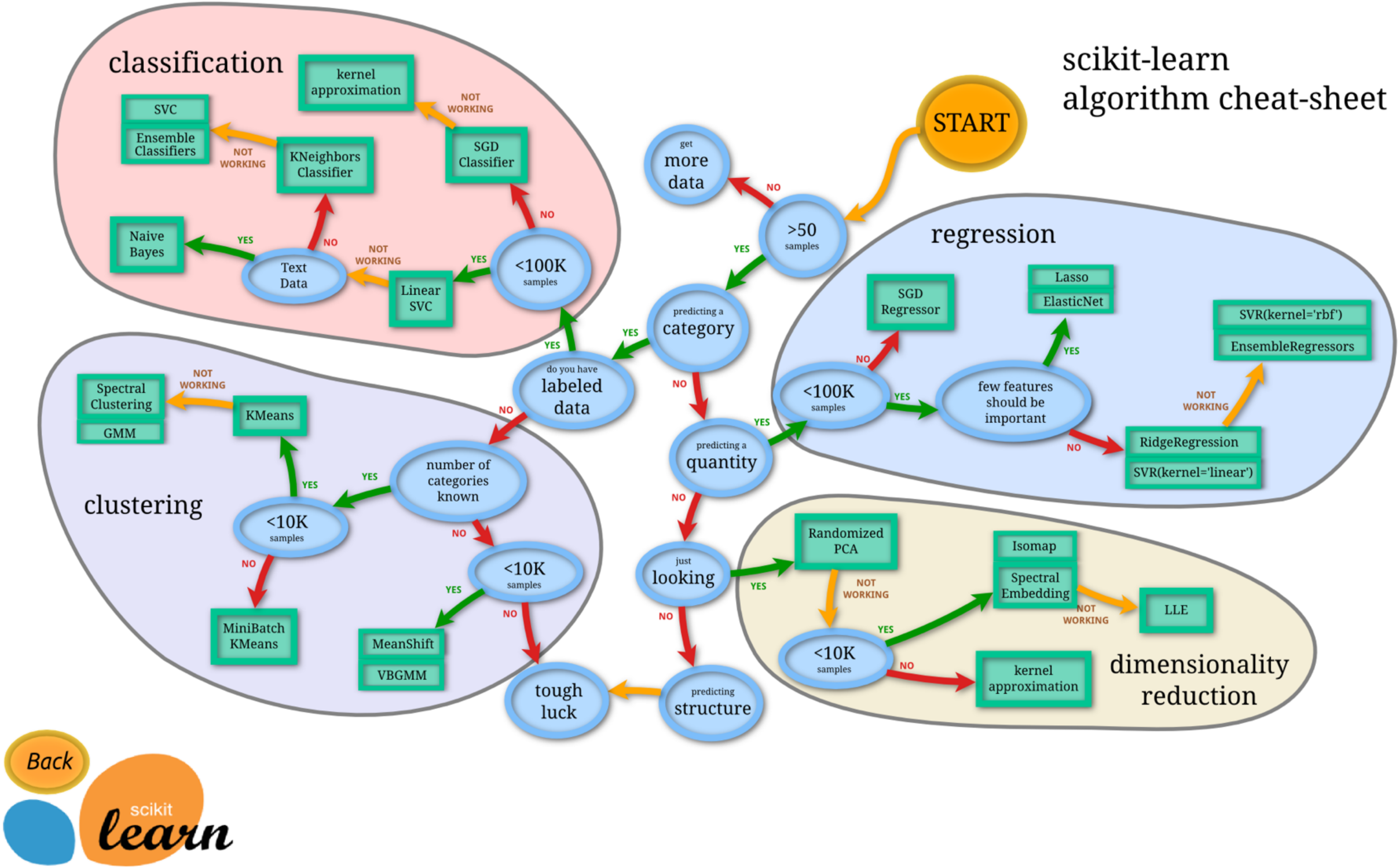
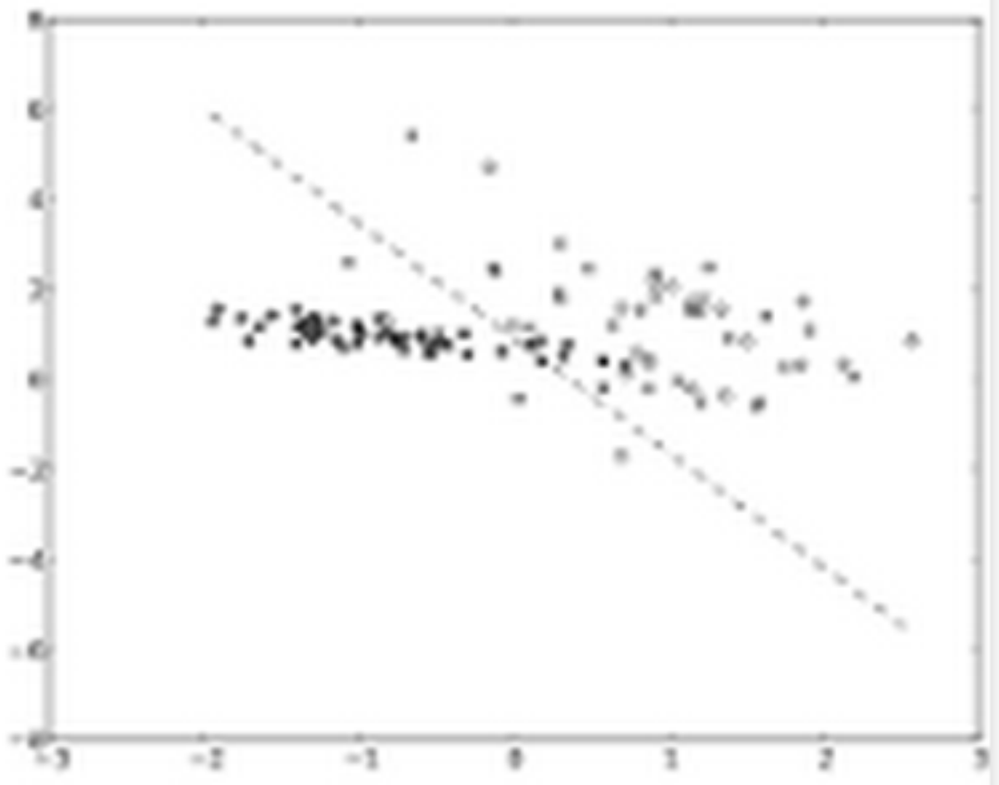
Machine Learning in Go

Decision Tree and Random Forest


scikit-learn
algorithm cheat-sheet
START
>50 samples
get more data
predicting a category
do you have labeled data
predicting a quantity
just looking
predicting structure
tough luck
classification
<100K samples
Linear SVC
Text Data
Naive Bayes
KNeighbors Classifier
SVC
Ensemble Classifiers
SGD Classifier
kernel approximation
clustering
number of categories known
<10K samples
KMeans
Spectral Clustering
GMM
MiniBatch KMeans
MeanShift
VBGMM
regression
SGD Regressor
few features should be important
Lasso
ElasticNet
RidgeRegression
SVR(kernel='linear')
SVR(kernel='rbf')
EnsembleRegressors
dimensionality reduction
Randomized PCA
Isomap
Spectral Embedding
LLE
kernel approximation
YES
NO
NOT WORKING
Back
scikit learn

we’ll start simple

In **machine learning** and statistics, **classification** is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.



Statistical classification - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Statistical_**classification** Wikipedia

classification examples

- spam/not-spam
- fraud/not-fraud
- OCR
- iris flower species (setosa, versicolor, virginica)

workflow

Training Data

Species	Sepal Length	Sepal Width	…
setosa	5.4	3.9	…
versicolor	5.5	2.6	…
virginica	6.3	2.5	…
…	…	…	…

machine learning algo

classification rule(s)

New Data

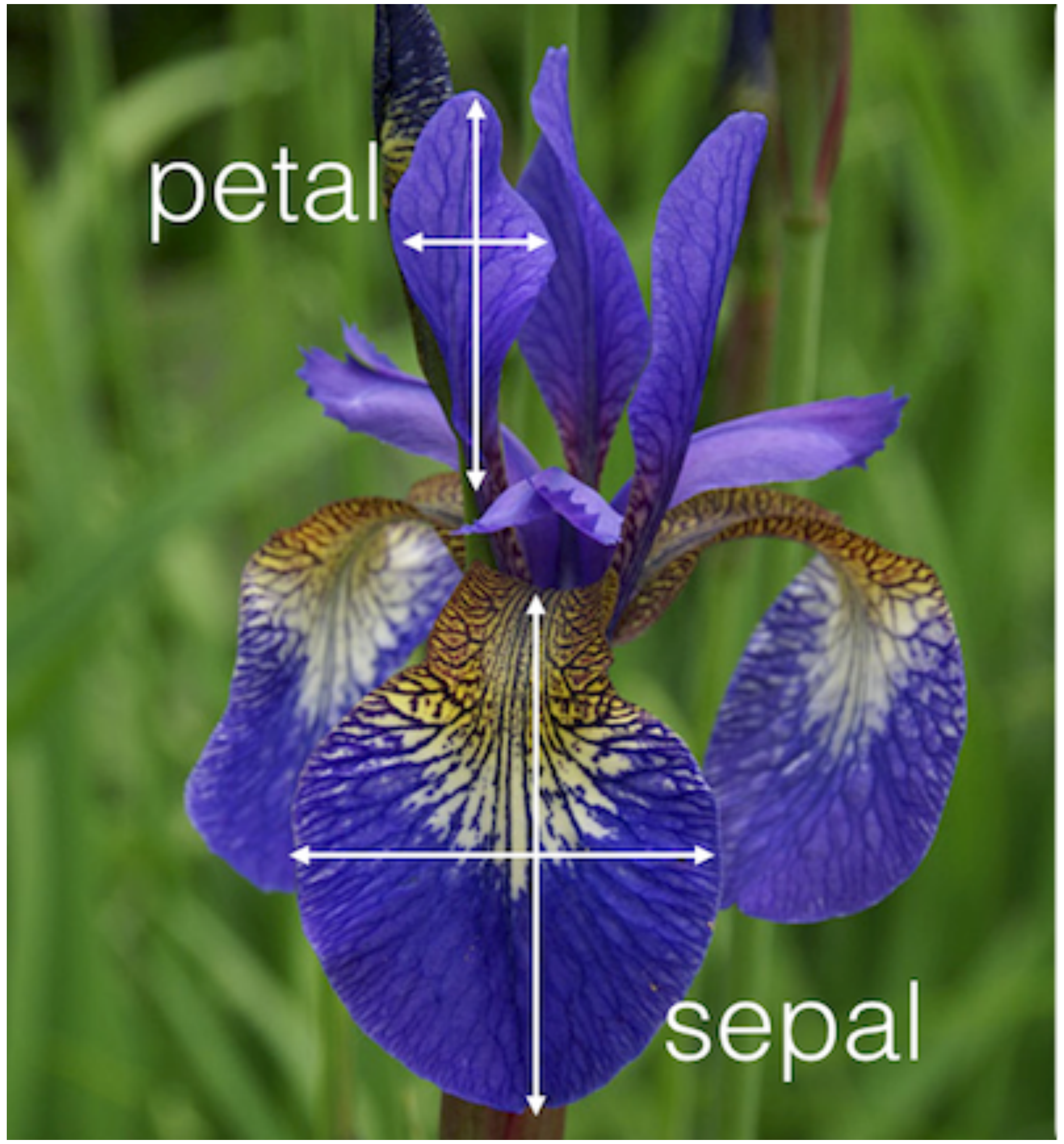
Species	Sepal Length	Sepal Width	…
?	5.4	3.9	…
?	5.5	2.6	…

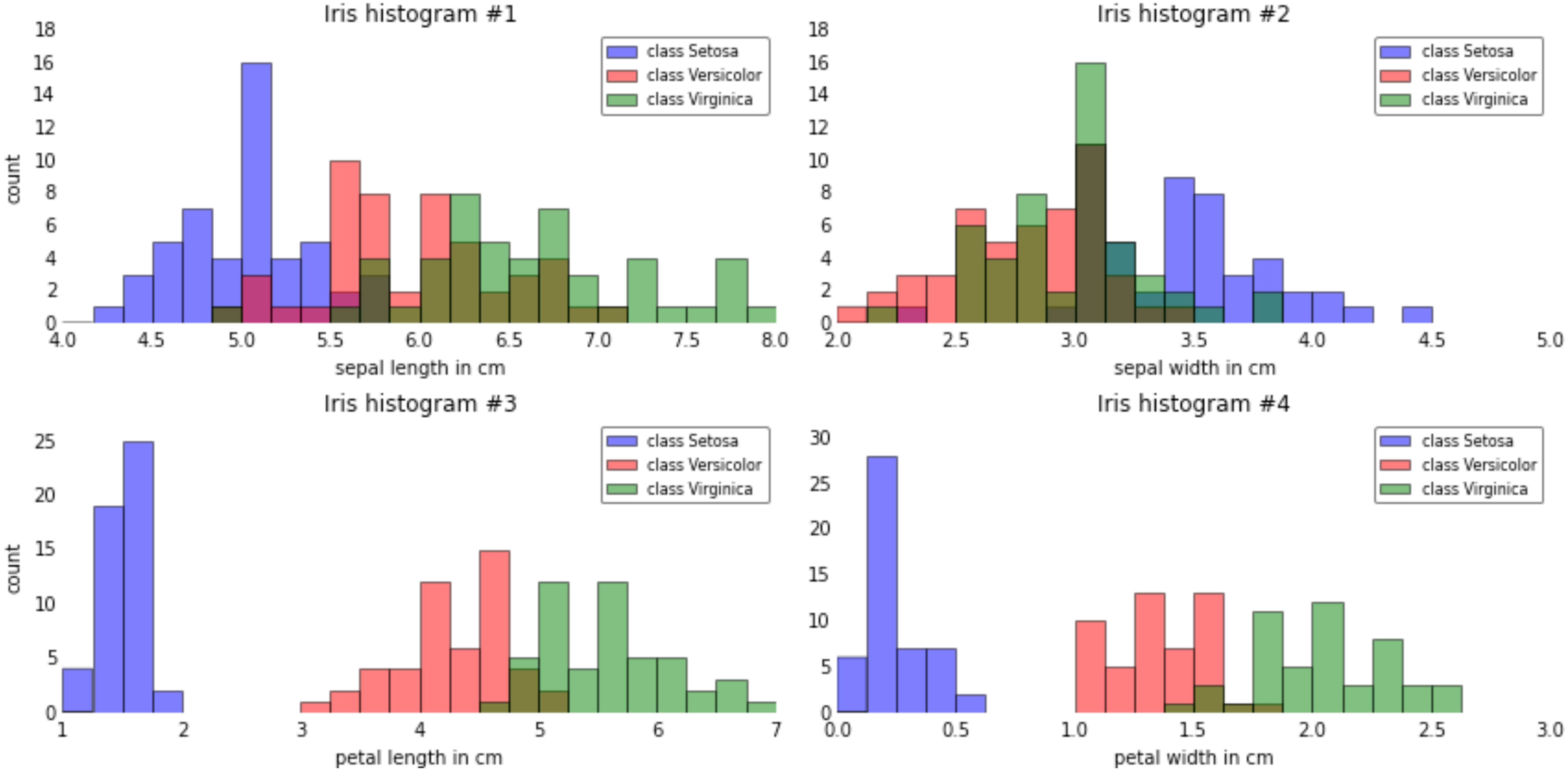
Predicted Labels

Species	Sepal Length	Sepal Width	…
versicolor	5.4	3.9	…
setosa	5.5	2.6	…

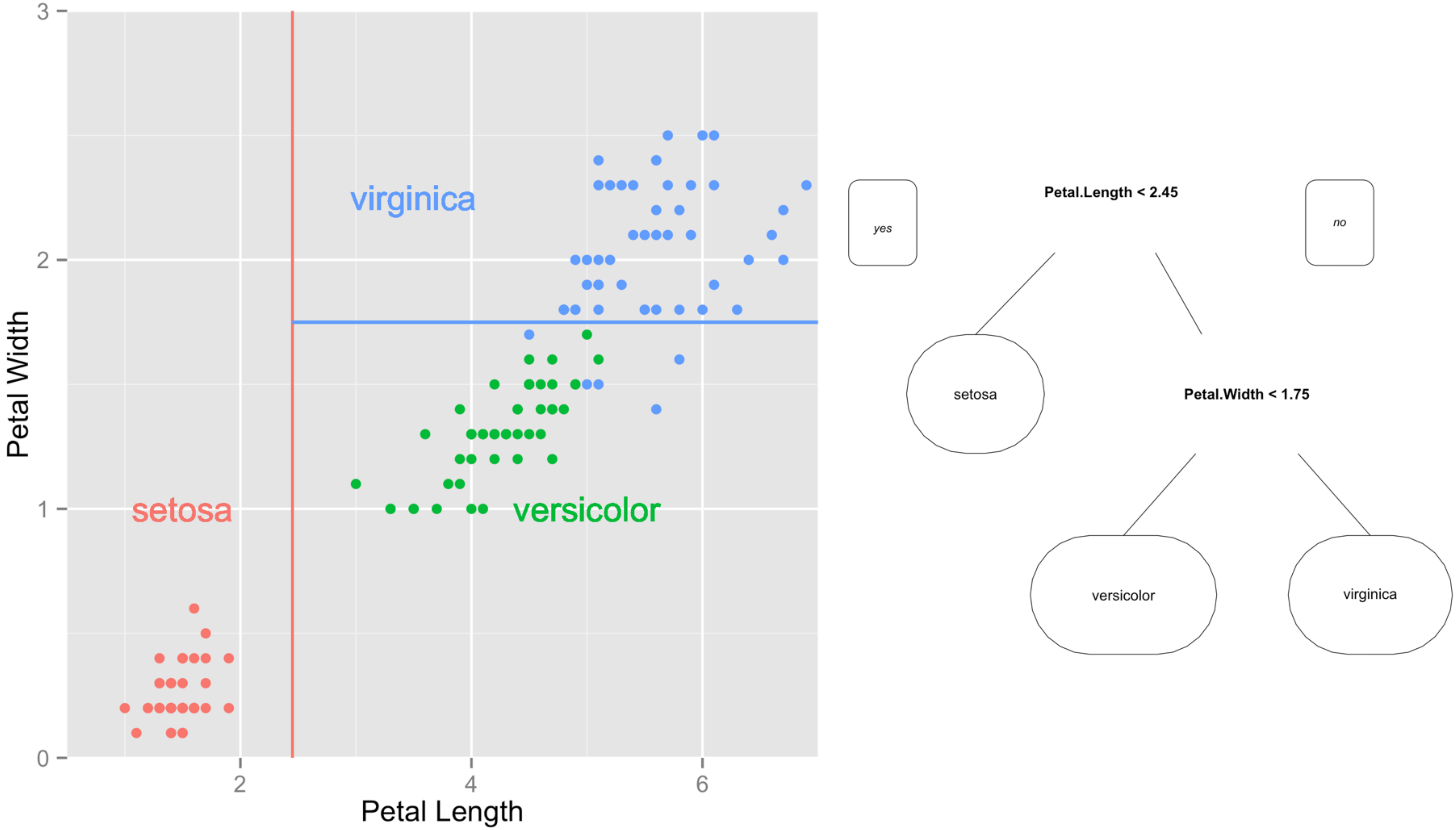
popular classification algos

- Naive Bayes (the “hello world!” of machine learning)
- Logistic Regression
- Decision Tree
- Support Vector Machine (SVM)
- Random Forest
- Gradient Boosted Tree (GBT/GBM)
- Neural Networks/Deep Learning


petal
sepal


Iris histogram #1
Iris histogram #2
Iris histogram #3
Iris histogram #4
class Setosa
class Versicolor
class Virginica
count
sepal length in cm
sepal width in cm
petal length in cm
petal width in cm

Decision Tree



virginica
setosa
versicolor
Petal Width
Petal Length
0
1
2
3
2
4
6
Petal.Length < 2.45
yes
no
setosa
Petal.Width < 1.75
versicolor
virginica

```
function BuildDecisionTree(L)
    Create node t from the learning sample L_t = L
    if the stopping criterion is met for t then
        ŷ_t = some constant value
    else
        Find the split on L_t that maximizes impurity decrease

            s* = arg max_{s∈Ω} Δi(s, t)

        Partition L_t into L_{t_L} ∪ L_{t_R} according to s*
        t_L = BuildDecisionTree(L_L)
        t_R = BuildDecisionTree(L_R)
    end if
    return t
end function
```

stopping rules

- all output values are equal
- all input variables are constant
- node size < **minSamplesSplit**
- depth > **maxDepth**
- split value < **minImpurityDecrease**

impurity metric

- **Entropy**, or **deviance**:

$$\mathbb{H}(\hat{\boldsymbol{\pi}}) = -\sum_{c=1}^{C} \hat{\pi}_c \log \hat{\pi}_c \tag{16.10}$$

- **Gini index**

$$\sum_{c=1}^{C} \hat{\pi}_c(1 - \hat{\pi}_c) = \sum_c \hat{\pi}_c - \sum_c \hat{\pi}_c^2 = 1 - \sum_c \hat{\pi}_c^2 \tag{16.14}$$

This is the expected error rate. To see this, note that $\hat{\pi}_c$ is the probability a random entry in the leaf belongs to class c, and $(1 - \hat{\pi}_c$ is the probability it would be misclassified.

impurity metric (in English)

How mixed up are the labels in the node?

code

```go
func (t *Tree) buildTree(X [][]float64, Y []int, depth int) *Node {
  n := t.makeNode(Y)
  if t.shouldStop(Y, n, depth) {
    return makeLeaf(n)
  }

  gain, splitVar, splitVal := t.findBestSplit(X, Y)
  if gain < 1e-7 {
    return makeLeaf(n)
  }
  n.SplitVar = splitVar
  n.SplitVal = splitVal

  XLeft, XRight, YLeft, YRight := partitionOnFeatureVal(X, Y, splitVar, splitVal)
  n.Left = t.buildTree(XLeft, YLeft, depth+1)
  n.Right = t.buildTree(XRight, YRight, depth+1)
  return n
}
```

```go
func (t *Tree) findBestSplit(X [][]float64, Y []int) (float64, int, float64) {
  var (
    bestFeature int
    bestVal     float64
    bestGain    float64
  )

  initialImpurity := giniImpurity(Y, len(t.ClassNames))

  for feature := range X[0] {
    gain, val, nLeft := findSplitOnFeature(X, Y, feature, len(t.ClassNames), initialImpurity)
    if nLeft < t.MinSamplesLeaf || len(X)-nLeft < t.MinSamplesLeaf {
      continue
    }

    if gain > bestGain {
      bestGain = gain
      bestFeature = feature
      bestVal = val
    }
  }

  return bestGain, bestFeature, bestVal
}
```

```go
func findSplitOnFeature(X [][]float64, Y []int, feature int, nClasses int, initialImpurity float64)
(float64, float64, int) {
  sortByFeatureValue(X, Y, feature)

  var (
    bestGain, bestVal float64
    nLeft             int
  )
  for i := 1; i < len(X); i++ {
    if X[i][feature] <= X[i-1][feature]+1e-7 { // can't split on locally constant val
      continue
    }

    gain := impurityGain(Y, i, nClasses, initialImpurity)
    if gain > bestGain {
      bestGain = gain
      bestVal = (X[i][feature] + X[i-1][feature]) / 2.0
      nLeft = i
    }
  }
  return bestGain, bestVal, nLeft
}
```

```go
func impurityGain(Y []int, i int, nClasses int, initImpurity float64) float64 {
  // initImpurity := giniImpurity(Y, nClasses)
  impurityLeft := giniImpurity(Y[:i], nClasses)
  impurityRight := giniImpurity(Y[i:], nClasses)

  fracLeft := float64(i) / float64(len(Y))
  fracRight := 1.0 - fracLeft

  return initImpurity - fracLeft*impurityLeft - fracRight*impurityRight
}


func giniImpurity(Y []int, nClasses int) float64 {
  classCt := countClasses(Y, nClasses)

  var gini float64
  for _, ct := range classCt {
    p := float64(ct) / float64(len(Y))
    gini += p * p
  }

  return 1.0 - gini
}
```

```go
func partitionOnFeatureVal(X [][]float64, Y []int, splitVar int, splitVal float64)
 ([][]float64, [][]float64, []int, []int) {

  i := 0
  j := len(X)
  for i < j {
    if X[i][splitVar] < splitVal {
      i++
    } else {
      j--
      X[j], X[i] = X[i], X[j]
      Y[j], Y[i] = Y[i], Y[j]
    }
  }
  return X[:i], X[i:], Y[:i], Y[i:]
}
```

pros

- interpretable output
- mixed categorical and numeric data (not in the example shown though)
- robust to noise, outliers, mislabeled data
- account for complex interactions between input variables (limited by depth of tree)
- fairly easy to implement

cons

- prone to overfitting
- not particularly fast
- sensitive to input data (high variance)
- tree learning is NP-Complete (practical algos are typically greedy)

Random Forest

Condorcet's Jury Theorem

If each voter has an independent probability $p > 0.5$ of voting for the correct decision, then adding more voters increases the probability that the majority decision is correct.

the idea

Improve on vanilla decision trees by averaging the predictions of many trees.

the catch

The predictions of each tree must be independent of the predictions of all the other trees.

the “random” in random forest

Decorrelate the trees by introducing some randomness in the learning algorithm.

- fit each tree on a random sample of the training data (bagging/bootstrap aggregating)
- only evaluate sa random subset of the input features when searching for the best split

```go
func (t *Tree) findBestSplit(X [][]float64, Y []int) (float64, int, float64) {
  var (
    bestFeature int
    bestVal     float64
    bestGain    float64
  )

  initialImpurity := giniImpurity(Y, len(t.ClassNames))

  for feature := randomSample(t.K, t.NFeatures) {
    gain, val := findSplitOnFeature(X, Y, feature, len(t.ClassNames), initialImpurity)

    if gain > bestGain {
      bestGain = gain
      bestFeature = feature
      bestVal = val
    }
  }

  return bestGain, bestFeature, bestVal
}
```

```go
func (f *Forest) Fit(X [][]float64, Y []string) {
  for i := 0; i < f.NTrees; i++ {
    x, y := bootstrapSample(X, Y)
    t := NewTree().Fit(x, y)
    f.Trees = append(f.Trees, t)
  }
}
```

some Ml libraries

- Scikit-Learn (python)
- R
- Vowpal Wabbit (C++)
- MLlib (Spark/Scala)
- GoLearn (Go)
- CloudForest (Go)

parting thoughts

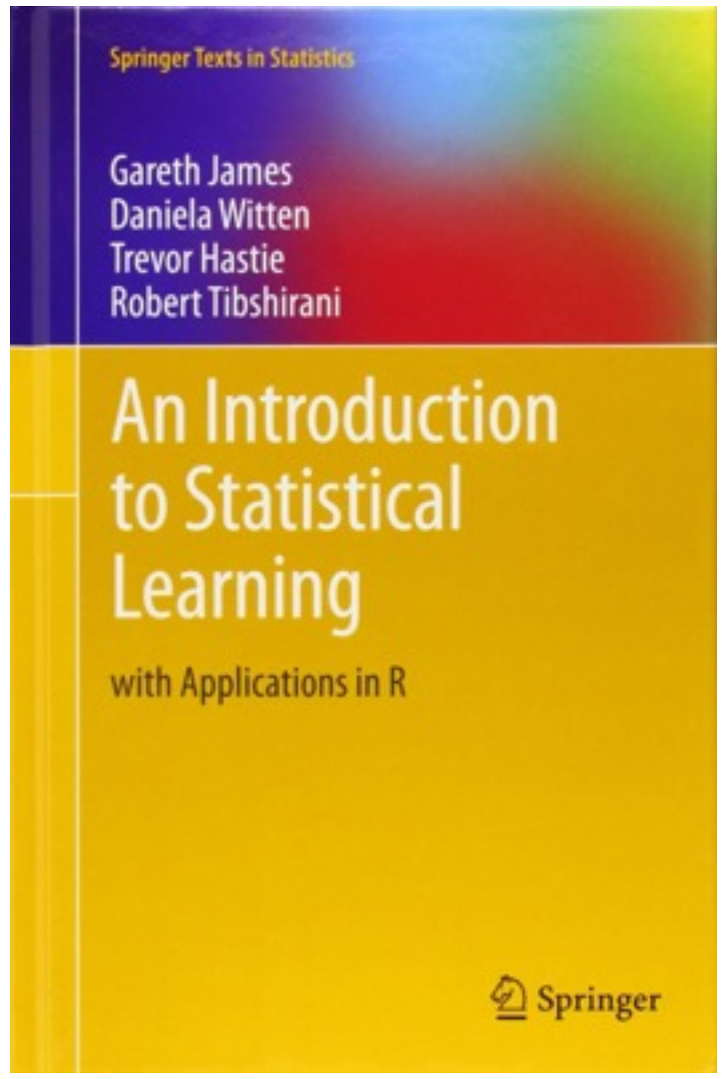
Take inspiration from the Scikit-Learn API:

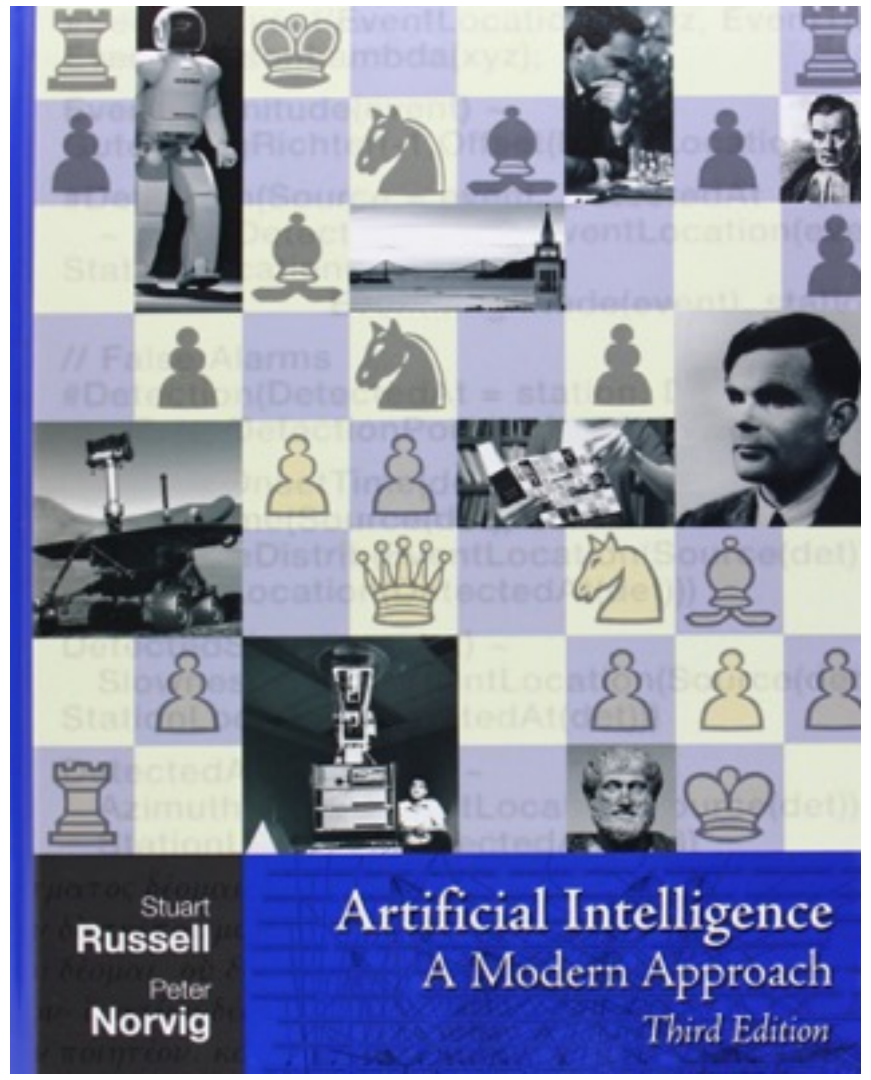
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(min_samples_split=20)
clf.fit(X,Y)
```

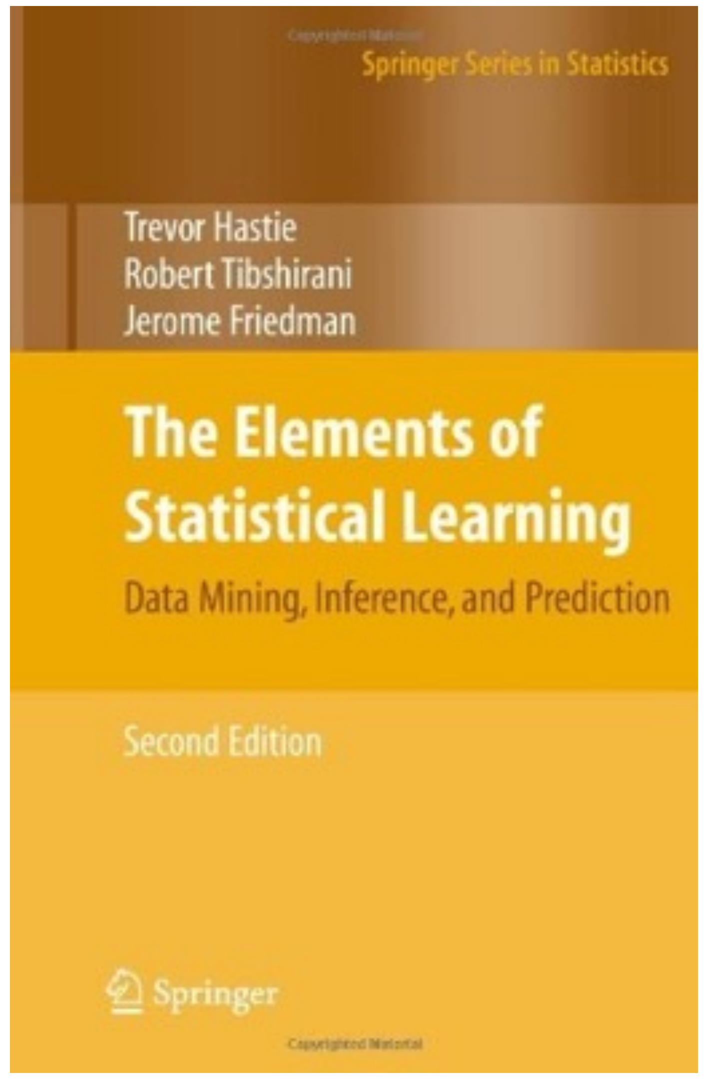
Compare to the signature for a similar model in GoLearn:

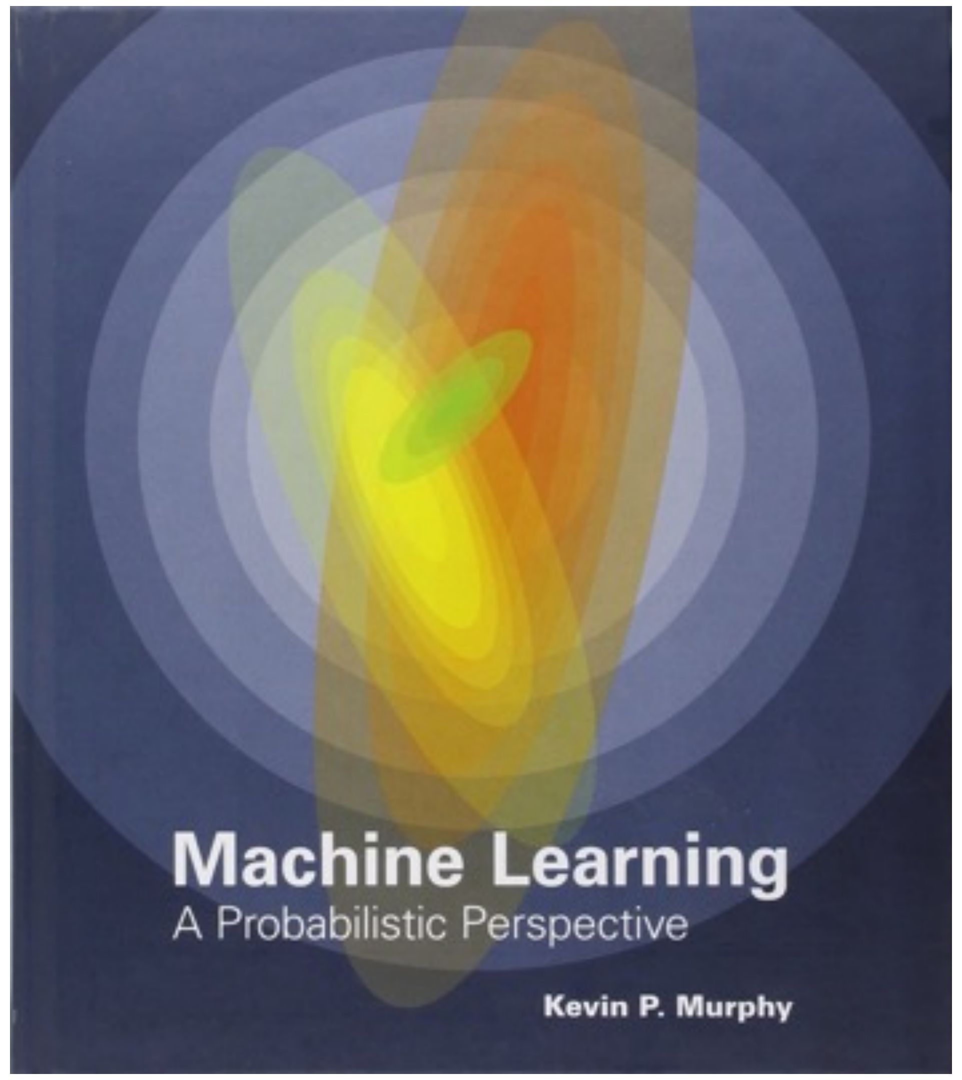
```
func (t *ID3DecisionTree) Fit(on base.FixedDataGrid) error
```

resources









1. An Introduction to Statistical Learning
2. Artificial Intelligence: A Modern Approach
3. The Elements of Statistical Learning
4. Machine Learning: A Probabilistic Perspective
5. Understanding Random Forests: From Theory to Practice

thanks.