DATASCI207-005/007 Applied Machine Learning

Vilena Livinsky, PhD(c)

School of Information, UC Berkeley

Week 7: 02/19/2025 & 02/20/2025

Today's Agenda

- KNN
- Decision Trees
- Ensemble Learning
- Walkthroughs/ Practice:
  - KNN
  - Decision Trees
  - Ensemble Learning

KNN: "Lazy Learner"

Train Step:

- Remember the train dataset
- Model: Nonparametric? Parametric?

Prediction Step:

- predict label of new data point ("Majority Vote")
- KNN algorithm
- Distance Measures
  - Scale your features!
  - Ex: Euclidean Distance (L2):
    - Larger distance values will dominate
  - https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.distance_metrics.html#sklearn.metrics.pairwise.distance_metrics

***Higher dimensional spaces: everything is further apart**

- Neighbours become less similar
- Density decreases with the number of dimensions

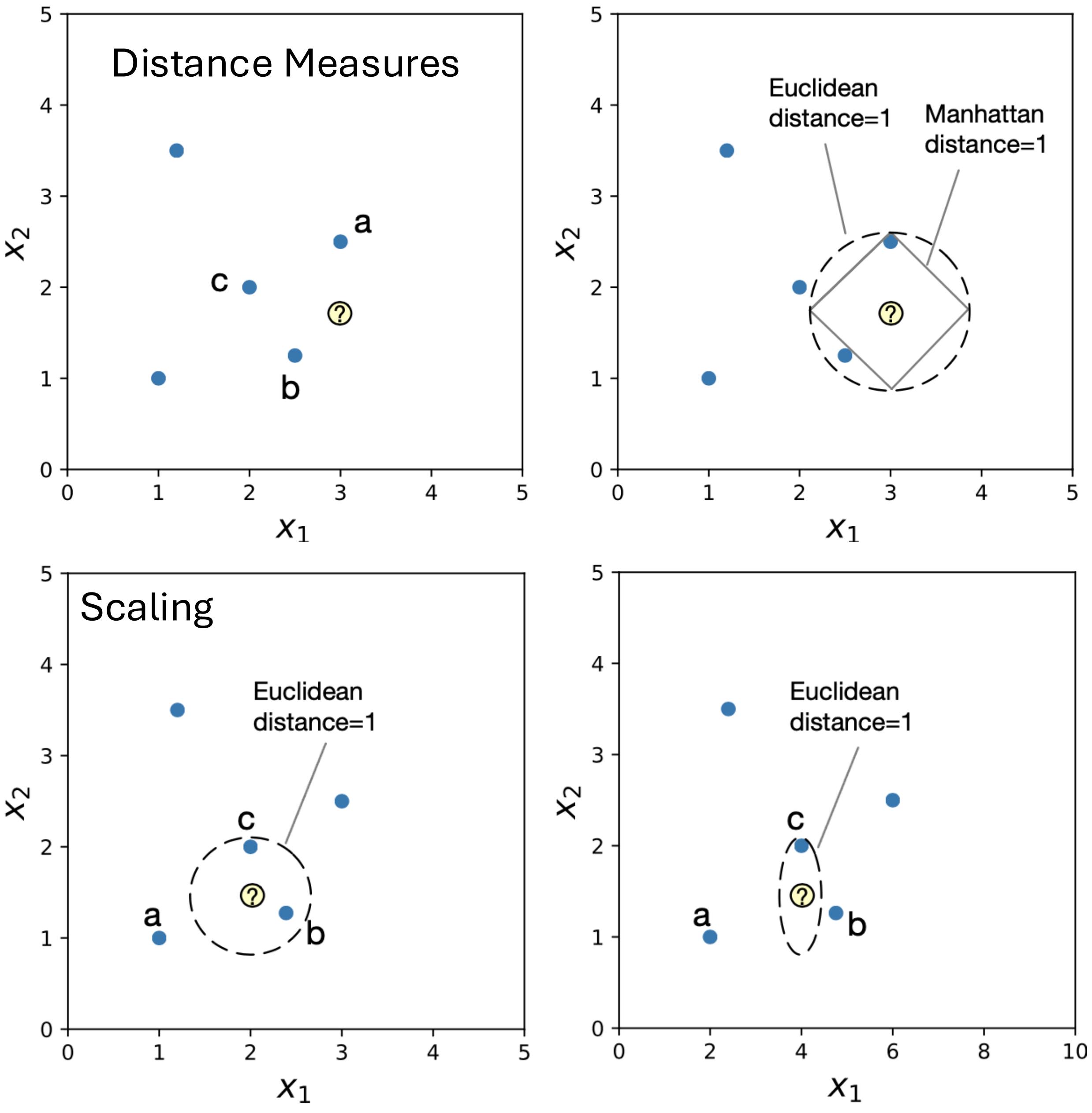


Image Source: Raschka S., Introduction to Machine Learning

k in kNN

- choosing k
  - Consider how smooth a boundary would be the larger/ smaller the k is
  - Ex.: Binary Classification
    - squares vs triangles
  - Where’s k = 1?

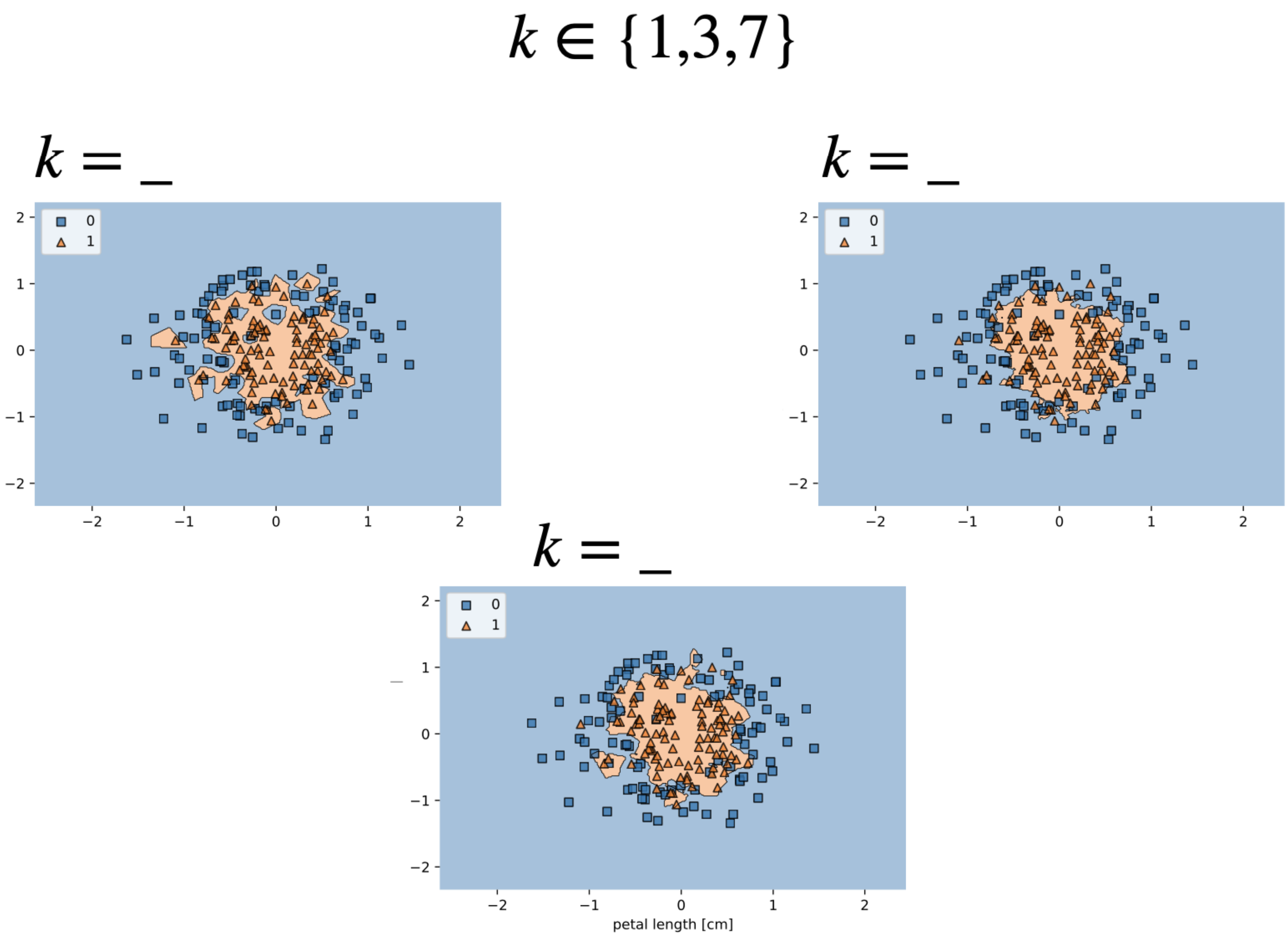


Image Source: Raschka S., Introduction to Machine Learning

Decision Trees

Decision Trees

Goal: Split nodes at the most informative features

- objective function: maximize the Information Gain at each split

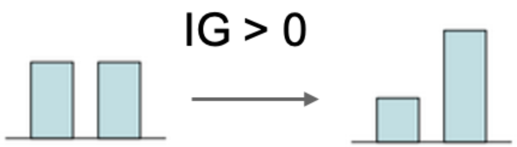
Impurity:

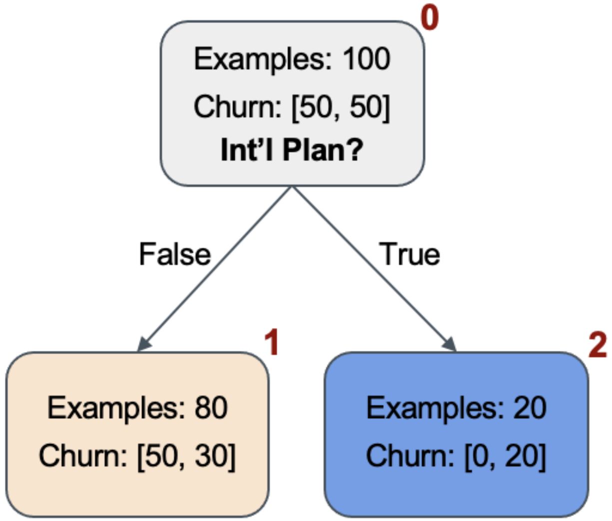
- Entropy
- Gini
- Both maximal if the classes are perfectly mixed

to reduce the combinatorial search space, most libraries (ex. scikit-learn) implement binary decision trees

Information Gain

- IG = Entropy before - Entropy after
- Weighted by number of examples





$H_0 =$ -.5 log(.5)
-.5 log(.5) = **1**

$H_1 =$ -.625 log(.625)
-.375 log(.375) =
.95

$H_2 =$ -0 log(0)
-1 log(1) = **0**

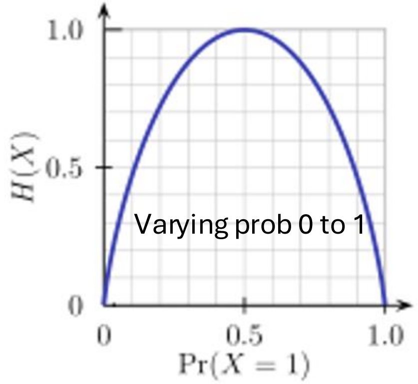
IG(Int'l Plan) =
1 - [.8(.95) + .2(0)] =
1 - .76 =
.24

Measuring "purity" of a split

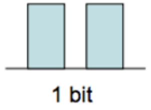
Certainty about Yes/No after a split:

pure set (5 yes, 0 no)	=	very much certain (100%)
impure (3 yes, 3 no)	=	very much uncertain (50%)

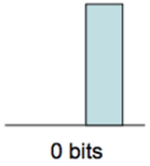
$$H = -\sum_i p_i (\log_2 p_i)$$



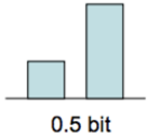
Uniform distribution (complete uncertainty) = Entropy is max

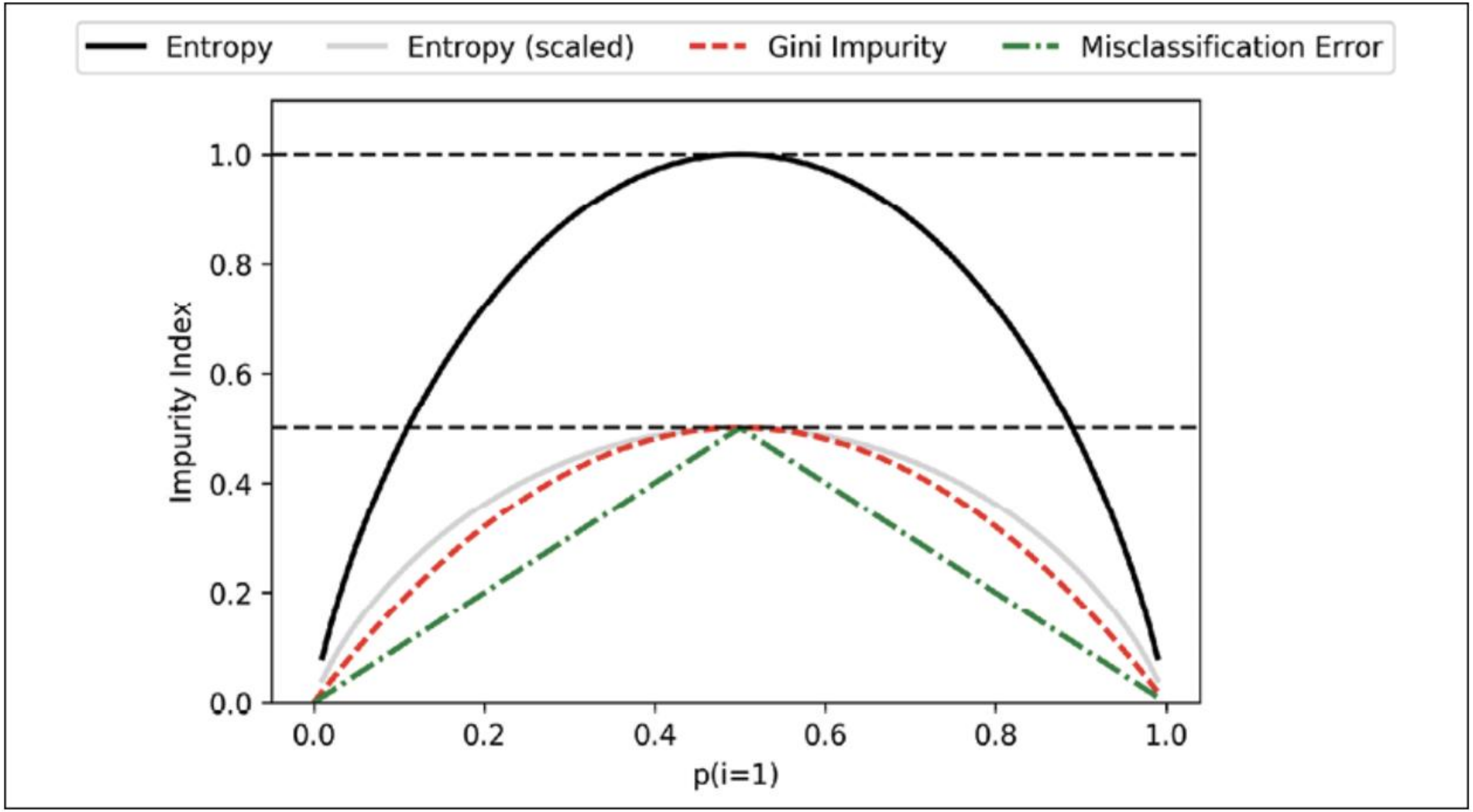


One distribution value has prob of 1 = Entropy is 0



Entropy btw. 0 & 1



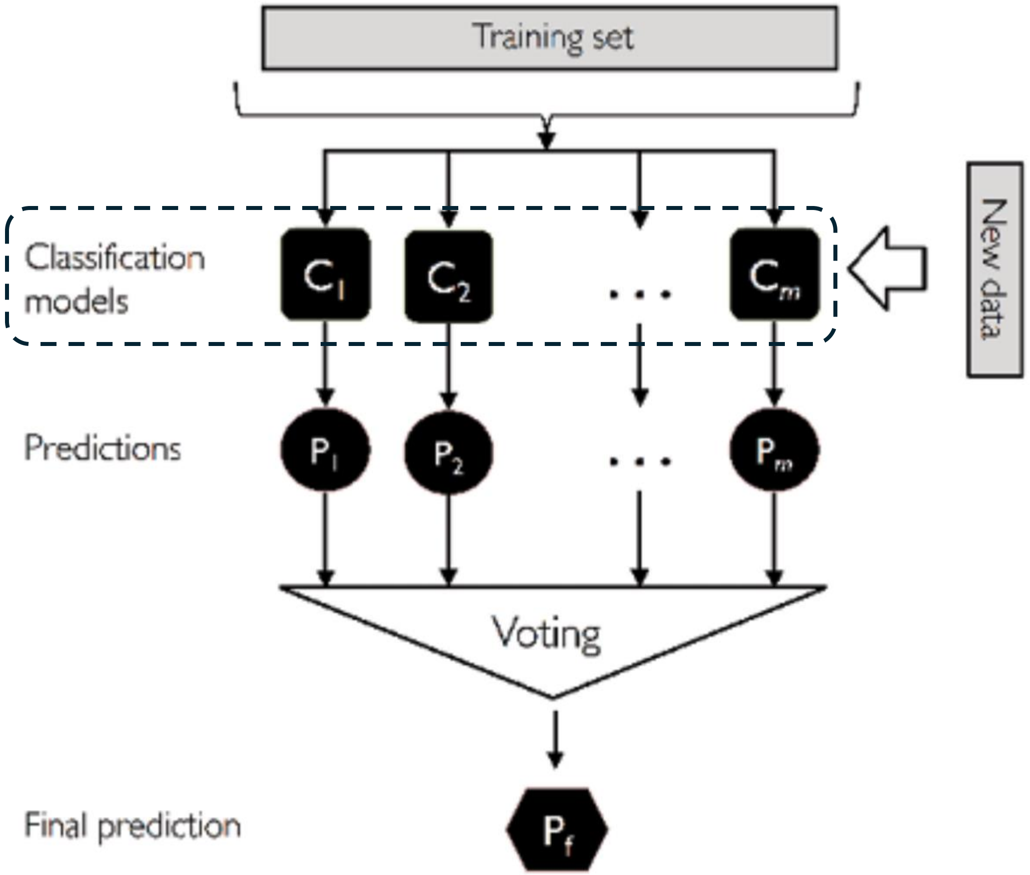

Entropy
Entropy (scaled)
Gini Impurity
Misclassification Error
Impurity Index
p(i=1)
0.0
0.2
0.4
0.6
0.8
1.0

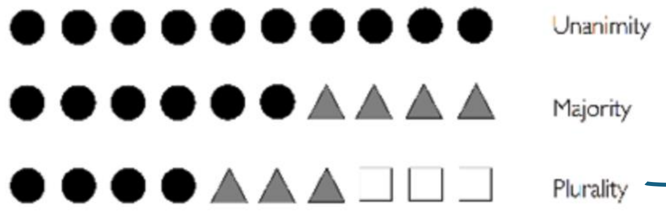
Ensemble Methods

Tree-based methods: Majority Voting, Bagging, Boosting (AdaBoost/ Gradient Boosting), Random Forests

Majority Voting

can be built from:	• different classification algorithms
can be built from:	• same base classification algorithm (ex.: different decision tree classifiers) fitting different subsets of the training dataset
Random Forests	• bagging (data sampling) with trees + random feature subsets at each node





$$\hat{y} = \text{mode}\{C_1(\boldsymbol{x}), C_2(\boldsymbol{x}), \ldots, C_m(\boldsymbol{x})\}$$

Image Source: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

"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 ith classifier for class label j

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

Binary classification example

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

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

$$p(j = 0 \mid \mathbf{x}) = 0.2 \cdot 0.9 + 0.2 \cdot 0.8 + 0.6 \cdot 0.4 = 0.58$$
$$p(j = 1 \mid \mathbf{x}) = 0.2 \cdot 0.1 + 0.2 \cdot 0.2 + 0.6 \cdot 0.6 = 0.42$$

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

Image (Edited) Source: Raschka S., Introduction to Machine Learning

- Scikit-learn default = “hard”

```python
from sklearn.datasets import make_moons
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X, y = make_moons(n_samples=500, noise=0.30, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

voting_clf = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression(random_state=42)),
        ('rf', RandomForestClassifier(random_state=42)),
        ('svc', SVC(random_state=42))
    ]
)
voting_clf.fit(X_train, y_train)
```

- To set to “soft”:

```python
voting_clf.voting = "soft"
voting_clf.named_estimators["svc"].probability = True
voting_clf.fit(X_train, y_train)
voting_clf.score(X_test, y_test)
```

- Example Implementation:
- https://github.com/ageron/handson-ml3/blob/main/07_ensemble_learning_and_random_forests.ipynb

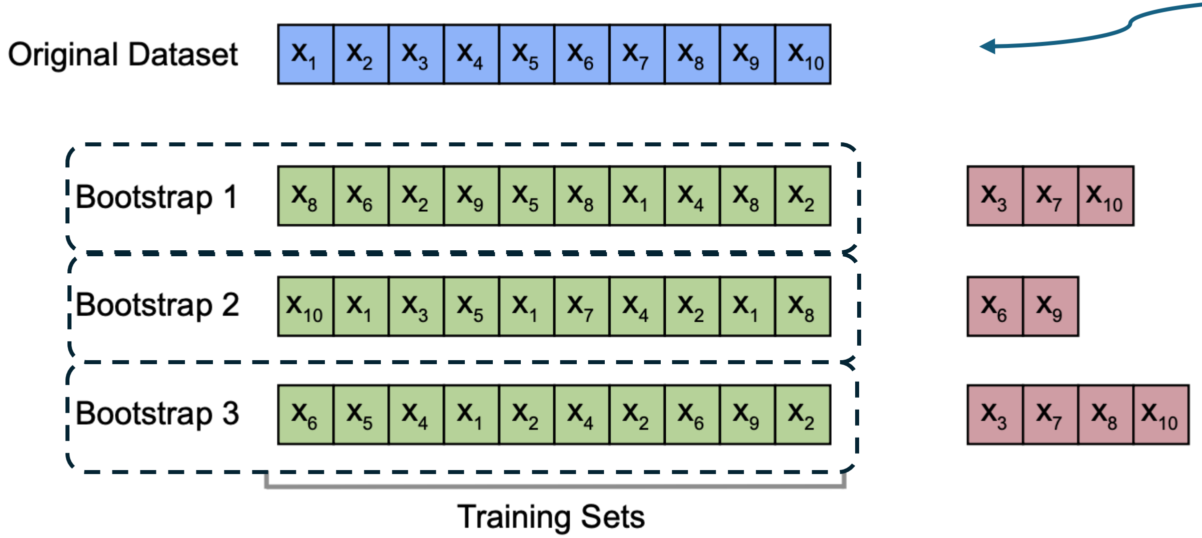
Example

- Example - Classifying Iris Flowers Using Different Classification Models (Raschka, mlxtend)
  - Notebook: https://rasbt.github.io/mlxtend/user_guide/classifier/EnsembleVoteClassifier/

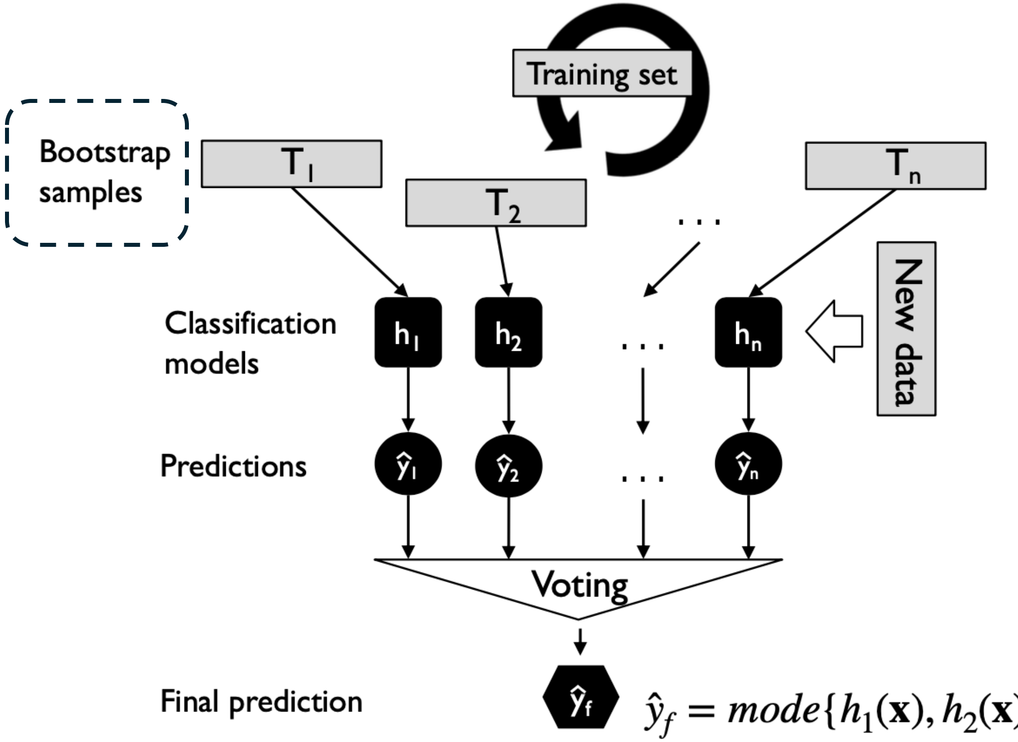
Bagging: Bootstrap Aggregating

Bootstrap Sampling

- No dependency between sampling rounds



Bagging Classifier



$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

where $h_i(\mathbf{x}) = \hat{y}_i$

Image (Edited) Source: Raschka S., Introduction to Machine Learning

Bagging: Bootstrap Aggregating

Bootstrap Sampling

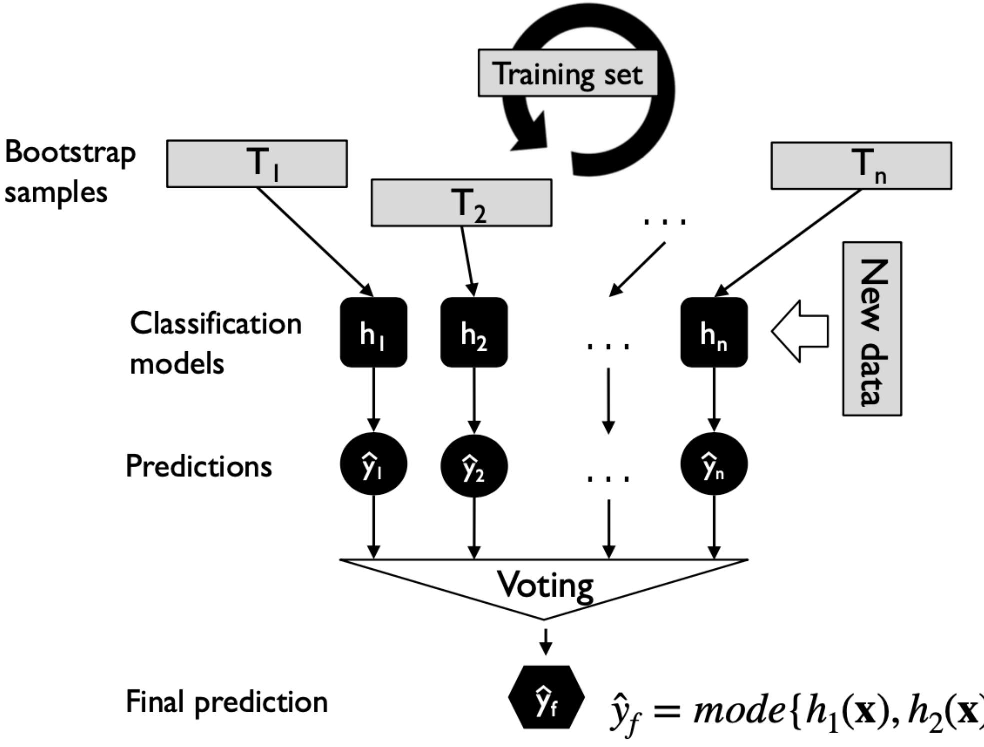
- No dependency between sampling rounds

Sample 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	...

C_1 C_2 C_m

Image Source: Raschka S., Introduction to Machine Learning

Bagging Classifier



$$\hat{y}_f = mode\{h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots h_n(\mathbf{x})\}$$

where $h_i(\mathbf{x}) = \hat{y}_i$

Bagging: Implementation (Scikit-learn)

- Pass a DecisionTreeClassifier to Bagging

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

- Evaluate

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

- Example:
- https://github.com/ageron/handson-ml2/blob/master/07_ensemble_learning_and_random_forests.ipynb

Decision Tree Classifier alone:

```
tree_clf = DecisionTreeClassifier(random_state=42)
tree_clf.fit(X_train, y_train)
y_pred_tree = tree_clf.predict(X_test)
print(accuracy_score(y_test, y_pred_tree))
```

Or

Random Forest (Bagging + Decision Trees):

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Boosting

- Ex.: Adaptive Boosting (AdaBoost)
- Ex.: Gradient Boosting
- **weak learners** (build very simple base classifiers)
  - Those with slightly better performance over random guessing
    - Ex.: decision tree stump
  - **Goal**: focus on training examples that are hard to classify
    - weak learners learn from misclassified training examples to improve the performance of the ensemble

Boosting (Original Procedure)

1. Draw a random subset (sample) of training examples, d_1, without replacement from the training dataset, D, to train a weak learner, C_1.
2. Draw a second random training subset, d_2, without replacement from the training dataset and add 50 percent of the examples that were previously misclassified to train a weak learner, C_2.
3. Find the training examples, d_3, in the training dataset, D, which C_1 and C_2 disagree upon, to train a third weak learner, C_3.
4. Combine the weak learners C_1, C_2, and C_3 via majority voting.

AdaBoost

1. Set the weight vector, $\boldsymbol{w}$, to uniform weights, where $\sum_i w_i = 1$.
2. For j in m boosting rounds, do the following:
    1. Train a weighted weak learner: $C_j = \text{train}(\boldsymbol{X}, \boldsymbol{y}, \boldsymbol{w})$.
    2. Predict class labels: $\hat{\boldsymbol{y}} = \text{predict}(C_j, \boldsymbol{X})$.
    3. Compute weighted error rate: $\varepsilon = \boldsymbol{w} \cdot (\hat{\boldsymbol{y}} \neq \boldsymbol{y})$.
    4. Compute coefficient: $\alpha_j = 0.5 \log \frac{1 - \boldsymbol{\varepsilon}}{\boldsymbol{\varepsilon}}$
    5. Update weights: $\boldsymbol{w} := \boldsymbol{w} \times \exp(-\alpha_j \times \hat{\boldsymbol{y}} \times \boldsymbol{y})$.
    6. Normalize weights to sum to 1: $\boldsymbol{w} := \boldsymbol{w} / \sum_i w_i$.
3. Compute the final prediction: $\hat{\boldsymbol{y}} = \left(\sum_{j=1}^{m} \left(\boldsymbol{\alpha}_j \times \text{predict}(C_j, \boldsymbol{X}) \right) > 0 \right)$.

Index	x	y	Weights	$\hat{y}$(x <= 3.0)?	Correct?	Updated weights
1	1.0	1	0.1	1	Yes	0.072
2	2.0	1	0.1	1	Yes	0.072
3	3.0	1	0.1	1	Yes	0.072
4	4.0	-1	0.1	-1	Yes	0.072
5	5.0	-1	0.1	-1	Yes	0.072
6	6.0	-1	0.1	-1	Yes	0.072
7	7.0	1	0.1	-1	No	0.167
8	8.0	1	0.1	-1	No	0.167
9	9.0	1	0.1	-1	No	0.167
10	10.0	-1	0.1	-1	Yes	0.072

Image Source: Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.