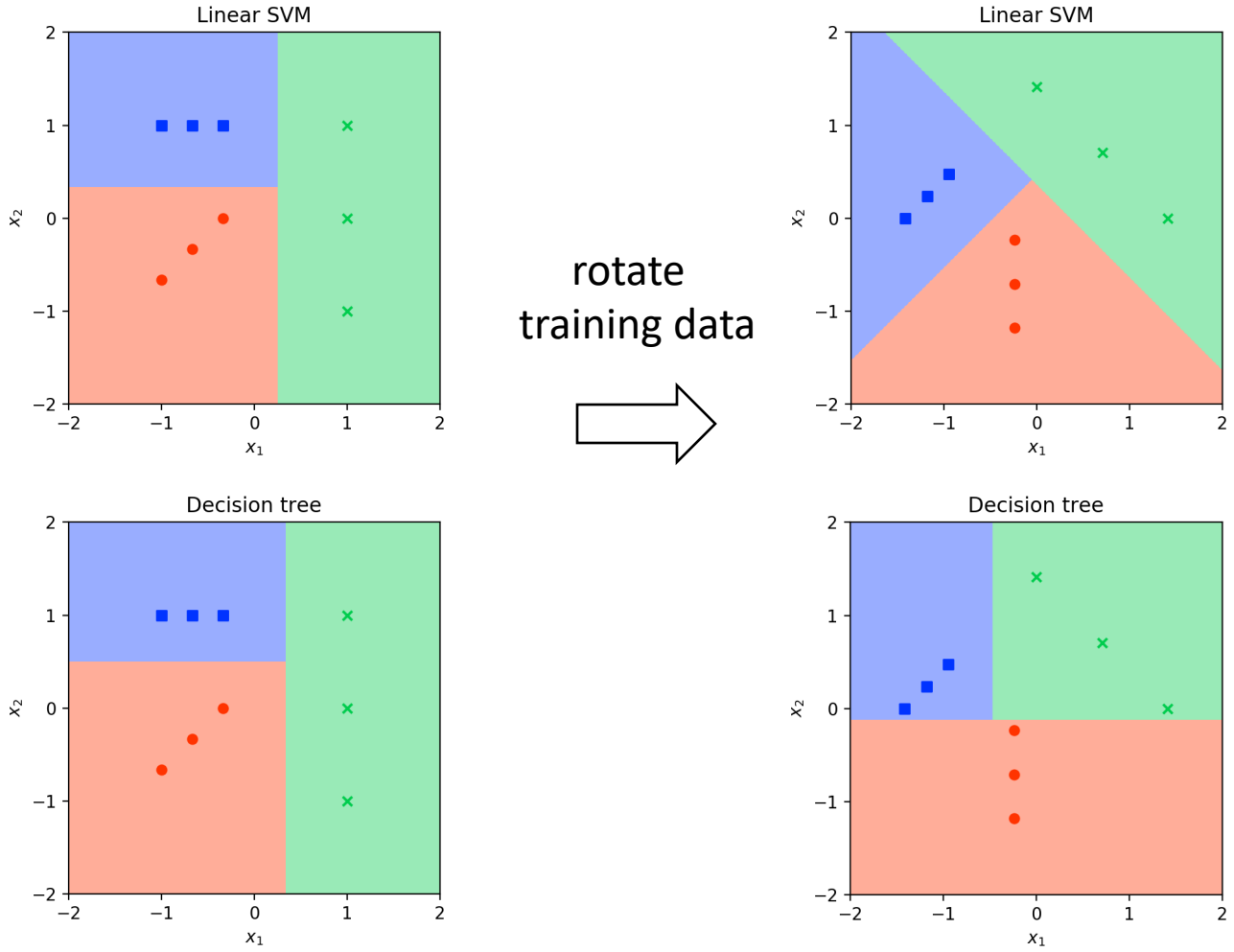




Decision tree classifiers

- **Sensitive to feature rotation.**

Why? Because tests one x_j at a time!!

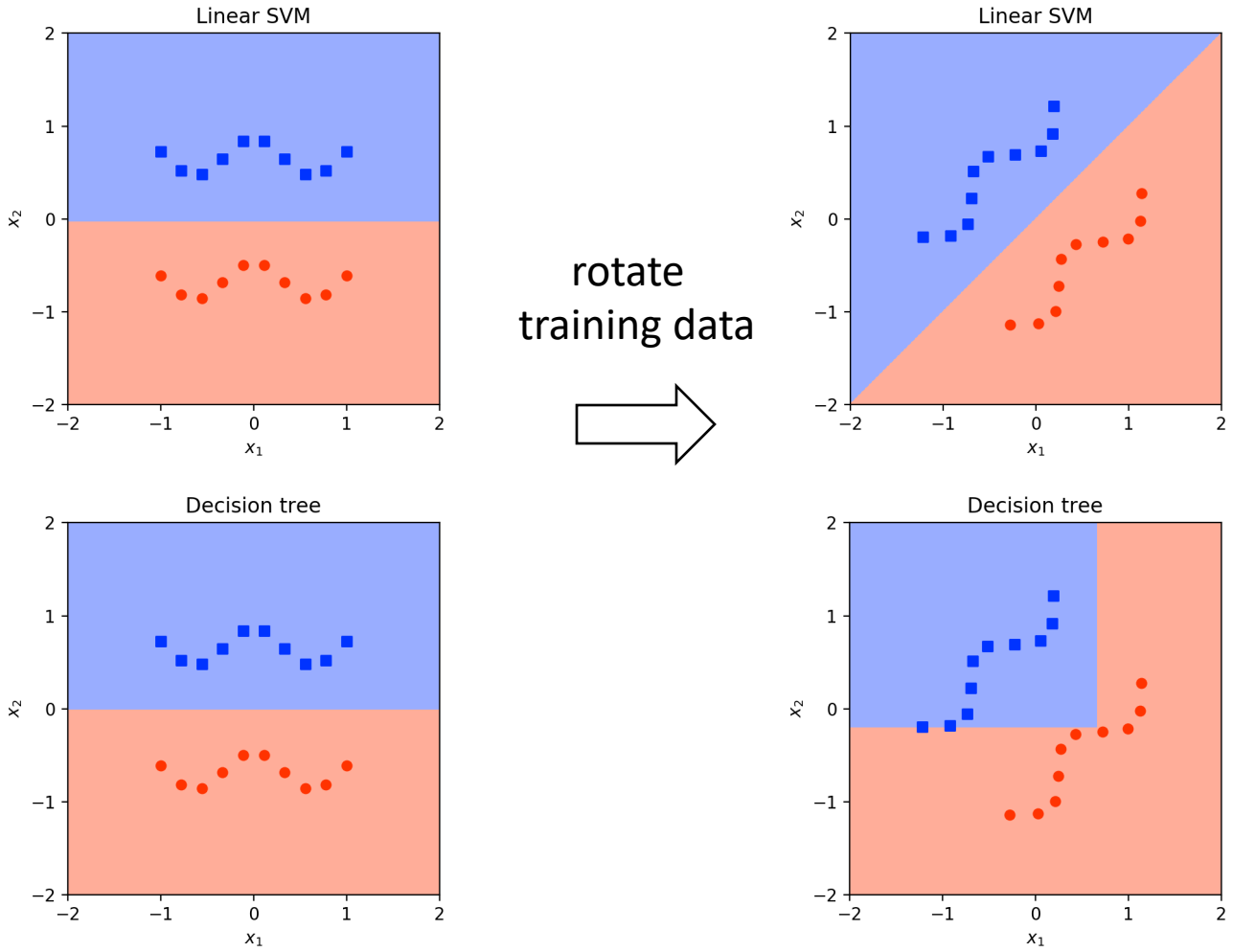


Decision tree classifiers

(There exist extensions to address this)

Why? Because tests one x_j at a time!!

- **Sensitive to feature rotation.**



Classifiers based on decision trees are state of the art for 'classic' ML

An Empirical Comparison of Supervised Learning Algorithms

	MODEL	CAL	ACC	FSC	LFT	ROC	APR	BEP	RMS	MXE	MEAN	OPT-SEL
●	BST-DT	PLT	.843*	.779	**.939**	**.963**	**.938**	.929*	**.880**	**.896**	**.896**	**.917**
●	RF	PLT	.872*	.805	.934*	.957	.931	**.930**	.851	.858	.892	.898
●	BAG-DT	–	.846	.781	.938*	.962*	.937*	.918	.845	.872	.887*	.899
●	BST-DT	ISO	.826*	.860*	.929*	.952	.921	.925*	.854	.815	.885	.917*
●	RF	–	**.872**	.790	.934*	.957	.931	**.930**	.829	.830	.884	.890
●	BAG-DT	PLT	.841	.774	.938*	.962*	.937*	.918	.836	.852	.882	.895
●	RF	ISO	.861*	**.861**	.923	.946	.910	.925	.836	.776	.880	.895
●	BAG-DT	ISO	.826	.843*	.933*	.954	.921	.915	.832	.791	.877	.894
	SVM	PLT	.824	.760	.895	.938	.898	.913	.831	.836	.862	.880
	ANN	–	.803	.762	.910	.936	.892	.899	.811	.821	.854	.885
	SVM	ISO	.813	.836*	.892	.925	.882	.911	.814	.744	.852	.882
	ANN	PLT	.815	.748	.910	.936	.892	.899	.783	.785	.846	.875
	ANN	ISO	.803	.836	.908	.924	.876	.891	.777	.718	.842	.884
●	BST-DT	–	.834*	.816	**.939**	**.963**	**.938**	.929*	.598	.605	.828	.851
	KNN	PLT	.757	.707	.889	.918	.872	.872	.742	.764	.815	.837
	KNN	–	.756	.728	.889	.918	.872	.872	.729	.718	.810	.830
	KNN	ISO	.755	.758	.882	.907	.854	.869	.738	.706	.809	.844
	BST-STMP	PLT	.724	.651	.876	.908	.853	.845	.716	.754	.791	.808
	SVM	–	.817	.804	.895	.938	.899	.913	.514	.467	.781	.810
	BST-STMP	ISO	.709	.744	.873	.899	.835	.840	.695	.646	.780	.810
	BST-STMP	–	.741	.684	.876	.908	.853	.845	.394	.382	.710	.726
●	DT	ISO	.648	.654	.818	.838	.756	.778	.590	.589	.709	.774
●	DT	–	.647	.639	.824	.843	.762	.777	.562	.607	.708	.763
●	DT	PLT	.651	.618	.824	.843	.762	.777	.575	.594	.706	.761
	LR	–	.636	.545	.823	.852	.743	.734	.620	.645	.700	.710
	LR	ISO	.627	.567	.818	.847	.735	.742	.608	.589	.692	.703
	LR	–	.636	.545	.823	.852	.743	.734	.620	.645	.700	.710
				567	.818	.847	.735	.742	.608	.589	.692	.703
				500	.823	.852	.743	.734	.593	.604	.685	.695

Caruana & Niculescu-Mizil, ICML 2006

sklearn.tree.DecisionTreeClassifier

class sklearn.tree.**DecisionTreeClassifier**(*criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False*) [source]

A decision tree classifier.

Read more in the User Guide.

Parameters:

criterion : *string, optional (default="gini")*

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : *string, optional (default="best")*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int or None, optional (default=None)*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

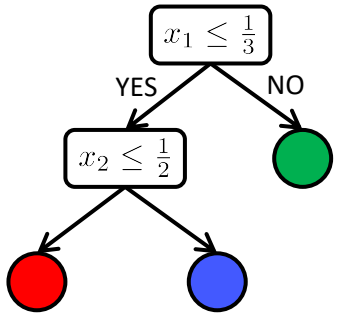
min_samples_split : *int, float, optional (default=2)*

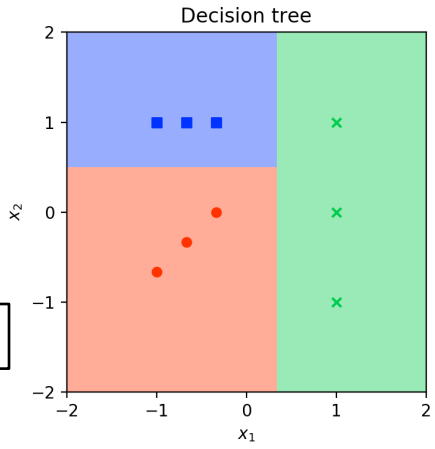
The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Building a decision tree (training)

- Given training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ where $y_i \in \{1, \ldots, K\}$
- **Goal:** build a tree such that each $\mathbf{x}_i$ follows a path to a leaf that assigns correct label y_i.
- Build the tree recursively:
  - Choose a feature x_j to test
  - Choose a value τ upon which to split
  - Built left and right trees with relevant data
- Outputs a "special kind of program" from






```
if (<feature> <op> <value>)
    goto <yes_node>;
else
    goto <no_node>;
```

and

```
return <class_label>;
```

where <op> $\in \{\leq, =, \neq\}$

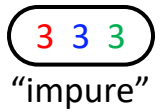
Decision tree training questions:

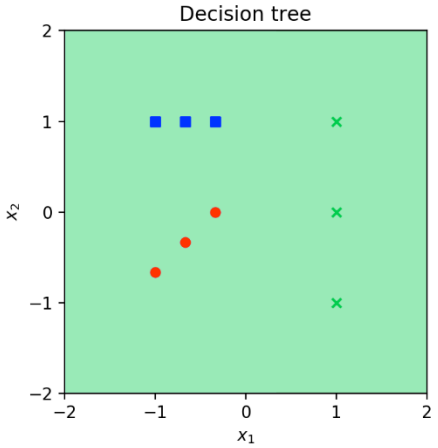
Questions as we consider creating a new node:

1. Should we do a *B*-way split, or 2-way split?
   - We focus on 2-way splits (binary trees), but others exist
2. Which feature x_j should be tested?
3. Which threshold τ should be used?
4. When should we stop splitting and declare a leaf?
5. When training samples from different classes can arrive at the same leaf (an "impure" leaf), what class should we assign the leaf?

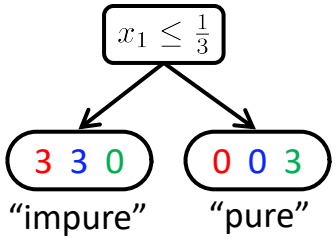
Notice that these are questions about the procedure of building a tree. Decision tree training tends to be described *procedurally*, and only *evaluated* using an explicit "loss function."

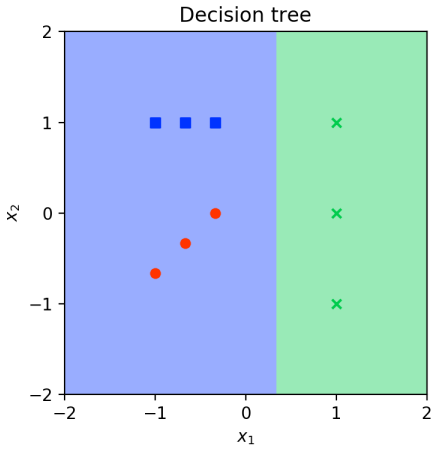
Notion of *Impurity*

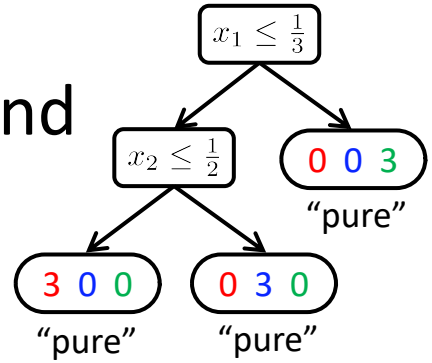


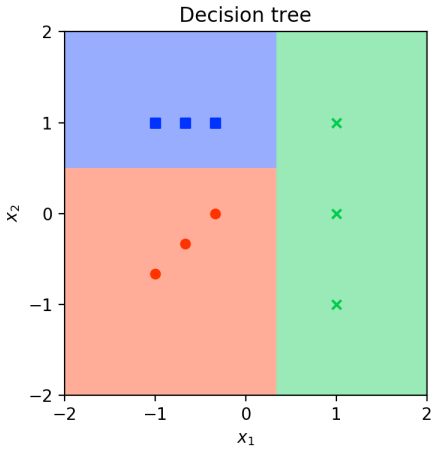


- If all training samples that arrive at a leaf are same class, the leaf is “pure.”
- If a leaf is “impure” then we *may* decide to split.
- If we do split, we guarantee the subtrees have leaves that are strictly “more pure.”
- Training algorithm should find splits that give *lowest* class impurity (uncertainty)!



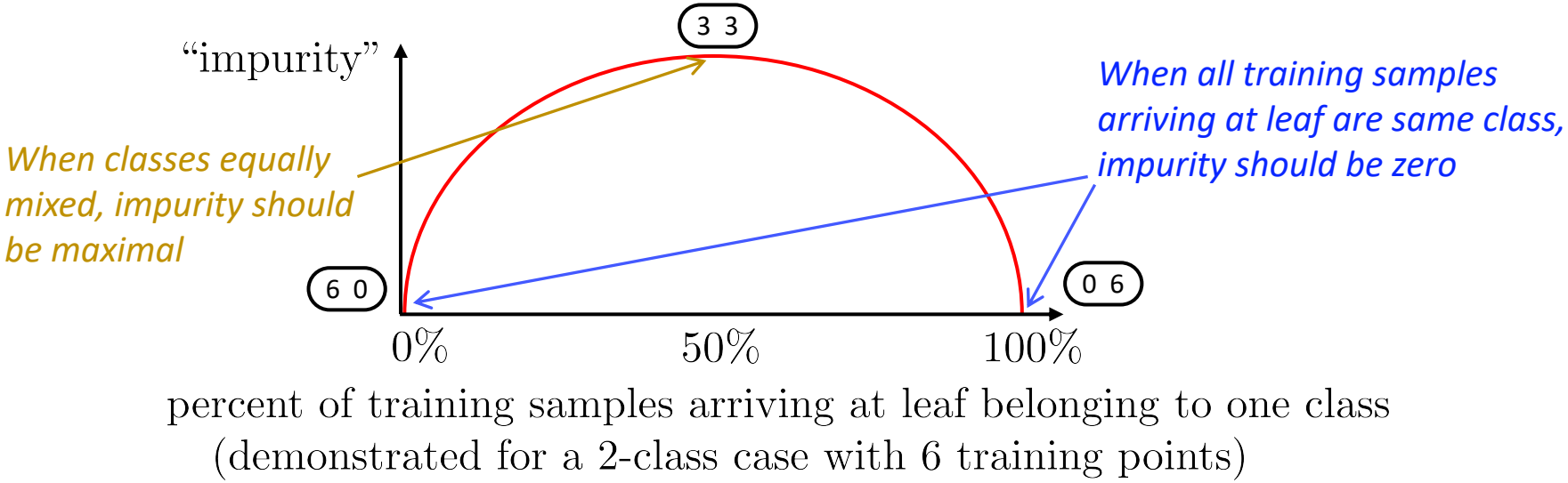






Measuring a leaf's *impurity*

What do we want from an "impurity" measure?



Two most popular measures of impurity:

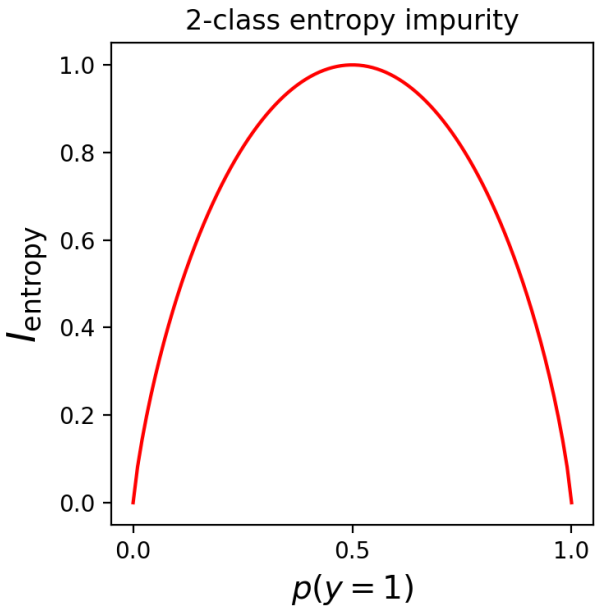
1. Entropy impurity (information impurity)
2. Gini impurity (Gini index)

Entropy impurity (Info impurity)

- Let entropy impurity I_{entropy} at a node be:

$$I_{\text{entropy}} = -\sum_{k=1}^{K} \underbrace{p(y=k)} \log_2 p(y=k)$$

Probability that a training sample $\mathbf{x}_i$ that arrives at this node will have class $y_i = k$, *i.e.* it is the fraction of training samples arriving at this node with $y_i = k$.



- Zero entropy means we have *complete* certainty about the training arriving at this node
  - i.e. node is pure, only one class
- Entropy is maximal when all classes equally likely

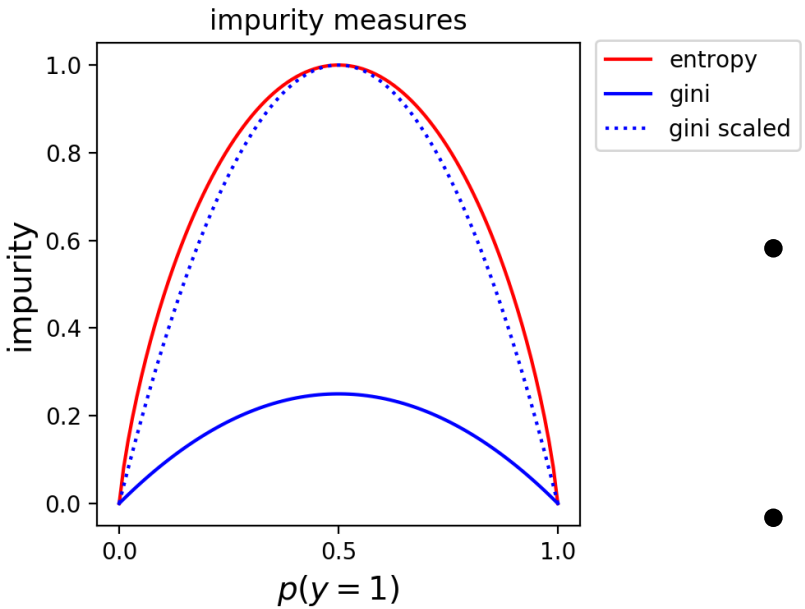
Gini impurity (Gini index)

- Let Gini impurity I_{gini} at a node be:

$$I_{\text{gini}} = \sum_{k \neq k'} p(y=k)p(y=k')$$

$$= \frac{1}{2}\left(1 - \sum_{k=1}^{K} \underbrace{p(y=k)}^{2}\right)$$

Same as before



- Similar to entropy, but slightly more ‘peaked’ around its maximal value.
- Doesn’t usually matter.

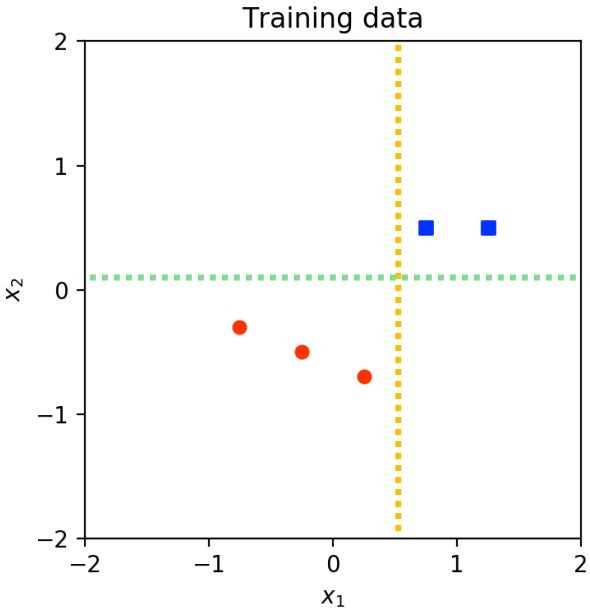
(We only focus on tests of the form $x_j \leq \tau$, see Duda, Hart, Stork for others)

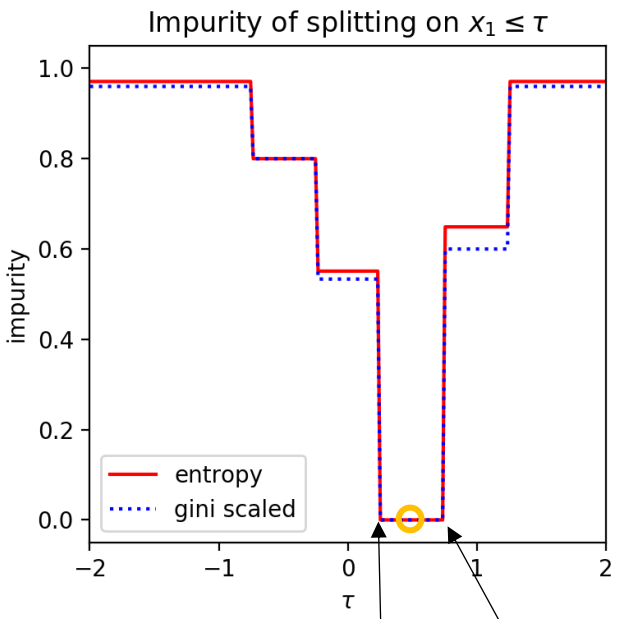
Choosing a split threshold τ

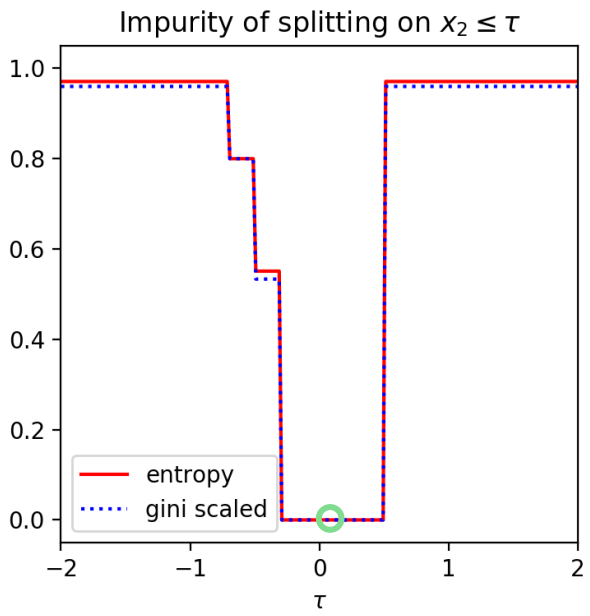
- Given choice of feature j and impurity measure I, the new impurity $I(\tau)$ after splitting node at τ is:

$$I(\tau) = \underbrace{P(\tau)}_{\text{fraction of } i \text{ for which } x_{ij} \leq \tau \text{ (fraction going left)}} \underbrace{I_L(\tau)}_{\text{impurity over } y_i \text{ for which } x_{ij} \leq \tau \text{ } (\mathbf{x}_i \text{ goes left})} + (1 - P(\tau)) \underbrace{I_R(\tau)}_{\text{impurity over } y_i \text{ for which } x_{ij} > \tau \text{ } (\mathbf{x}_i \text{ goes right})}$$

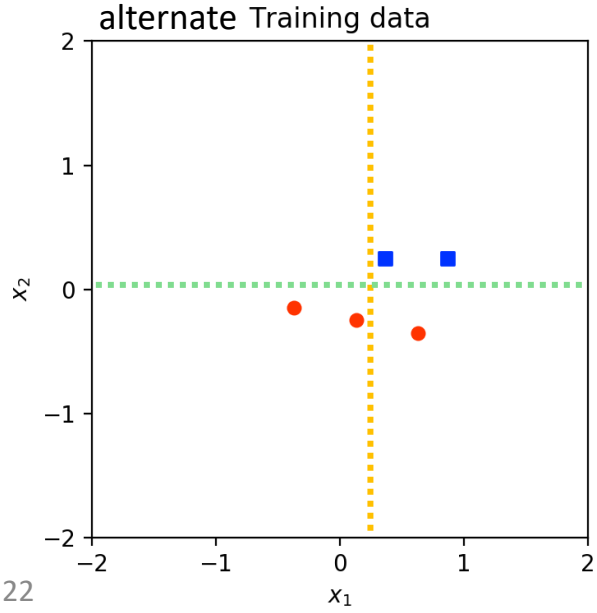
- Find $\tau^* \in \arg\min_\tau I(\tau)$, a 1D optimization problem!
  - Sort the x_{ij} values across i for all $\mathbf{x}_i$ arriving at this node
  - Find each i for which $y_i \neq y_{i+1}$ in sorted order, and compute $I(x_{ij})$, keeping track of i^* with lowest impurity
  - Split at midpoint $\tau^* = \frac{1}{2}(x_{i^*,j} + x_{i^*+1,j})$

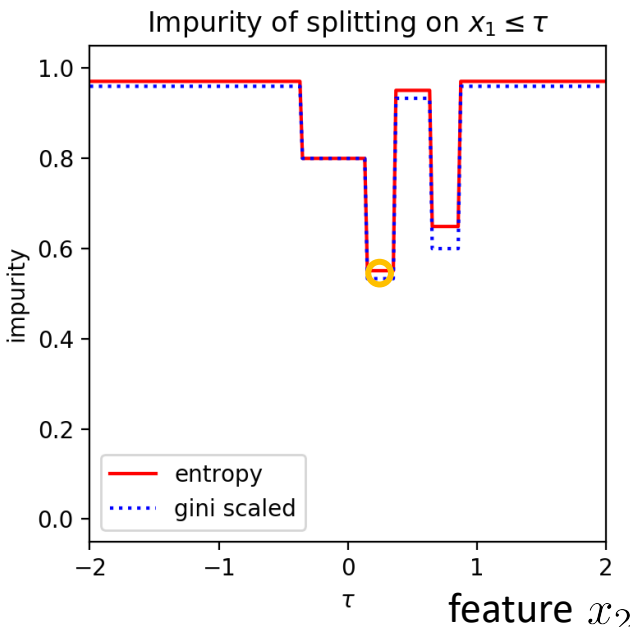


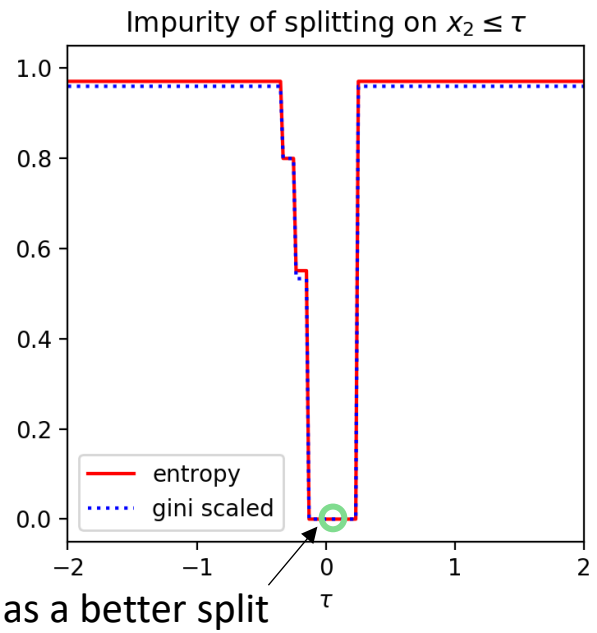




typically choose midpoint between $x_{i,j}$ and $x_{i+1,j}$







feature x_2 has a better split

Scikit-learn implementation

```
275  cdef class BestSplitter(BaseDenseSplitter):
276      """Splitter for finding the best split."""
277
278      cdef int node_split(self, double impurity, SplitRecord* split,
279                          SIZE_t* n_constant_features) nogil except -1:
280          """Find the best split on node samples[start:end]
281
282          Returns -1 in case of failure to allocate memory (and raise MemoryError)
283          or 0 otherwise.
284          """
```

```
378                  # Sort samples along that feature
379                  sort(Xf + start, samples + start, end - start)
```

```
420                          self.criterion.update(current.pos)
421                          current_proxy_improvement = self.criterion.proxy_impurity_improvement()
422                          if current_proxy_improvement > best_proxy_improvement:
423                              best_proxy_improvement = current_proxy_improvement
424                              # sum of halves is used to avoid infinite value
425                              current.threshold = Xf[p - 1] / 2.0 + Xf[p] / 2.0
426                              best = current  # copy
```

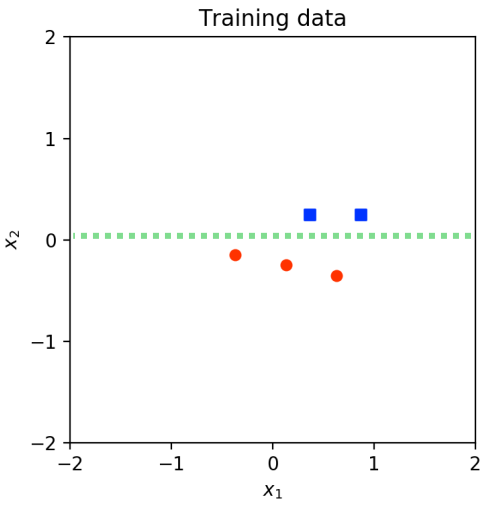
could be entropy or Gini

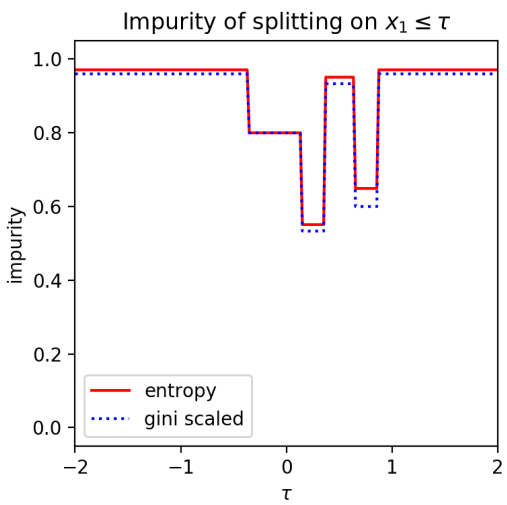
https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/tree/_splitter.pyx

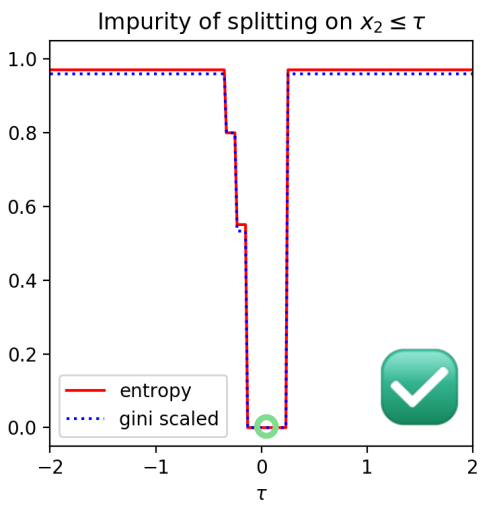
Choosing a feature x_j

How should we choose the feature x_j for splitting?

1. **Random.** Choose a $j \in \{1, \ldots, D\}$ at random
2. **Best.** Choose the j that allows for the smallest impurity $I(\tau)$, *i.e.* the biggest drop in impurity.
3. **Best from random subset.** A compromise.







When to stop splitting

- **Case 1:** When the leaf is pure, stop splitting.
- **Case 2:** When a leaf is impure but $I(\tau)$ cannot be decreased for any choice of τ, stop splitting.

However, using only the above criteria will lead to complex decision trees that likely over-fit the data. So, we may also want to 'regularize' by imposing:

- **Max depth:** If the height of the leaf in the tree is already at some maximum depth, stop splitting.
- **Min samples:** If fewer than some number of training samples can arrive at the leaf, stop splitting.
- ...

The training algorithm is *greedy*, local, no global objective. Can we do better?

Volume 5, number 1 INFORMATION PROCESSING LETTERS May 1976

CONSTRUCTING OPTIMAL BINARY DECISION TREES IS NP-COMPLETE*

Laurent HYAFIL
IRIA – Laboria, 78150 Rocquencourt, France

Ronald L. RIVEST
Dept. of Electrical Engineering and Computer Science, M.I.T.

1. Introduction

We demonstrate that constructing optimal binary decision trees is an NP-complete problem, where an optimal tree is one which minimizes the expected number of tests required to identify the unknown object. Precise definitions of NP-complete problems are given in refs. [1,2,4].

While the proof to be given is relatively simple, the importance of this result can be measured in terms of the large amount of effort that has been put into finding efficient algorithms for constructing optimal binary decision trees (see [3,5,6] and their references). Thus at present we may conjecture that no such efficient algorithm exists (on the supposition that P ≠ NP), thereby supplying motivation for finding efficient heuristics for constructing near-optimal decision trees.

- Building the simplest possible decision tree is an NP-complete problem
- However, NP-complete doesn't mean can't attack it with stronger algorithms!

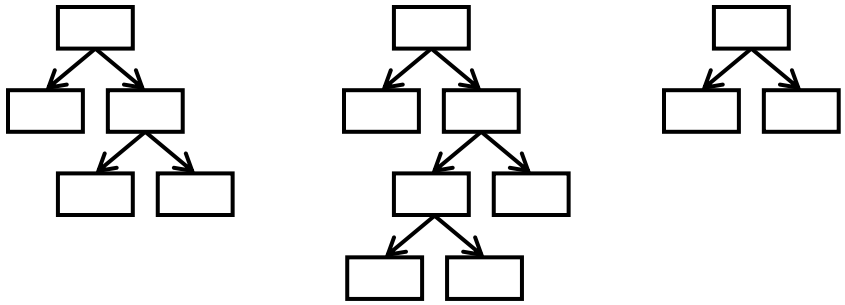
Optimal classification trees "OCT"

Dimitris Bertsimas[1] · **Jack Dunn**[2]

Received: 17 September 2015 / Accepted: 3 March 2017

Abstract State-of-the-art decision tree methods apply heuristics recursively to create each split in isolation, which may not capture well the underlying characteristics of the dataset. The optimal decision tree problem attempts to resolve this by creating the entire decision tree at once to achieve global optimality.

Random Forests



Random Forests

- **Observation:** There is a lot of "arbitrariness" in the way decision trees are built.
  - Tree is highly sensitive to individual training points
  - Different splits early on can lead to completely different trees and completely different decision regions.
  - Data dramatically under-determines the possible trees.
- **Idea:** What if we could "integrate out" the arbitrariness, to get some kind of kind of expectation over all the decision trees that we could have chosen to fit our data.
  - Build lots of different decision trees (a 'forest') that all fit the data, and then average over their predictions.
  - A different form of 'regularization' than we've seen!

To do this we must introduce 'variation' into the decision trees. But how? This paper by Breiman established a good technique.

Bagging Predictors

Leo Breiman[1]
Department of Statistics
University of California at Berkeley

Abstract

Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy. The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Bagging predictors

L Breiman - Machine learning, 1996 - Springer

Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The …

☆ 99 Cited by 21633 Related articles All 65 versions »

Introducing variation into DTs

- Remember, decision trees (DTs) are sensitive to specific training set; adding 1 point can affect a split!
- Rather than randomizing the DT training algorithm, introduce variation by varying the *training set itself*.
- In Random Forests (RFs), each DT is trained on a different *sampling* of the full size-N training set.
  - Should we *perturb* the data? Possible, but hard to know what perterbations help or 'corrupt' in harmful ways.
  - Should we *subsample* the data? Reasonable! But each new training set is smaller than N, and by how much?
  - Should we *re-sample* the data *with replacement*? Good! Some $(\mathbf{x}_i, y_i)$ get duplicated, but no parameters to choose.

Random Forests do *this*

```
print(X)
print(y)
```

```
[[0. 0.]
 [1. 1.]
 [2. 2.]
 [3. 3.]
 [4. 4.]]
[0 1 2 3 4]
```

Original training set $(N = 5, D = 2)$

How do we **resample** this data?

```
sample_indices = np.random.randint(0, N, N)
print(sample_indices)
```

```
[2 4 2 1 3]
```

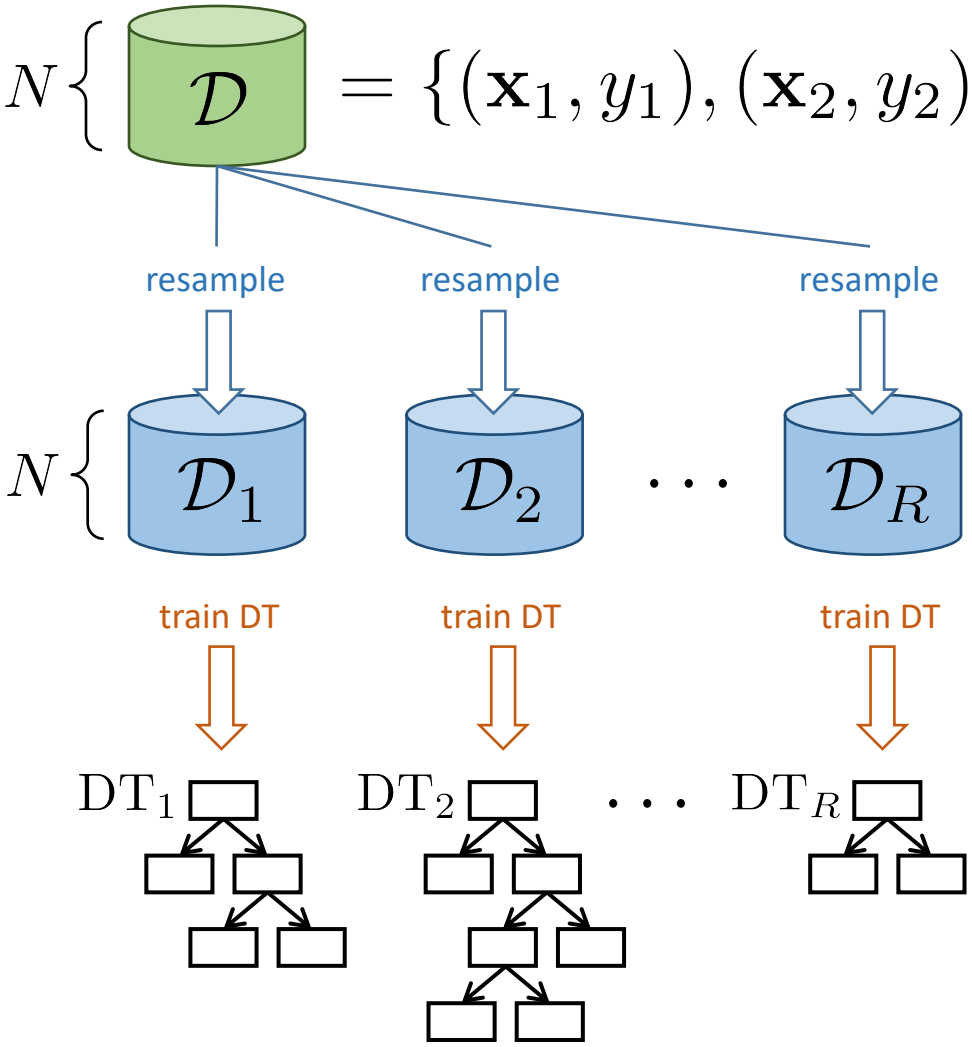
A random sample of N indices from $\{0, \ldots, N-1\}$, with replacement

```
X_resampled = X[sample_indices]
y_resampled = y[sample_indices]
print(X_resampled)
print(y_resampled)
```

```
[[2. 2.]
 [4. 4.]
 [2. 2.]
 [1. 1.]
 [3. 3.]]
[2 4 2 1 3]
```

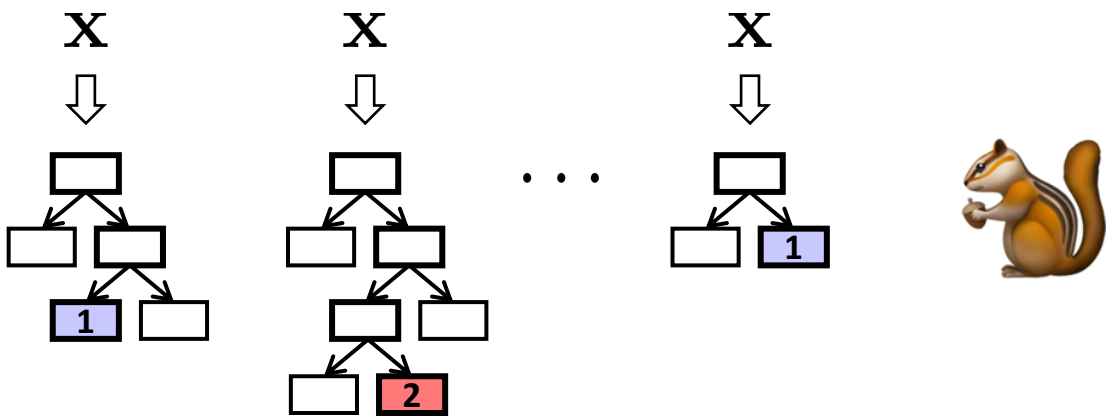
A resampled training set of size N

Random Forest Training



Random Forest Prediction

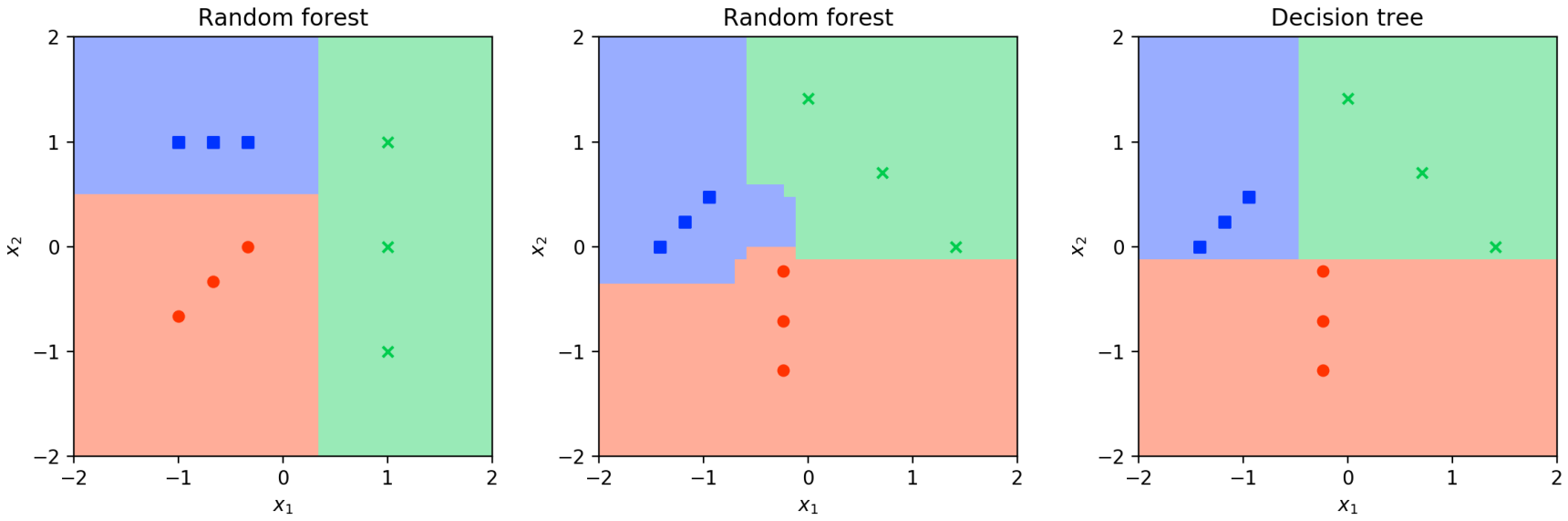
- Given query point $\mathbf{x}$, run it through each tree



- Predict the class with largest share of *R* 'votes'
- Guarantees we don't make bad prediction because of an "unlucky split"
- Fast, can be parallelized on CPU or GPU

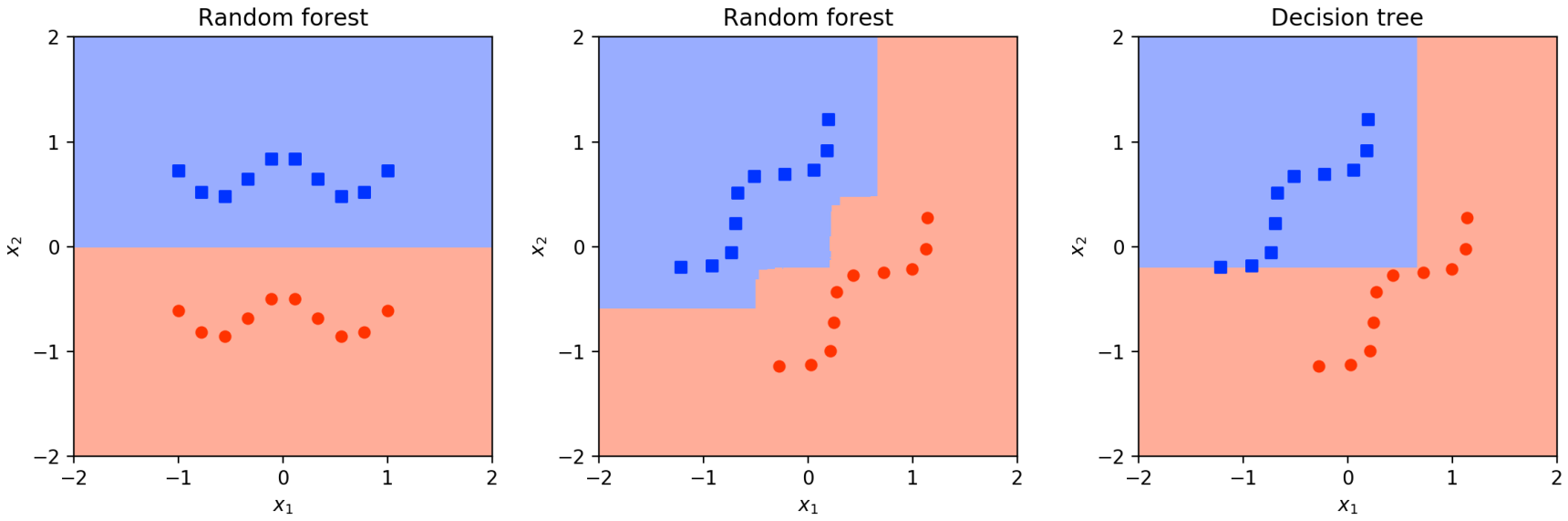
Random Forests

- Hugely successful. Harder to interpret than decision trees, but almost always better predictions.
- Inherits the pros/cons of decision trees, *e.g.* sensitive to feature rotation, not scaling/shifts



Random Forests

- Hugely successful. Harder to interpret than decision trees, but almost always better predictions.
- Inherits the pros/cons of decision trees, *e.g.* sensitive to feature rotation, not scaling/shifts



The team that developed the "pose-recognition" for the original Xbox Kinect

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. *Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.*

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

Random Forest

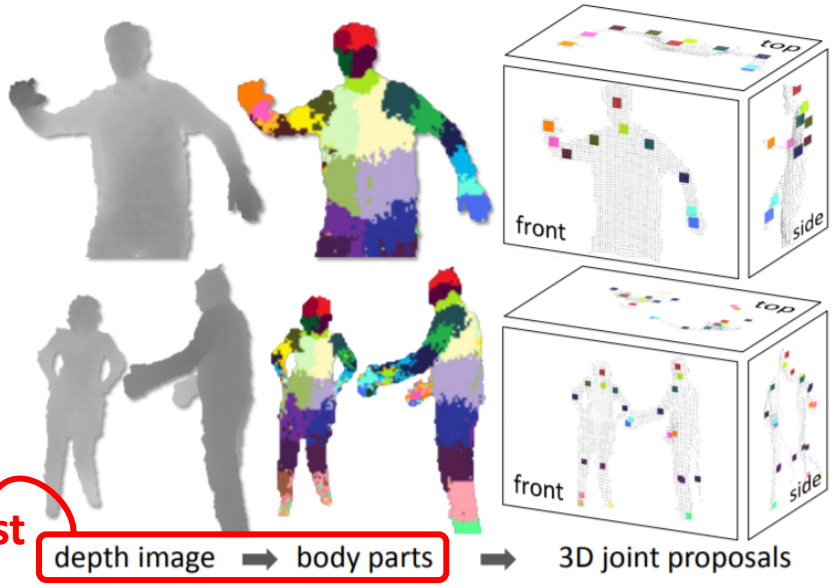


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/BodyPartRecognition.pdf

⬆ **Train on Kinect data from real humans and from synthetic 3D models**

⬇ **Test on Kinect data from real humans**

Figure 2. **Synthetic and real data**. Pairs of depth image and ground truth body parts. Note wide variety in pose, shape, clothing, and crop.

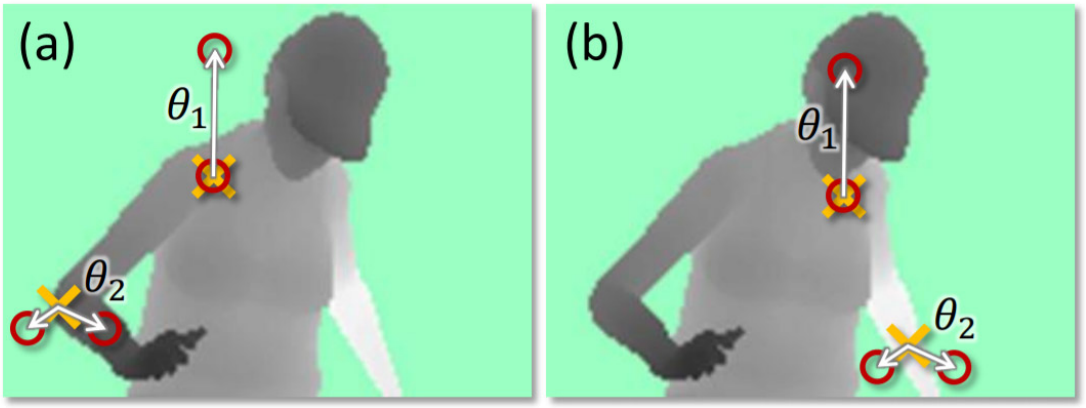


Figure 3. **Depth image features.** The yellow crosses indicates the pixel $\mathbf{x}$ being classified. The red circles indicate the offset pixels as defined in Eq. 1. In (a), the two example features give a large depth difference response. In (b), the same two features at new image locations give a much smaller response.

(classifying which body part the pixel likely belongs to)

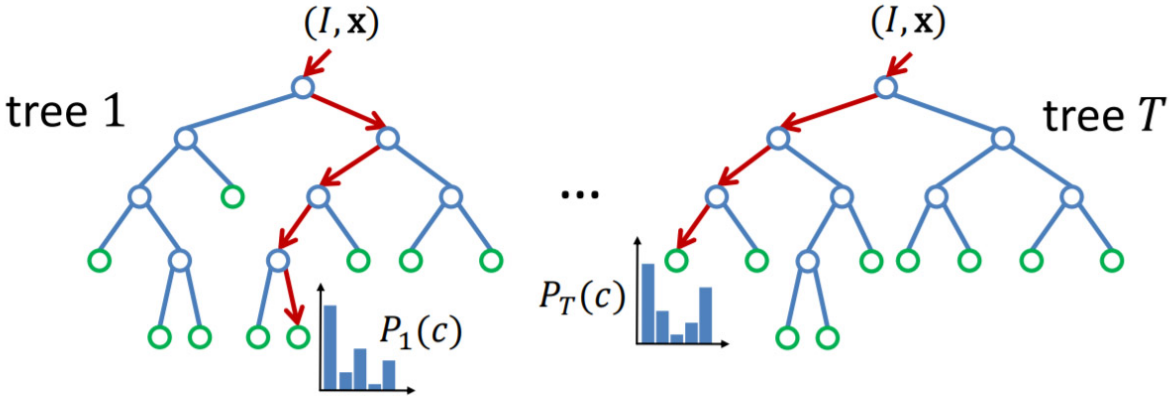


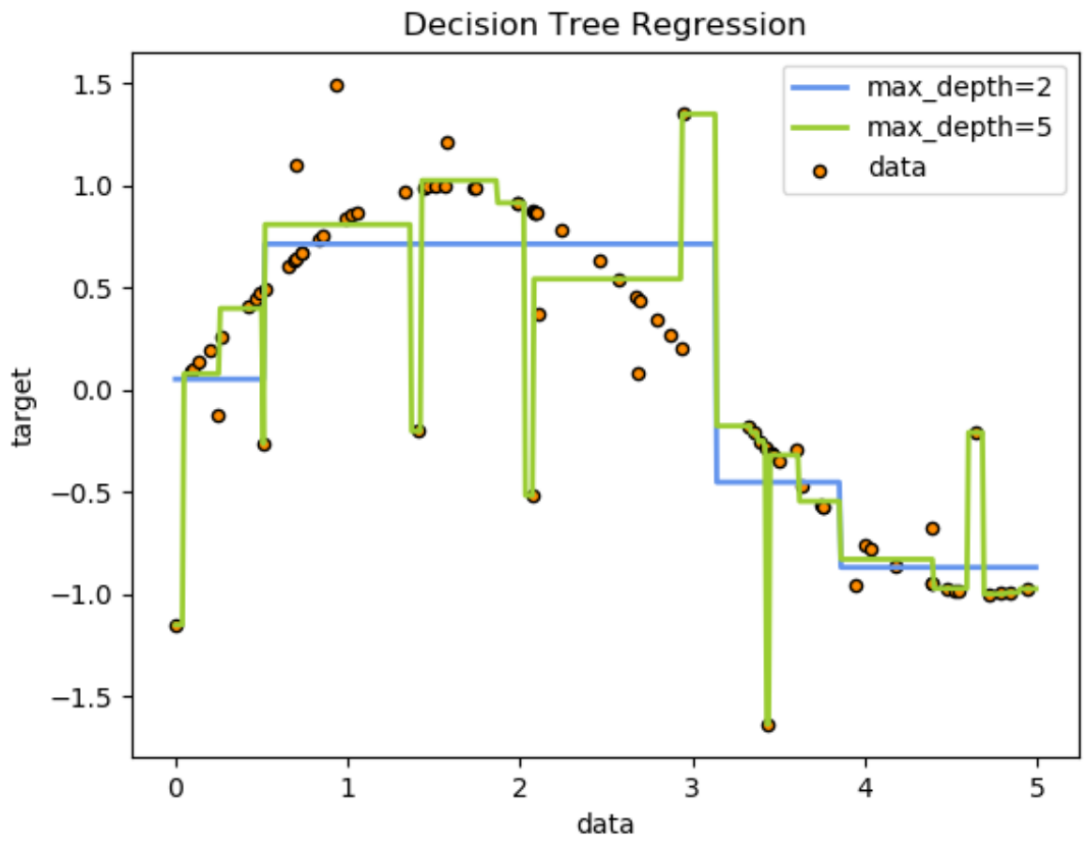
Figure 4. **Randomized Decision Forests.** A forest is an ensemble of trees. Each tree consists of split nodes (blue) and leaf nodes (green). The red arrows indicate the different paths that might be taken by different trees for a particular input.

Decision Tree Regression

A 1D regression with decision tree.

The decision trees is used to fit a sine curve with addition noisy observation. As a result, it learns local linear regressions approximating the sine curve.

We can see that if the maximum depth of the tree (controlled by the `max_depth` parameter) is set too high, the decision trees learn too fine details of the training data and learn from the noise, i.e. they overfit.



Decision trees and random forests can both be extended to handle regression

You may see "Classification and Regression Trees" (CART) mentioned. CART refers to the general framework for growing decision trees for either classification or regression. It's a family of related tree-based learning algorithms.

Journal of Machine Learning Research 17 (2016) 1-5 Submitted 7/15; Revised 3/16; Published 7/16

Are Random Forests Truly the Best Classifiers?

Michael Wainberg
Babak Alipanahi
Brendan J. Frey

Abstract

The JMLR study *Do we need hundreds of classifiers to solve real world classification problems?* benchmarks 179 classifiers in 17 families on 121 data sets from the UCI repository and claims that "the random forest is clearly the best family of classifier". In this response, we show that the study's results are biased by the lack of a held-out test set and the exclusion of trials with errors. Further, the study's own statistical tests indicate that random forests do not have significantly higher percent accuracy than support vector machines and neural networks, calling into question the conclusion that random forests are the best classifiers.

Keywords: classification, benchmarking, random forests, support vector machines, neural networks

Still, even if Random Forests don't always win, or don't always win by much, they're an excellent model and should be part of your baselines whenever possible!!

http://www.jmlr.org/papers/volume17/15-374/15-374.pdf

Readings

PRML book unfortunately does not explain decision trees or random forests ☹. So, you are not expected to know more than is in these slides.

However, the “Pattern Classification” book (“Duda, Hart, Stork book”) contains good readings:

- §8.1 Non-Metric Methods
- §8.2 Decision Trees
- §8.3 Classification and Regression Trees (CART)