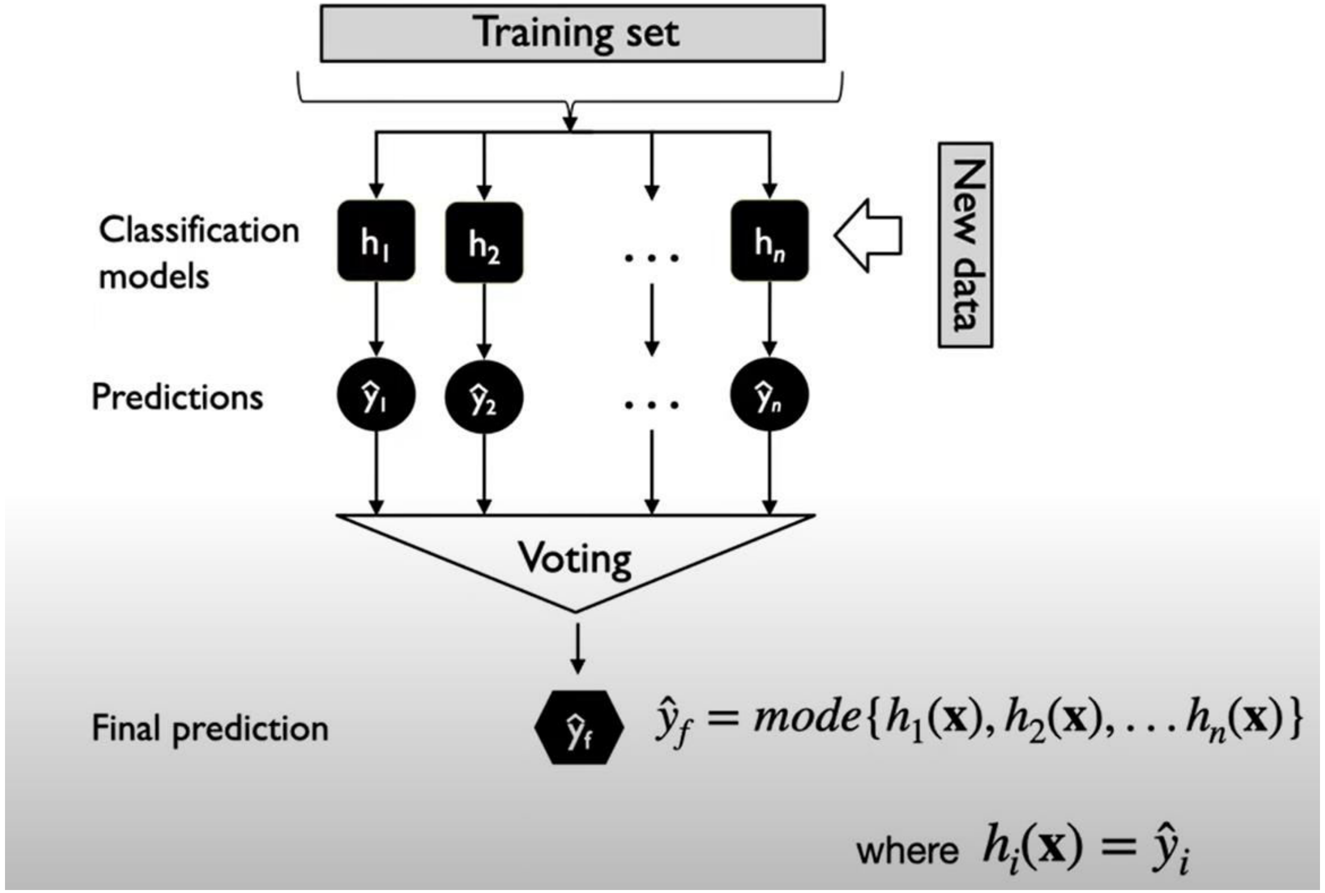
# Ensemble Methods

# Ensemble Methods

- Ensemble methods are techniques that aim at improving the accuracy of results in models by combining multiple models instead of using a single model.

# 1. Majority Voting Classifier



# 1. Majority Voting Classifier

## "Soft" Voting

$$\hat{y} = \arg\max_{j} \sum_{i=1}^{n} w_i p_{i,j}$$

$p_{i,j}$ : predicted class membership probability of the *i*th classifier for class label *j*

$w_i$ : optional weighting parameter, default $w_i = 1/n, \forall w_i \in \{w_1, \ldots, w_n\}$

# 1. Majority Voting Classifier

## "Soft" Voting

$$\hat{y} = \arg\max_{j} \sum_{i=1}^{n} w_i p_{i,j}$$

Binary classification example

$j \in \{0,1\}$ $\quad$ $h_i (i \in \{1,2,3\})$

$$h_1(\mathbf{x}) \rightarrow [0.9, 0.1]$$
$$h_2(\mathbf{x}) \rightarrow [0.8, 0.2]$$
$$h_3(\mathbf{x}) \rightarrow [0.4, 0.6]$$

# 1. Majority Voting Classifier

**"Soft" Voting** $\hat{y} = \arg\max_j \sum_{i=1}^{n} w_i p_{i,j}$

Binary classification example

$j \in \{0,1\} \qquad h_i(i \in \{1,2,3\})$

$$h_1(\mathbf{x}) \rightarrow [0.9,0.1]$$
$$h_2(\mathbf{x}) \rightarrow [0.8,0.2]$$
$$h_3(\mathbf{x}) \rightarrow [0.4,0.6]$$

$$p(j=0 \mid \mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58$$

$$p(j=1 \mid \mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42$$

$$\hat{y} = \arg\max_j \left\{ p(j=0 \mid \mathbf{x}), p(j=1 \mid \mathbf{x}) \right\}$$

# 1. Majority Voting Classifier (code)

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.datasets import *
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

clf1 = LogisticRegression(random_state=1)
clf2 = RandomForestClassifier(n_estimators=50, random_state=1)
clf3 = GaussianNB()

votc = VotingClassifier(estimators=[
        ('lr', clf1), ('rf', clf2), ('gnb', clf3)], voting='hard')
votc=votc.fit(X_train,y_train)
print('Accuracy of Voting Classifier: {:.2f}'.format(votc.score(X_test, y_test)))
```

```
Accuracy of Voting Classifier: 0.98
```

# Voting Regressor

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingRegressor
from sklearn.neighbors import KNeighborsRegressor
r1 = LinearRegression()
r2 = RandomForestRegressor(n_estimators=10, random_state=1)
r3 = KNeighborsRegressor()
X = np.array([[1, 1], [2, 4], [3, 9], [4, 16], [5, 25], [6, 36]])
y = np.array([2, 6, 12, 20, 30, 42])
er = VotingRegressor([('lr', r1), ('rf', r2), ('r3', r3)])
print(er.fit(X, y).predict(X))
```

```
[ 6.86666667  8.46666667 12.53333333 17.8        26.         34.2       ]
```

# 2. Bagging

(Bootstrap Aggregating)

**Algorithm 1** Bagging

Let $n$ be the number of bootstrap samples

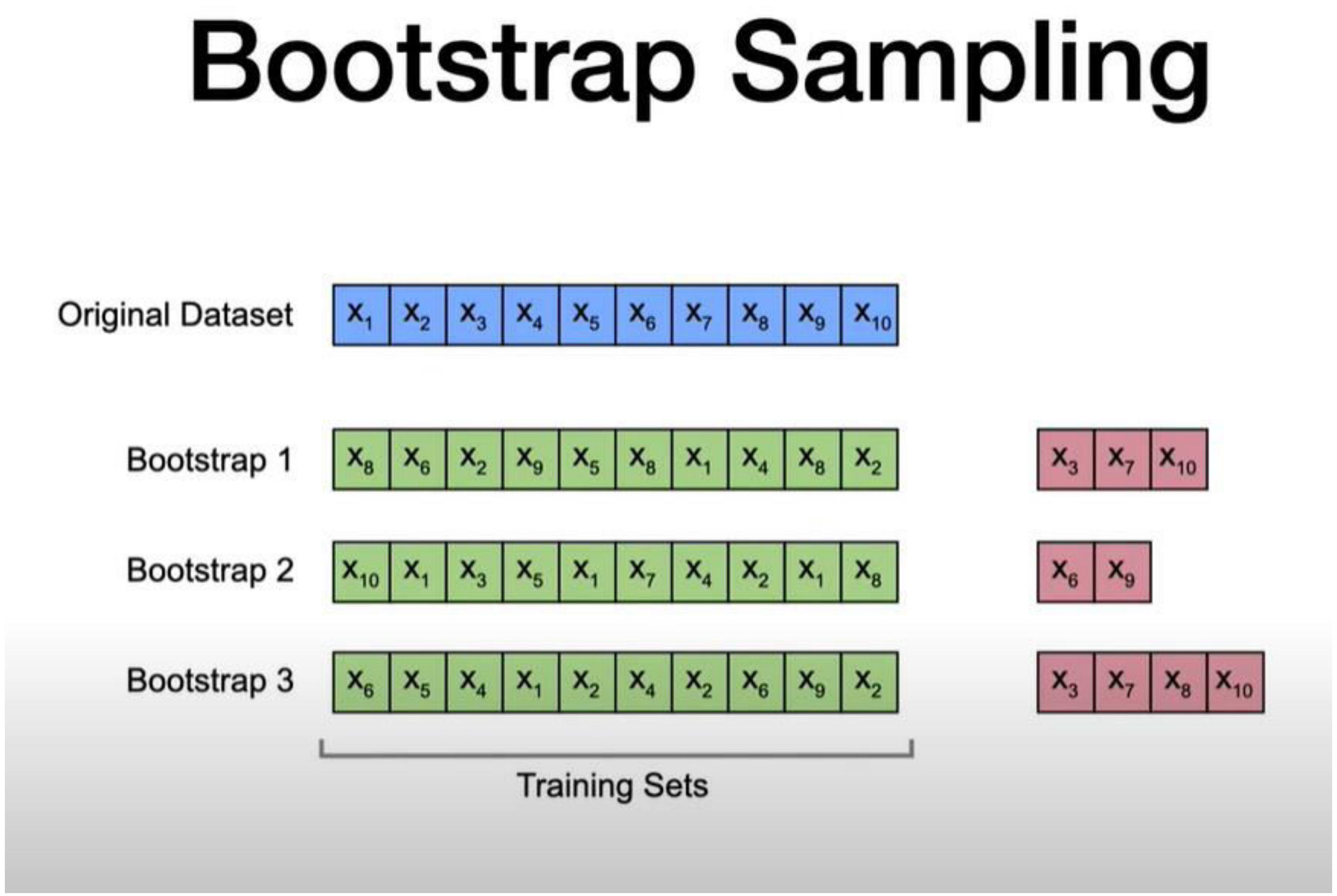
**for** i=1 to $n$ **do**

  Draw bootstrap sample of size $m$, $\mathcal{D}_i$

  Train base classifier $h_i$ on $\mathcal{D}_i$

$\hat{y} = mode\{h_1(\mathbf{x}), ..., h_n(\mathbf{x})\}$

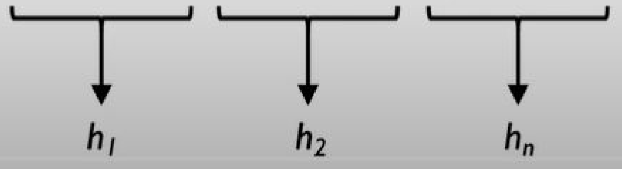
# 2. Bagging


Bootstrap Sampling
Original Dataset
$X_1$ $X_2$ $X_3$ $X_4$ $X_5$ $X_6$ $X_7$ $X_8$ $X_9$ $X_{10}$
Bootstrap 1
$X_8$ $X_6$ $X_2$ $X_9$ $X_5$ $X_8$ $X_1$ $X_4$ $X_8$ $X_2$
$X_3$ $X_7$ $X_{10}$
Bootstrap 2
$X_{10}$ $X_1$ $X_3$ $X_5$ $X_1$ $X_7$ $X_4$ $X_2$ $X_1$ $X_8$
$X_6$ $X_9$
Bootstrap 3
$X_6$ $X_5$ $X_4$ $X_1$ $X_2$ $X_4$ $X_2$ $X_6$ $X_9$ $X_2$
$X_3$ $X_7$ $X_8$ $X_{10}$
Training Sets

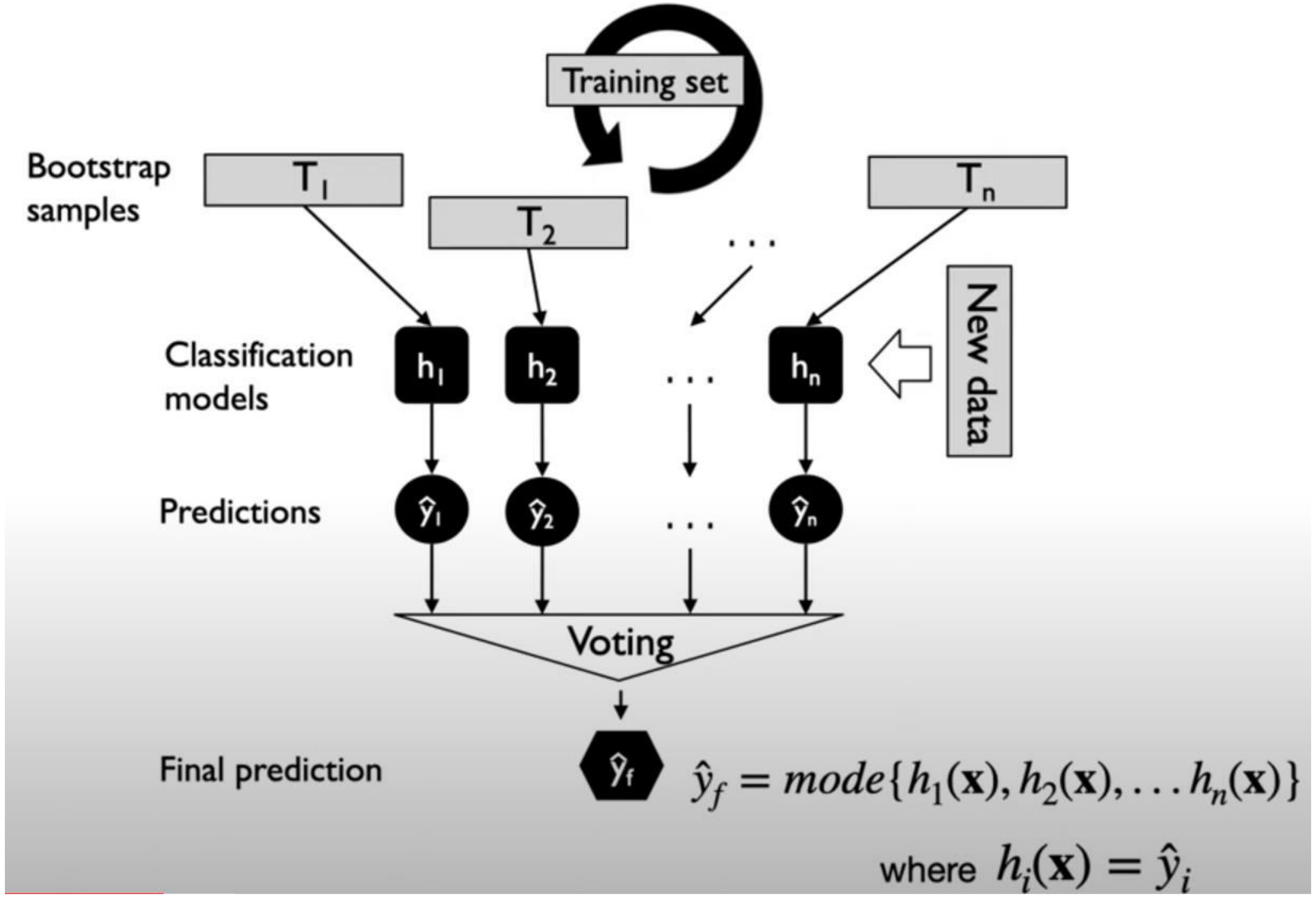
# 2. Bagging

## Bootstrap Sampling

| Training example indices | Bagging round 1 | Bagging round 2 | ... |
|---|---|---|---|
| 1 | 2 | 7 | ... |
| 2 | 2 | 3 | ... |
| 3 | 1 | 2 | ... |
| 4 | 3 | 1 | ... |
| 5 | 7 | 1 | ... |
| 6 | 2 | 7 | ... |
| 7 | 4 | 7 | ... |



# 2. Bagging



Training set
Bootstrap samples
$T_1$
$T_2$
$T_n$
Classification models
$h_1$
$h_2$
$h_n$
New data
Predictions
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_n$
Voting
Final prediction
$\hat{y}_f$
$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \dots h_n(\mathbf{x})\}$
where $h_i(\mathbf{x}) = \hat{y}_i$

# 2. Bagging (code)

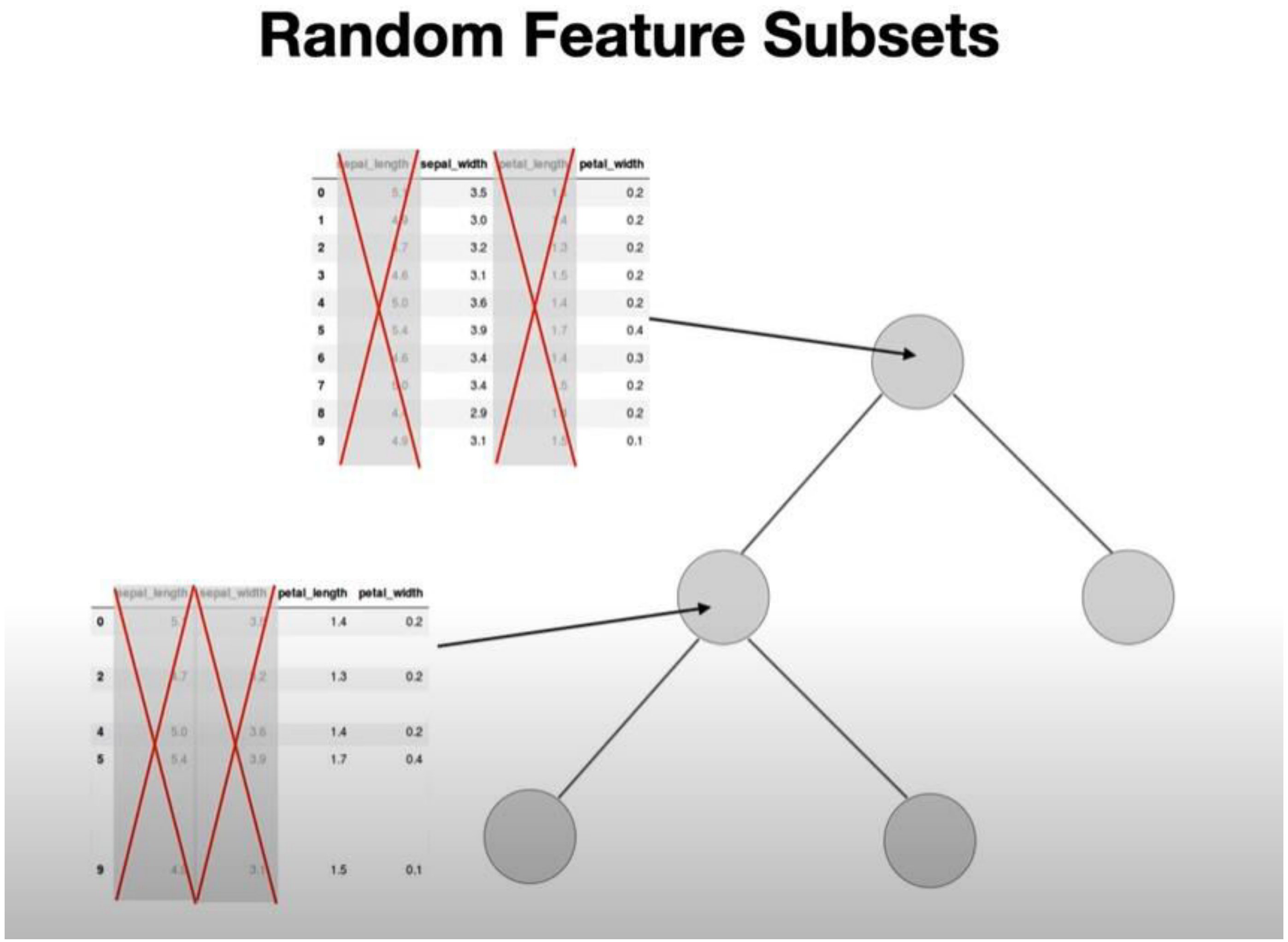
```
from sklearn import model_selection
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import datasets
from sklearn.ensemble import BaggingClassifier

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])

tree = DecisionTreeClassifier(criterion='entropy',
                              random_state=1,
                              max_depth=None)


bag = BaggingClassifier(base_estimator=tree,
                        n_estimators=500,
                        oob_score=True,
                        bootstrap=True,
                        bootstrap_features=False,
                        n_jobs=1,
                        random_state=1)

bag.fit(X_train, y_train)

print("OOB Accuracy: %0.2f" % bag.oob_score_)
print("Test Accuracy: %0.2f" % bag.score(X_test, y_test))
```

```
Train/Valid/Test sizes: 84 28 38
OOB Accuracy: 0.93
Test Accuracy: 0.95
```

# Random forests

= Bagging w. trees + random feature subsets

# Random forests



# Random forests

num features $= \log_2 m + 1$

where *m* is the number of input features

## Random Feature Subset for each Tree or Node?

Tin Kam Ho used the "**random subspace method,**" where each tree got a random subset of features.

"Our method relies on an autonomous, pseudo-random procedure to select a small number of dimensions from a given feature space …"

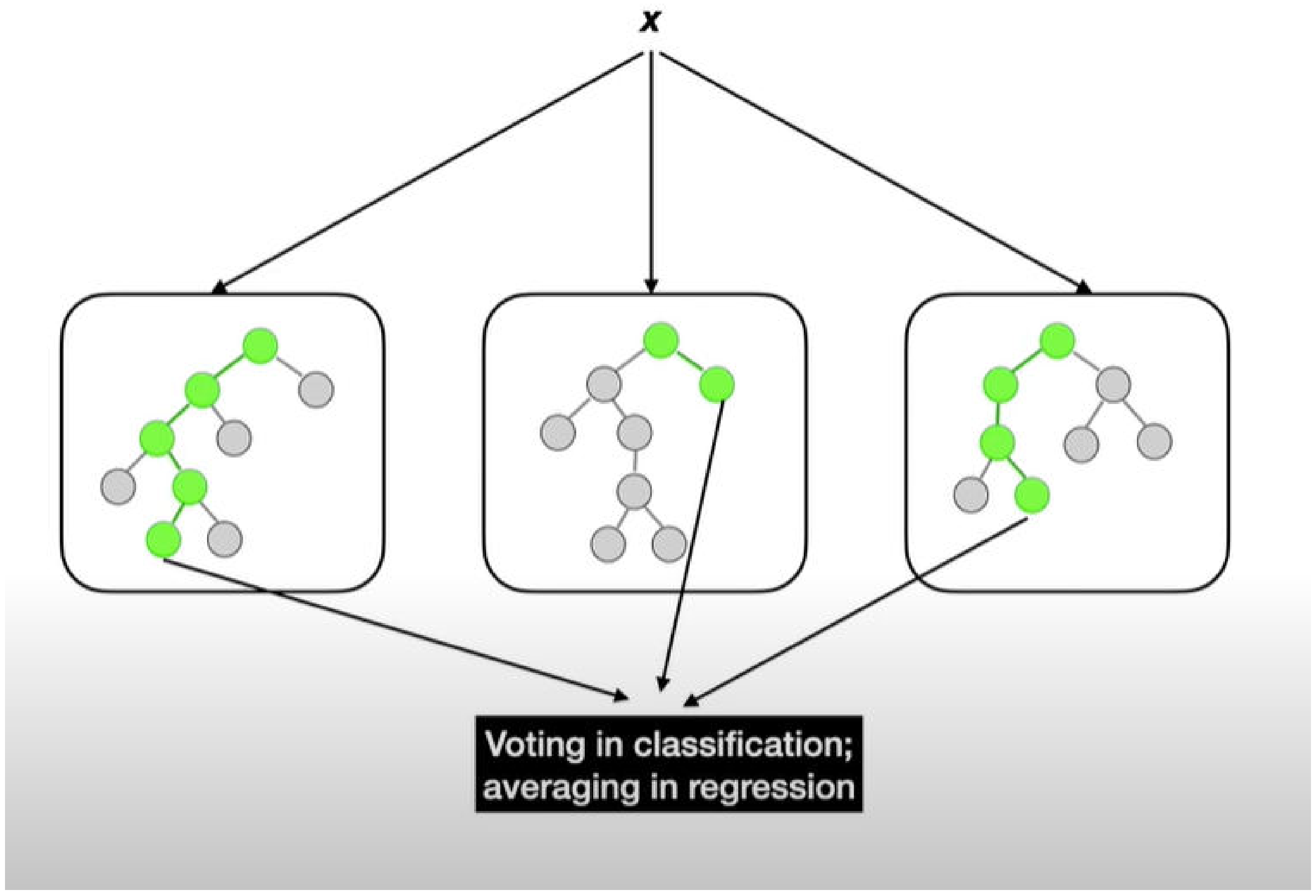
- Ho, Tin Kam. "The random subspace method for constructing decision forests." IEEE transactions on pattern analysis and machine intelligence 20.8 (1998): 832-844.

**"Trademark" random forest:**

*"… random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on."*

- Breiman, Leo. "Random Forests" Machine learning 45.1 (2001): 5-32.

# Random forests



# Random forests

In contrast to the original publication
[Breiman, “Random Forests”, Machine Learning, 45(1), 5-32, 2001]
the scikit-learn implementation combines classifiers by averaging their probabilistic prediction, instead of letting each classifier vote for a single class.

"Soft Voting"

# Random forests (code)

```
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import datasets

iris = datasets.load_iris()
X, y = iris.data[:, [0, 3]], iris.target

X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.25, random_state=1)

X_train, X_val, y_train, y_val = \
    train_test_split(X_train, y_train, test_size=0.25, random_state=1)

print('Train/Valid/Test sizes:', y_train.shape[0], y_val.shape[0], y_test.shape[0])
```

```
Train/Valid/Test sizes: 84 28 38
```

```
from sklearn.ensemble import RandomForestClassifier


forest = RandomForestClassifier(n_estimators=100,
                                random_state=1)

forest.fit(X_train, y_train)

print("Test Accuracy: %0.2f" % forest.score(X_test, y_test))
```

```
Test Accuracy: 0.95
```