Day-I Session-II

Exploring Machine Learning Techniques

Dr. Sibarama Panigrahi

Department of Computer Science & Engineering

National Institute of Technology Rourkela

Outlines…

- Learning
- Machine Learning (ML)
- A loose taxonomy of ML
  - Supervised Learning
  - Unsupervised Learning
  - Reinforcement Learning
- ML Perspective
- Data and Features
- Performance Measures
- Decision Tree
- Random Forest

Learning

- Learning = Improving with experience at some task
  - Improve over task T
  - with respect to performance measure P
  - based on experience E

- E.g.
  - Task T = Predicting the Gender of a Person from a Photo
  - Performance measure=Classification Accuracy
  - Experience E

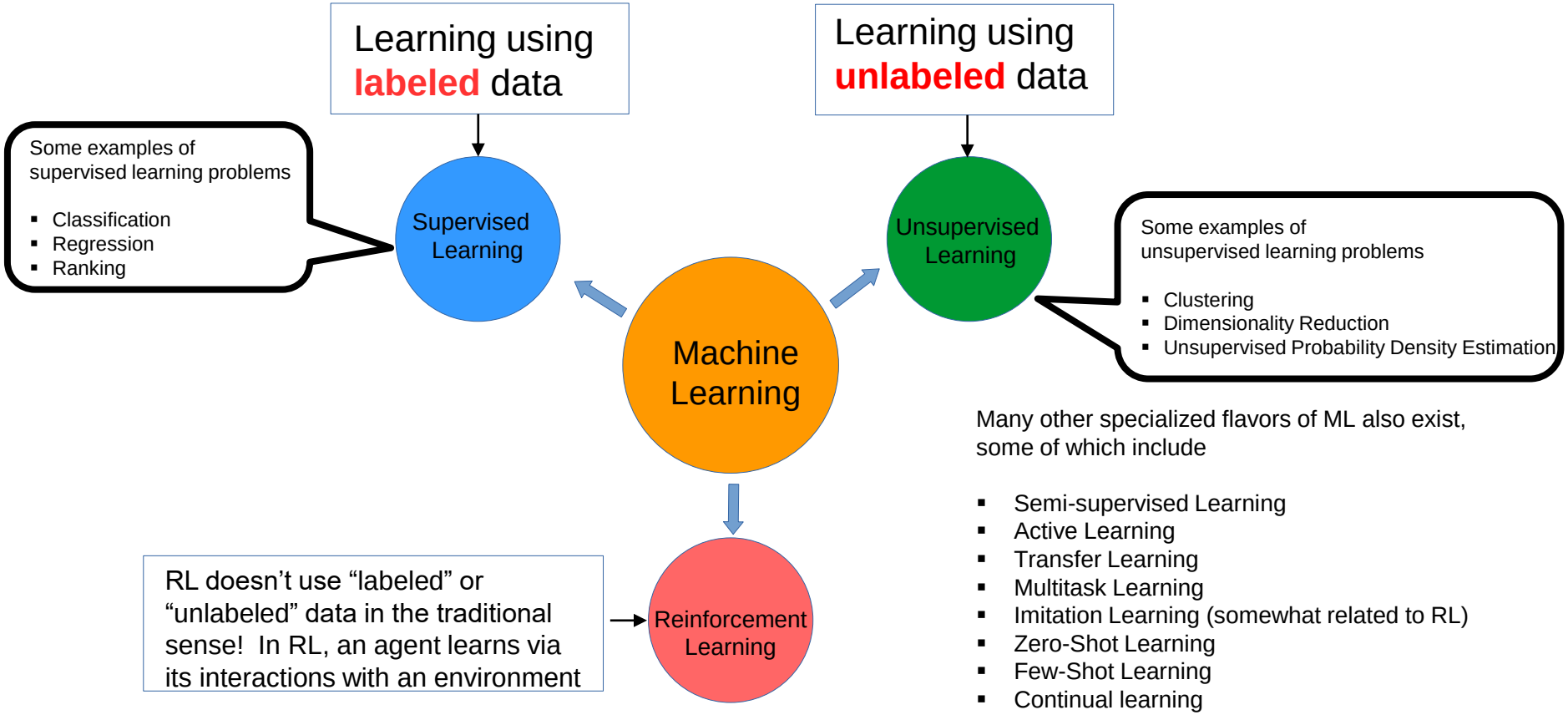
Machine Learning (ML)

- "field of study that gives computers the ability to learn without being explicitly programmed" (Samuel 1959)

Traditional Programming

Machine Learning

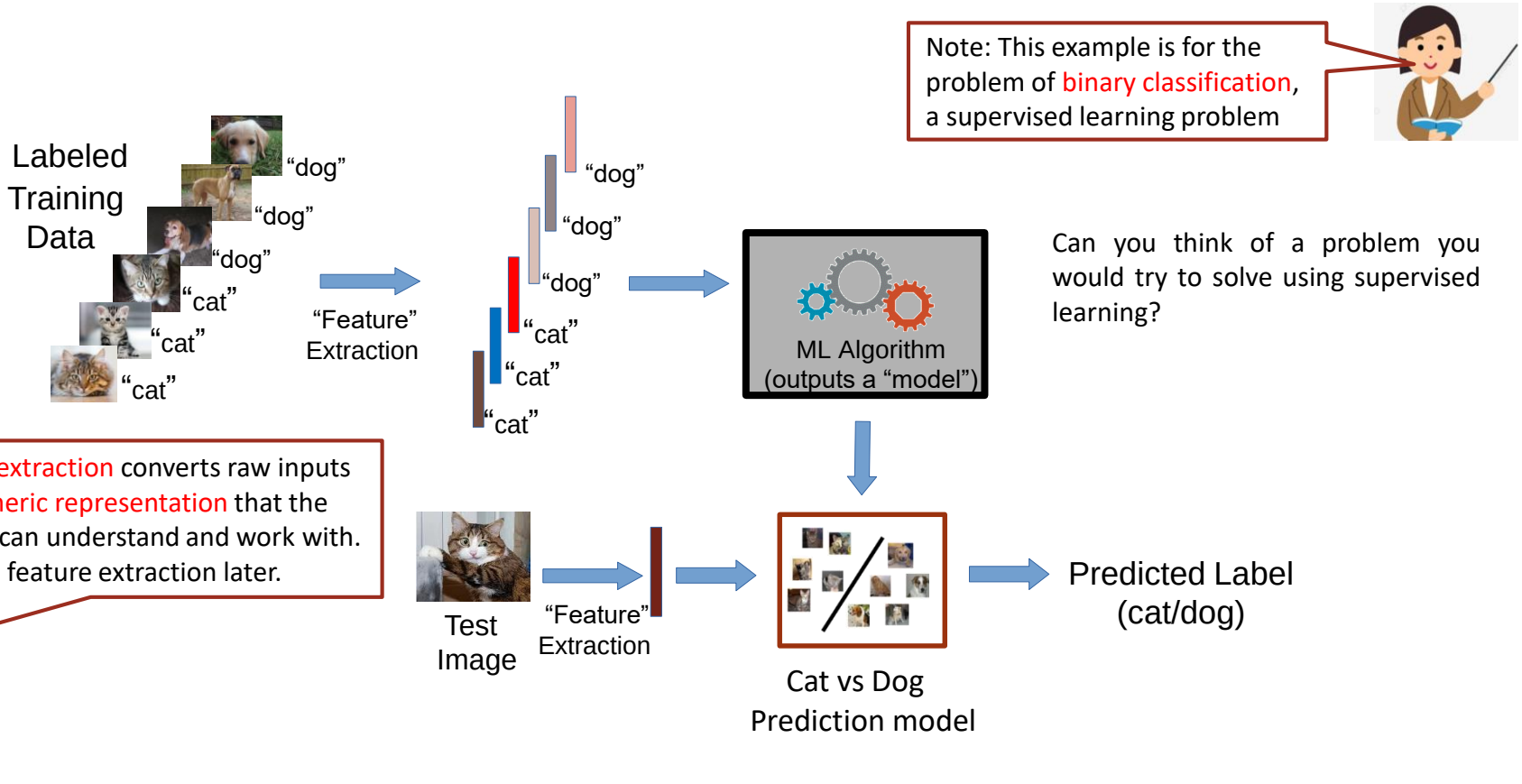
A Loose Taxonomy of ML



Many other specialized flavors of ML also exist, some of which include

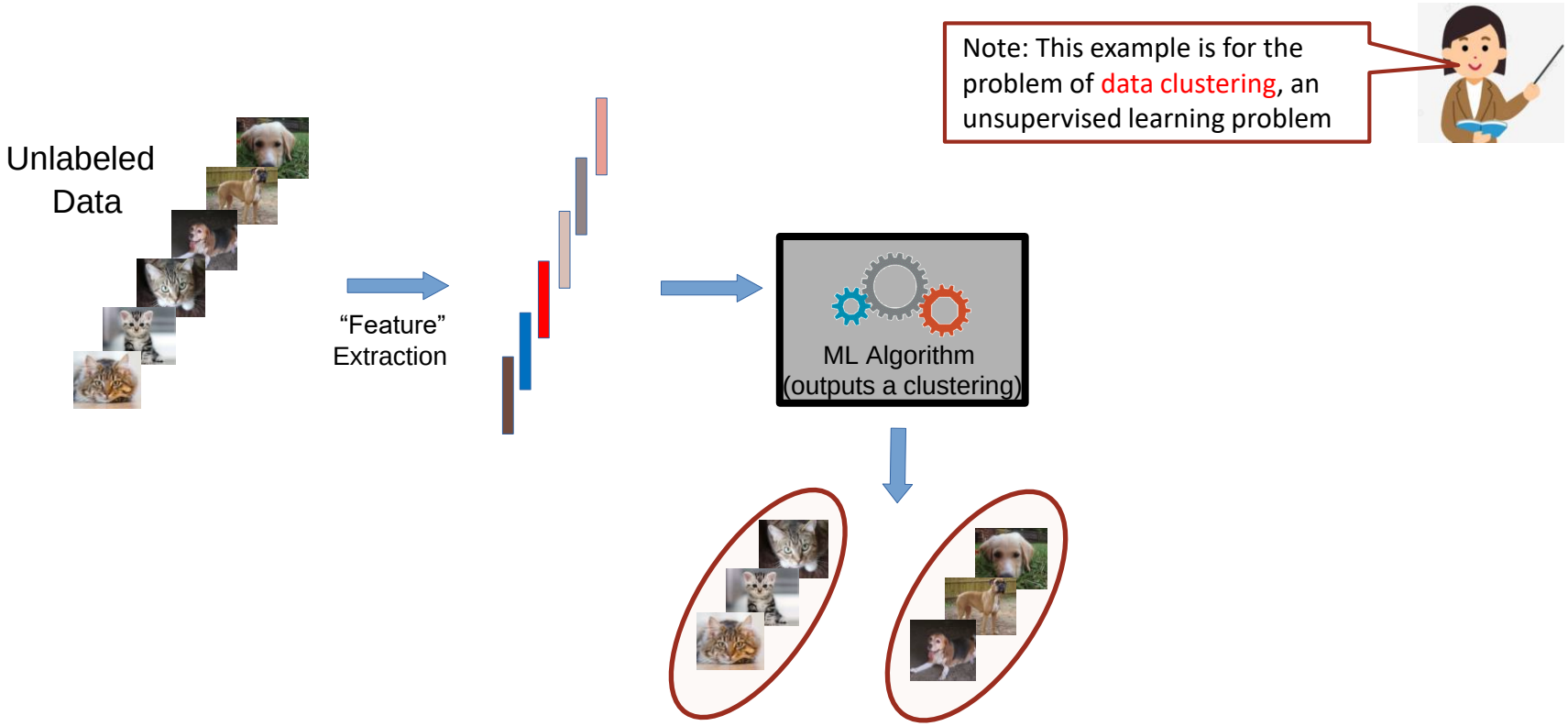
- Semi-supervised Learning
- Active Learning
- Transfer Learning
- Multitask Learning
- Imitation Learning (somewhat related to RL)
- Zero-Shot Learning
- Few-Shot Learning
- Continual learning

A Typical Supervised Learning Workflow



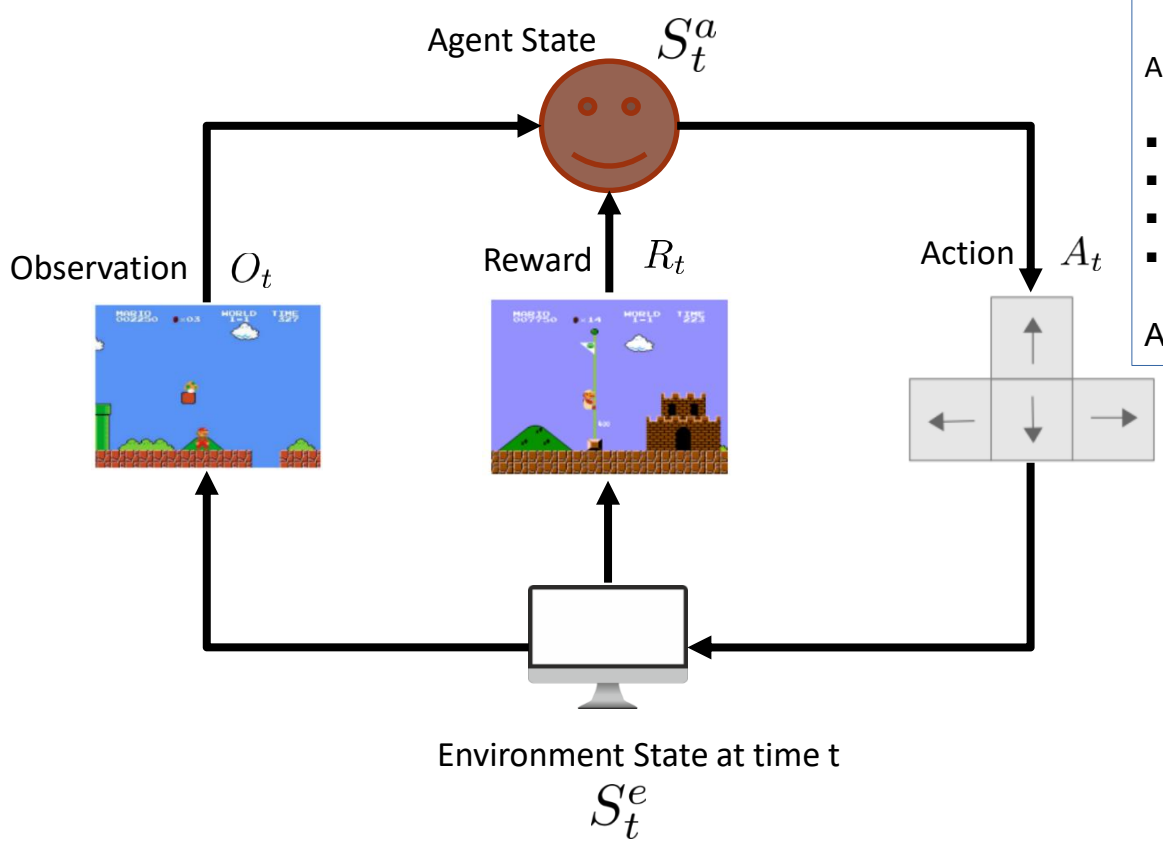
https://www.pinclipart.com/, http://www.pngtree.com

A Typical Unsupervised Learning Workflow



https://www.pinclipart.com/, http://www.pngtree.com

A Typical Reinforcement Learning Workflow



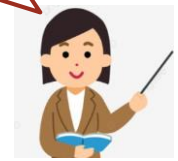
Wish to teach an agent optimal policy for some task

Agent does the following repeatedly

- Senses/observes the environment
- Takes an action based on its current policy
- Receives a reward for that action
- Updates its policy

Agent's goal is to maximize its overall reward

There IS supervision, not explicit (as in Supervised Learning) but rather implicit (feedback based)

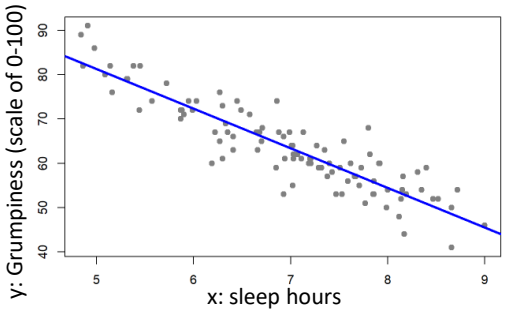


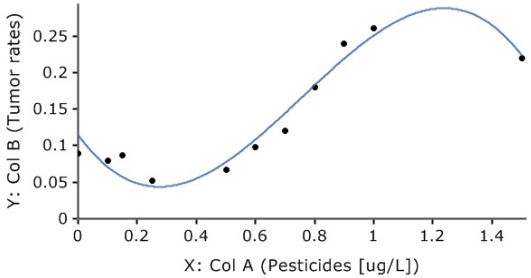
Geometric Perspective

Recall that feature extraction converts inputs into a numeric representation

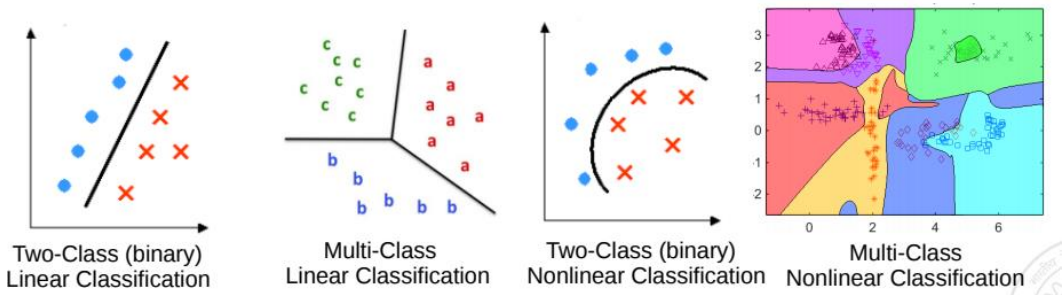
- Basic fact: Inputs in ML problems can often be represented as points or vectors in some vector space
- Doing ML on such data can thus be seen from a geometric view

Regression: A supervised learning problem. Goal is to model the relationship between input (x) and real-valued output (y). This is a kind of a line or curve fitting problem





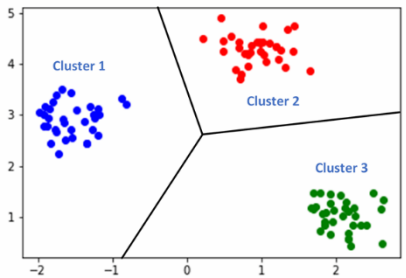
Classification: A supervised learning problem. Goal is to learn a to predict which of the two or more classes an input belongs to. Akin to learning linear/nonlinear separator for the inputs



Pic from: https://learningstatisticswithr.com/book/regression.html, https://maxstat.de/

Geometric Perspective

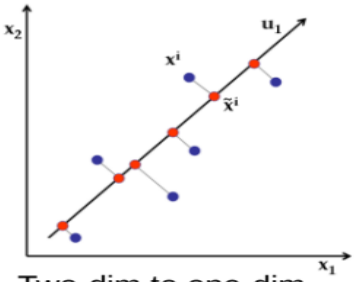
Clustering: An unsupervised learning problem. Goal is to group inputs in a few clusters based on their similarities with each other



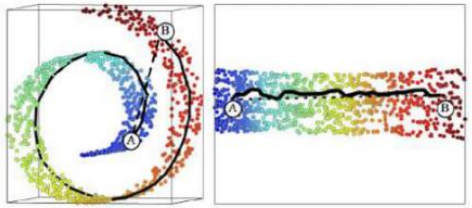
Clustering looks like classification to me. Is there any difference?

Yes. In clustering, we don't know the labels. Goal is to separate them without any labeled "supervision"

Dimensionality Reduction: An unsupervised learning problem. Goal is to compress the size of each input without losing much information present in the data



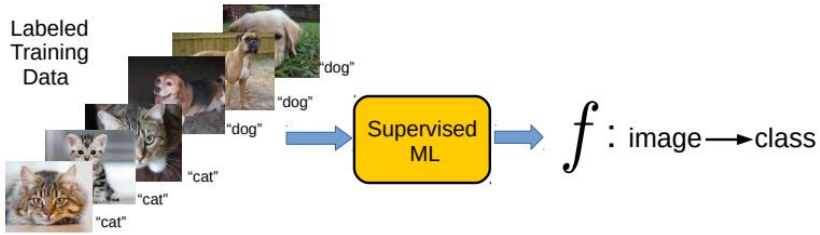
Two-dim to one-dim linear projection



Three-dim to two-dim nonlinear projection (a.k.a. manifold learning)

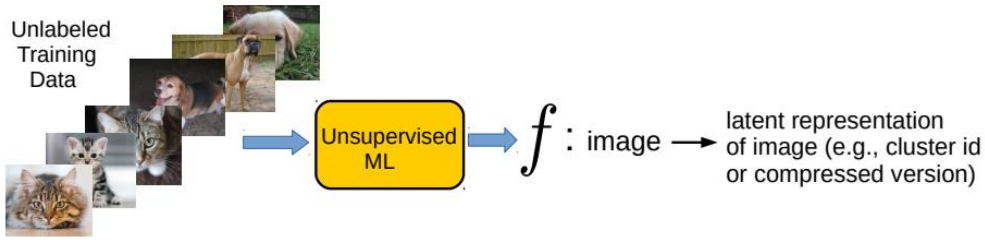
Perspective as function approximation

- Supervised Learning ("predict output given input") can be usually thought of as learning a function f that maps each input to the corresponding output



- Unsupervised Learning ("model/compress inputs") can also be usually thought of as learning a function f that maps each input to a compact representation

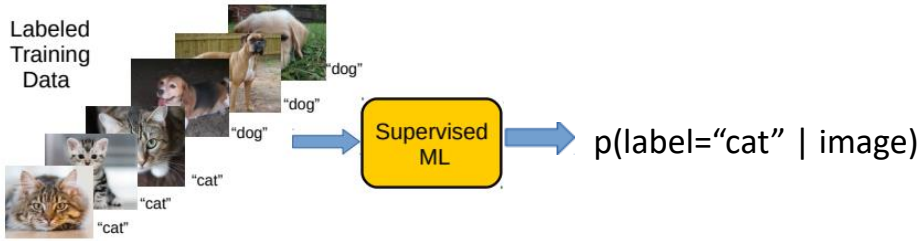
Harder since we don't know the labels in this case



- Reinforcement Learning can also be seen as doing function approximation

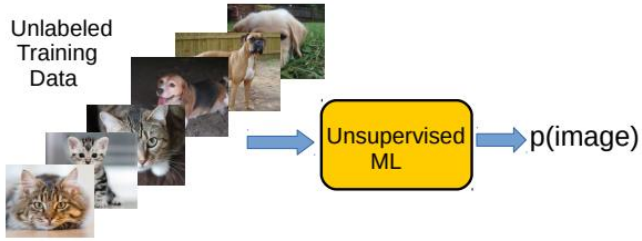
Perspective as probability estimation

- Supervised Learning ("predict output given input") can be thought of as estimating the conditional probability of each possible output given an input



- Unsupervised Learning ("model/compress inputs") can be thought of as estimating the probability density of the inputs

Harder since we don't know the labels in this case



- Reinforcement Learning can also be seen as estimating probability densities

Data and Features

- ML algorithms require a numeric feature representation of the inputs
- Features can be obtained using one of the two approaches
  - Approach 1: Extracting/constructing features manually from raw inputs
  - Approach 2: Learning the features from raw inputs
- Approach 1 is what we will focus on primarily for now
- Approach 2 is what is followed in Deep Learning algorithms (will see in later)
- Approach 1 is not as powerful as Approach 2 but still used widely

Example: Feature Extraction for Text Data

- Consider some text data consisting of the following sentences:
  - John likes to watch movies
  - Mary likes movies too
  - John also likes football

BoW is just one of the many ways of doing feature extraction for text data. Not the most optimal one, and has various flaws (can you think of some?), but often works reasonably well

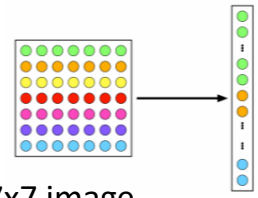
- Want to construct a feature representation for these sentences
- Here is a "bag-of-words" (BoW) feature representation of these sentences

$$
\begin{array}{r}
\\
\text{Sentence 1} \\
\text{Sentence 2} \\
\text{Sentence 3}
\end{array}
\begin{pmatrix}
\text{John} & \text{likes} & \text{to} & \text{watch} & \text{movies} & \text{Mary} & \text{too} & \text{also} & \text{football} \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1
\end{pmatrix}
$$

- Each sentence is now represented as a binary vector (each feature is a binary value, denoting presence or absence of a word). BoW is also called "unigram" representation.

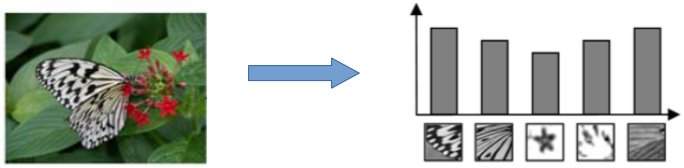
Example: Feature Extraction for Image Data

- A very simple feature extraction approach for image data is flattening



Flattening and histogram based methods destroy the spatial information in the image but often still work reasonably well

- Histogram of visual patterns is another popular feature extr. method for images



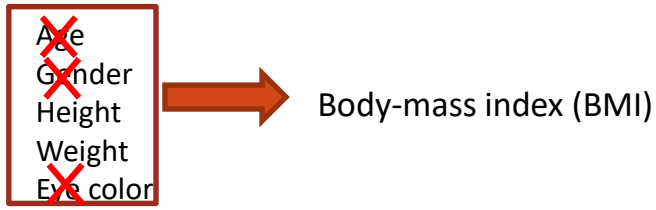
- Many other manual feature extraction techniques developed in computer vision and image processing communities (SIFT, HoG, and others)

Pic credit: cat.uab.cat/Research/object-recognition

Feature Selection

- Not all the extracted features may be relevant for learning the model (some may even confuse the learner)

- Feature selection (a step after feature extraction) can be used to identify the features that matter, and discard the others, for more effective learning

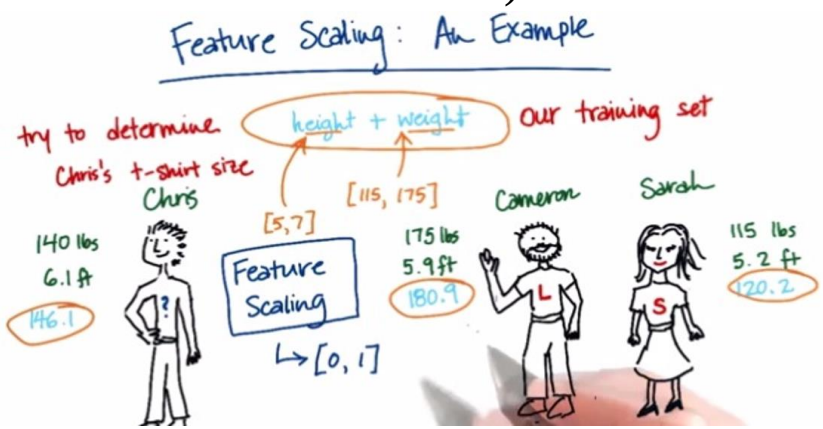


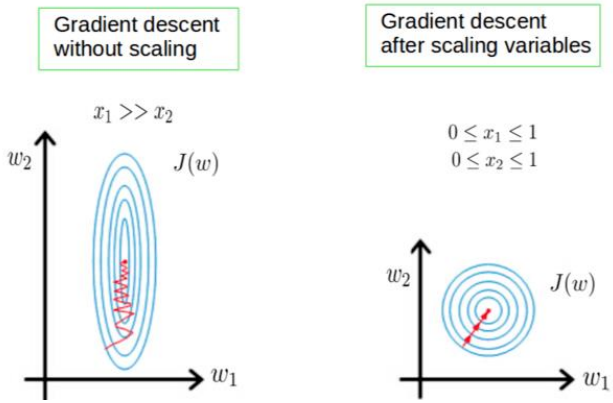
- Many techniques exist – some based on intuition, some based on algorithmic principles.

- More common in supervised learning but can also be done for unsup. learning

Some More Postprocessing: Feature Scaling

- Even after feature selection, the features may not be on the same scale
- This can be problematic when comparing two inputs – features that have larger scales may dominate the result of such comparisons
- Therefore helpful to standardize the features (e.g., by bringing all of them on the same scale such as between 0 to 1)





- Also helpful for stabilizing the optimization techniques used in ML algos

Pic credit: https://becominghuman.ai/demystifying-feature-scaling-baff53e9b3fd, https://stackoverflow.com/

Some Notation/Nomenclature/Convention

- Sup. learning requires training data as N input-output pairs $\{(\mathbf{x_n}, y_n)\}_{n=1}^{N}$
- Unsupervised learning requires training data as N inputs $\{\mathbf{x_n}\}_{n=1}^{N}$

RL and other flavors of ML problems also use similar notation

- Each input $\mathbf{x_n}$ is (usually) a vector containing the values of the features or attributes or covariates that encode properties of the it represents, e.g.,
  - For a 7×7 image: $\mathbf{x_n}$ can be a 49×1 vector of pixel intensitie

Size or length of the input $\mathbf{x_n}$ is commonly known as data/input dimensionality or feature dimensionality

- (In sup. Learning) Each y_n is the output or response or label associated with input $\mathbf{x_n}$ (and its value is known for the training inputs)
  - Output can be a scalar, a vector of numbers, or even an structured object (more on this later)

Performance Measures

- Performance measures are used to evaluate the performance of ML models for a specific problem.

- Classification Problem Performance Measures

- Regression Problem Performance Measures

Classification Problem Performance Measures

- **Confusion Matrix**
  - **True Positive (TP):** Both Actual & Predicted Values are Positive
  - **True Negative (TN):** Both Actual & Predicted Values are Negative
  - **False Positive (FP):** Actual Negative & Predicted Positive
  - **False Negative (FN):** Actual Positive & Predicted Negative

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	**TP** True Positive	**FP** False Positive
	negatives	**FN** False Negative	**TN** True Negative

Classification Problem Performance Measures

- **Accuracy:** It can also be calculated in terms of positives and negatives for binary classification.

- It doesn't grant us much information regarding the distribution of false positives and false negatives.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Shouldn't be used when the dataset is unbalanced.**

```
from sklearn.metrics import accuracy_score
y_true = [1,1,1,0,0,1,0,1]
y_pred = [0,0,0,0,1,1,1,0]
print("Accuracy Score:", accuracy_score(y_true, y_pred))

#Output
Accuracy Score: 0.25
```

Classification Problem Performance Measures

- **Precision:** It is the ratio of True Positives to all the positives predicted by the model.

$$Precision = \frac{TP}{TP + FP}$$

- It is useful for the skewed and unbalanced dataset.
- The more False positives the model predicts, the lower the precision.

```
from sklearn.metrics import precision_score
y_true = [1,1,1,0,0,1,0,1]
y_pred = [0,0,0,0,1,1,1,0]
print("Precision Score:", precision_score(y_true, y_pred))

#Output
Precision Score: 0.333333333333
```

Classification Problem Performance Measures

- **Recall or Sensitivity or True Positive Rate(TPR):** It is the ratio of true positives to all the positives in your dataset.

$$TPR = Recall = \frac{TP}{TP + FN}$$

- It measures the model's ability to detect positive samples.

- **The more false negatives the model predicts, the lower the recall.**

```
from sklearn.metrics import recall_score
y_true = [1,1,1,0,0,1,0,1]
y_pred = [0,0,0,0,1,1,1,0]
print("Recall Score:", recall_score(y_true, y_pred))

#Output
Recall Score: 0.2
```

Classification Problem Performance Measures

- **F1-score or F-measure:** It is a single metric that combines both Precision and Recall.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

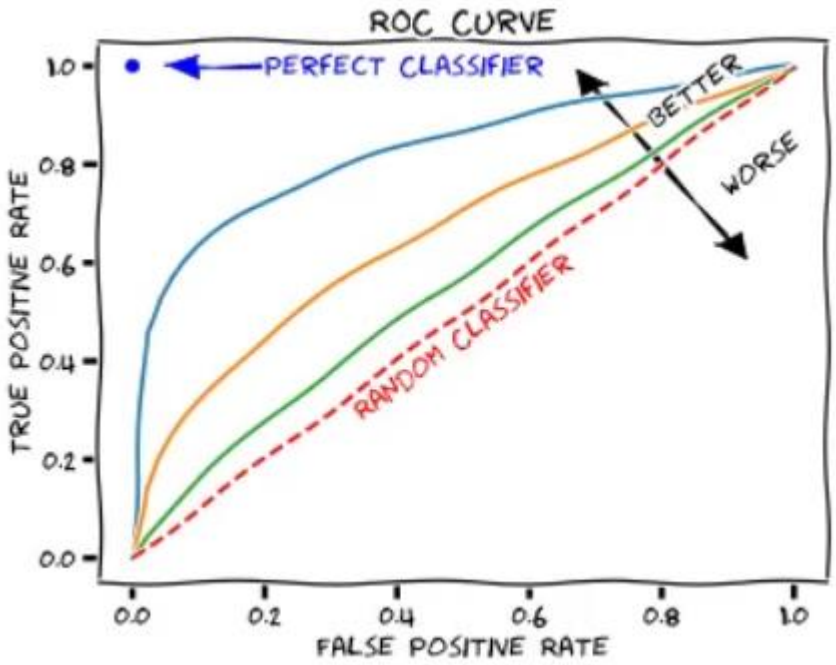
- The higher the F1 score, the better is the performance of our model.

- The range for F1-score is [0,1].

```
from sklearn.metrics import f1_score
y_true = [1,1,1,0,0,1,0,1]
y_pred = [0,0,0,0,1,1,1,0]
print("F1 Score:", f1_score(y_true, y_pred))

#Output
F1 Score: 0.25
```

Classification Problem Performance Measures

- **ROC curve (Receiver Operating Characteristic curve):**
  - It is a way to visualize the tradeoff between the True Positive Rate (TPR) and False Positive Rate(FPR) using different decision thresholds.



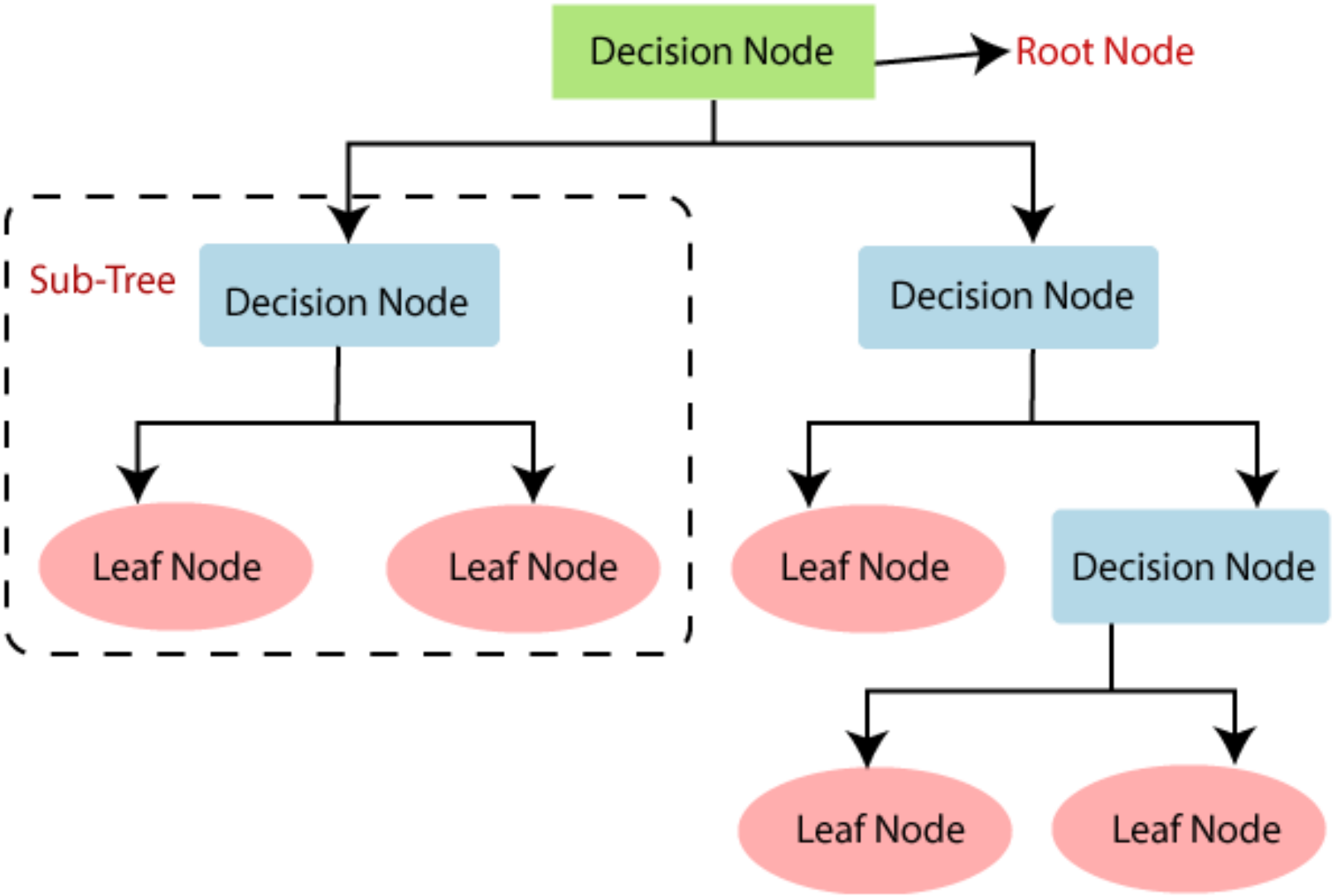

```python
from sklearn.metrics import roc_curve
import numpy as np

y = np.array([1,2,2,4])
score = np.array([0.8,0.4,0.35,0.26])

fpr, tpr, thresholds = roc_curve(y, score, pos_label=2)

print("False Positive Rate: {}\nTrue Positve Rate:
{}\nThresholds: {}".format(fpr, tpr, thresholds))


#Output
False Positive Rate: [0.  0.5 0.5 1. ]
True Positve Rate: [0. 0. 1. 1.]
Thresholds: [1.8  0.8  0.35 0.26]
```

Regression Problem Performance Measures

Error-metric	Accuracy Measure	Formula
Scale-dependent	Mean Absolute Error (MAE) or Mean Absolute Deviation (MAD)	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$
	Geometric Mean Absolute Error (GMAE)	$\left(\prod_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert\right)^{\frac{1}{n}}$
	Mean Square Error (MSE)	$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
	Root Mean Square Error (RMSE)	$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
Percentage	Mean Absolute Percentage Error(MAPE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{y_i}\times 100$
	Symmetric Mean Absolute Percentage Error(SMAPE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{(\lvert y_i\rvert + \lvert\hat{y}_i\rvert)/2}$
Relative	Median Relative Absolute Error (MdRAE)	$median\left(\left\lvert\frac{y_i - \hat{y}_i}{y_i - \hat{y}_i^*}\right\rvert\right)$
	Geometric Mean Relative Absolute Error (GMRAE)	$\left(\prod_{i=1}^{n}\left\lvert\frac{y_i - \hat{y}_i}{y_i - \hat{y}_i^*}\right\rvert\right)^{\frac{1}{n}}$
Scale-free	Mean Absolute Scaled Error (MASE)	$\frac{1}{n}\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{\frac{1}{n-1}\sum_{j=2}^{n}\lvert y_i - y_{i-1}\rvert}$

where y_i and $\hat{y}_i$ denote the ith actual and forecasted value, n denotes the number of observations. **Lower the value better the forecasts.**

Decision Tree



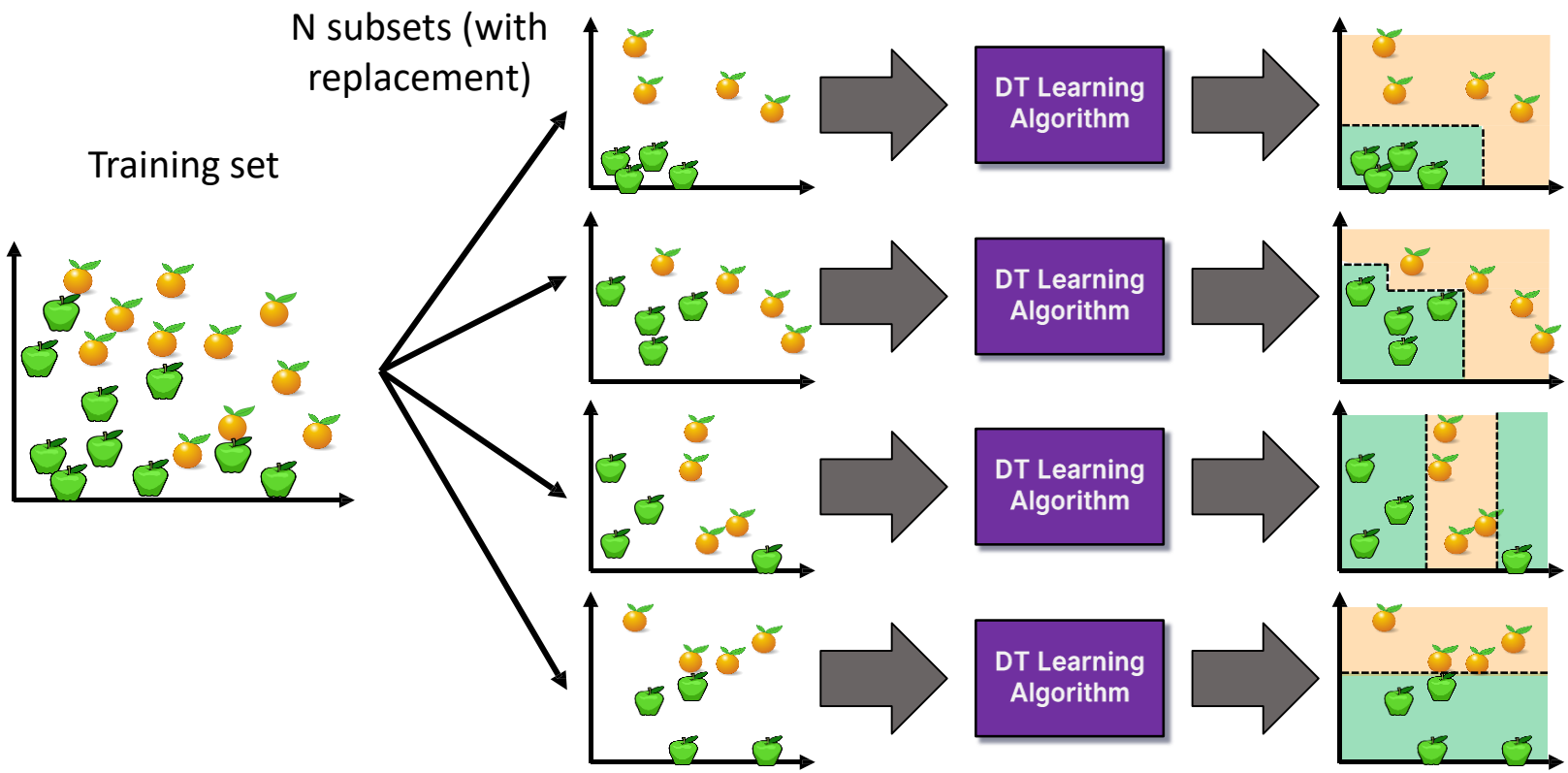
Random Forests

- Random Forests:
  - ➢ Instead of building a single decision tree and use it to make predictions, build many slightly different trees and combine their predictions
- We have a single data set, so how do we obtain slightly different trees?

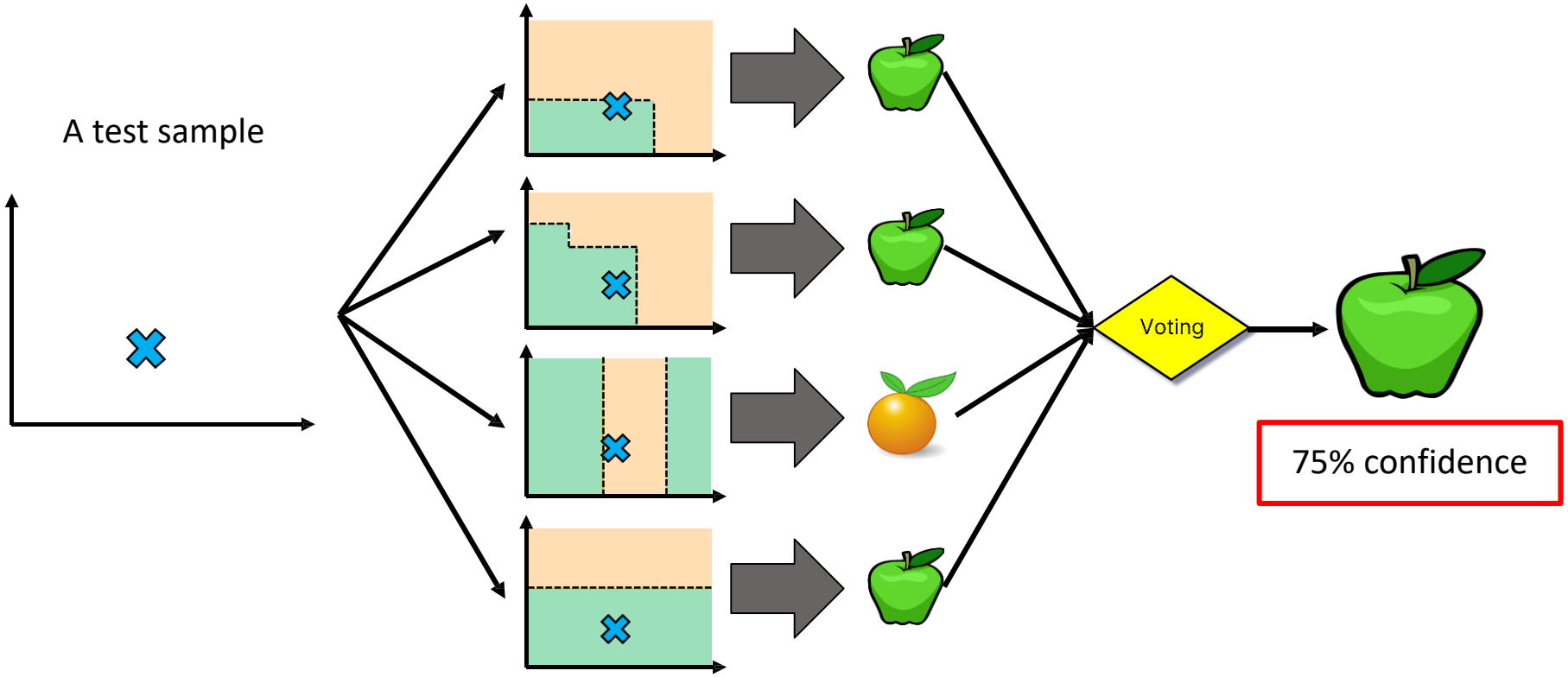
  1. Bagging (**B**ootstrap **Agg**regat**ing**):
  - ➢ Take random subsets of data points from the training set to create N smaller data sets
  - ➢ Fit a decision tree on each subset

  2. Random Subspace Method (also known as Feature Bagging):
  - ➢ Fit N different decision trees by constraining each one to operate on a random subset of features

Bagging at training time



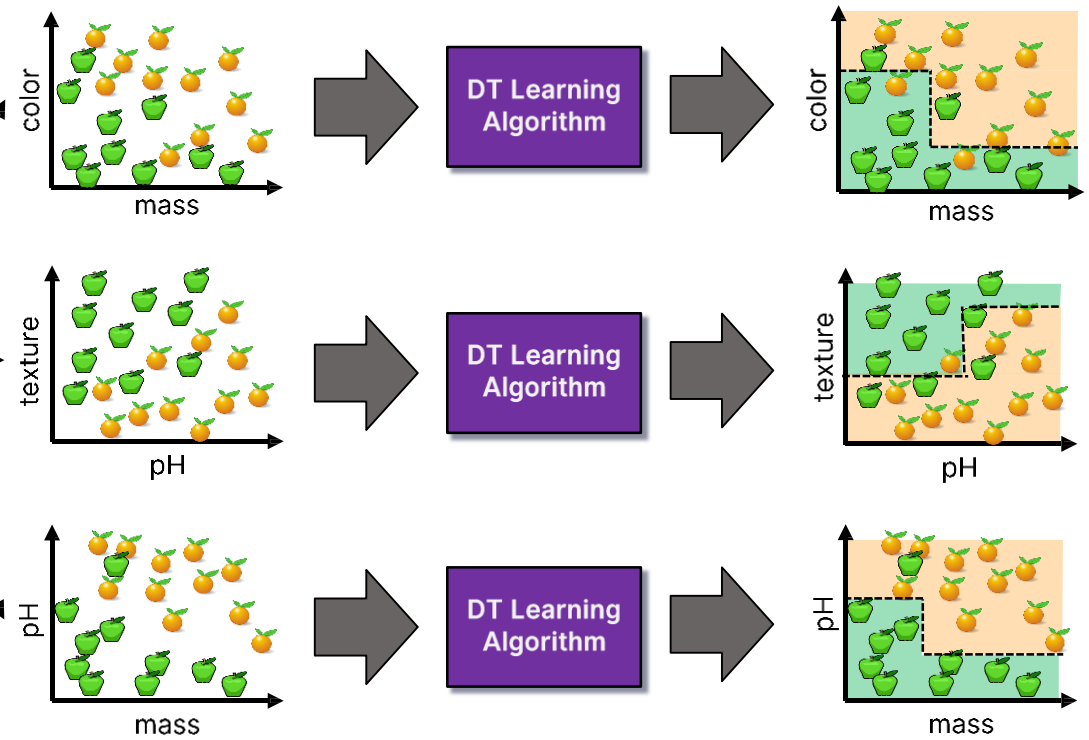
Bagging at inference time



Random Subspace Method at inference time

Training data

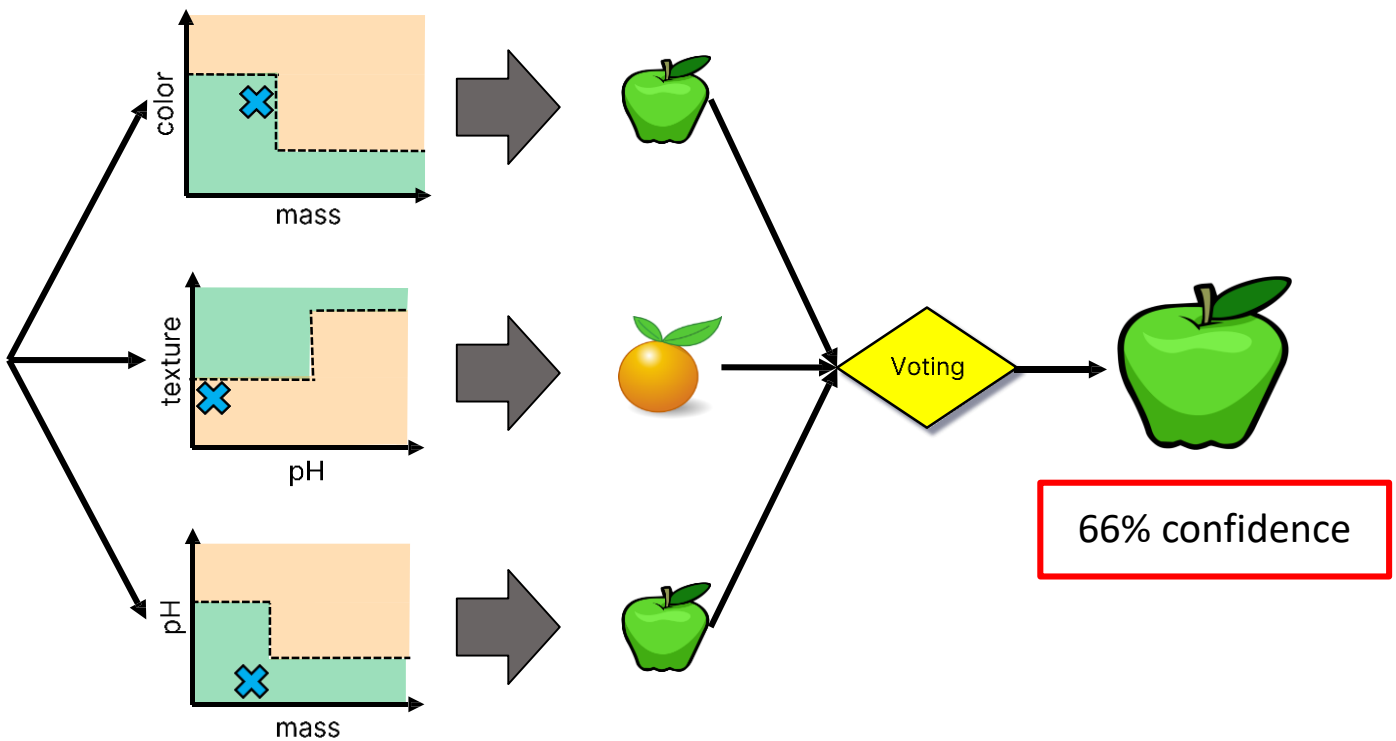
Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	**Apple**
121	Orange	Rough	3.9	**Orange**
85	Red	Smooth	3.3	**Apple**
101	Orange	Smooth	3.7	**Orange**
111	Green	Rough	3.5	**Apple**
...				
117	Red	Rough	3.4	**Orange**



Random Subspace Method at inference time

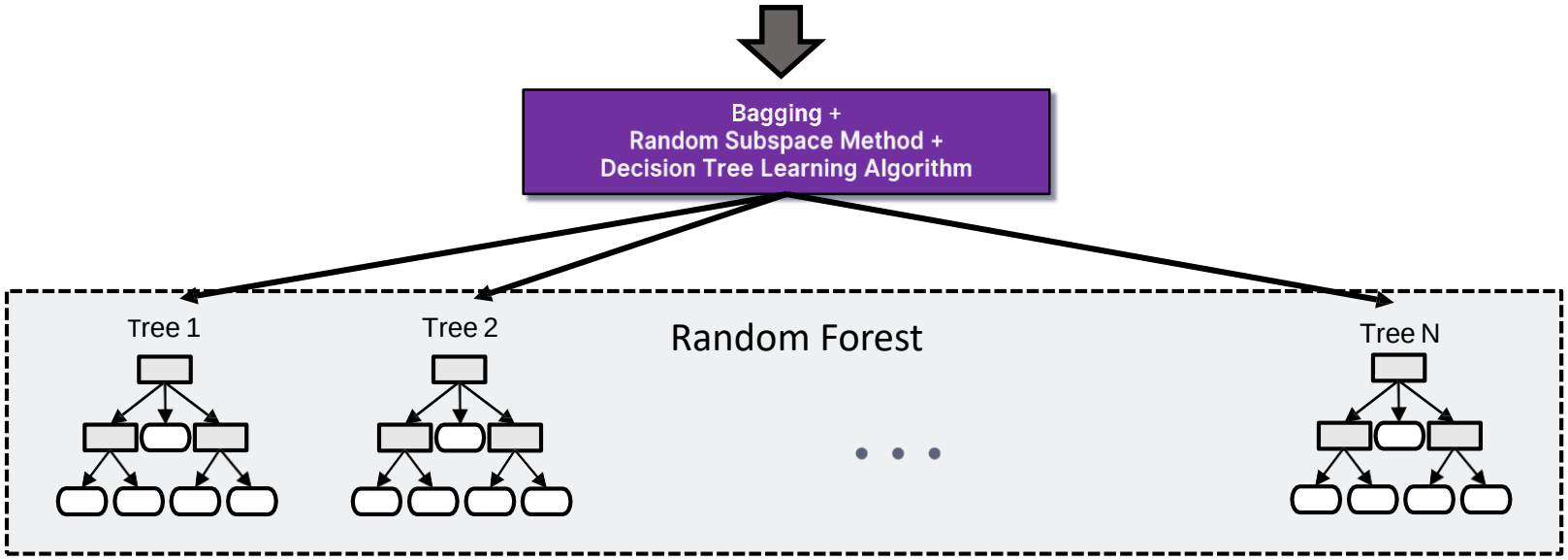
A test sample

87	Red	Smooth	3.1



Random Forests

Mass (g)	Color	Texture	pH	Label
84	Green	Smooth	3.5	**Apple**
121	Orange	Rough	3.9	**Orange**
85	Red	Smooth	3.3	**Apple**
101	Orange	Smooth	3.7	**Orange**
111	Green	Rough	3.5	**Apple**
...				
117	Red	Rough	3.4	**Orange**



History of Random Forests

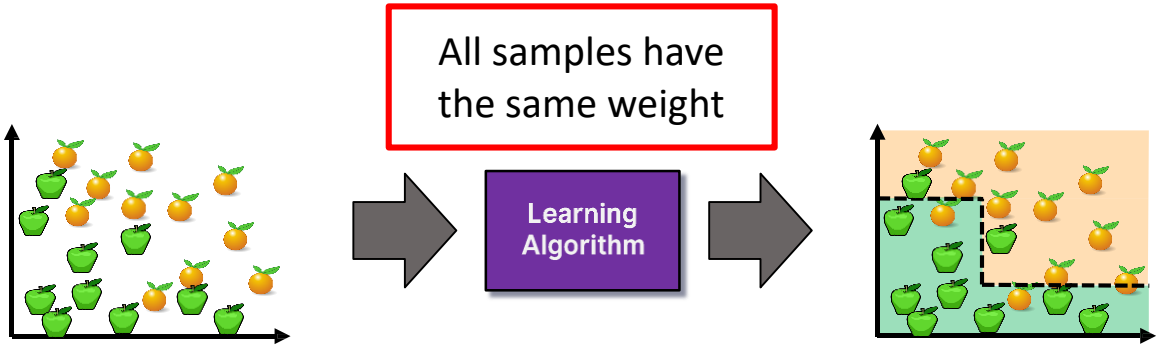
- Introduction of the Random Subspace Method
  - "Random Decision Forests" [Ho, 1995] and "The Random Subspace Method for Constructing Decision Forests" [Ho, 1998]

- Combined the Random Subspace Method with Bagging. Introduce the term Random Forest (a trademark of Leo Breiman and Adele Cutler)
  - "Random Forests" [Breiman, 2001]

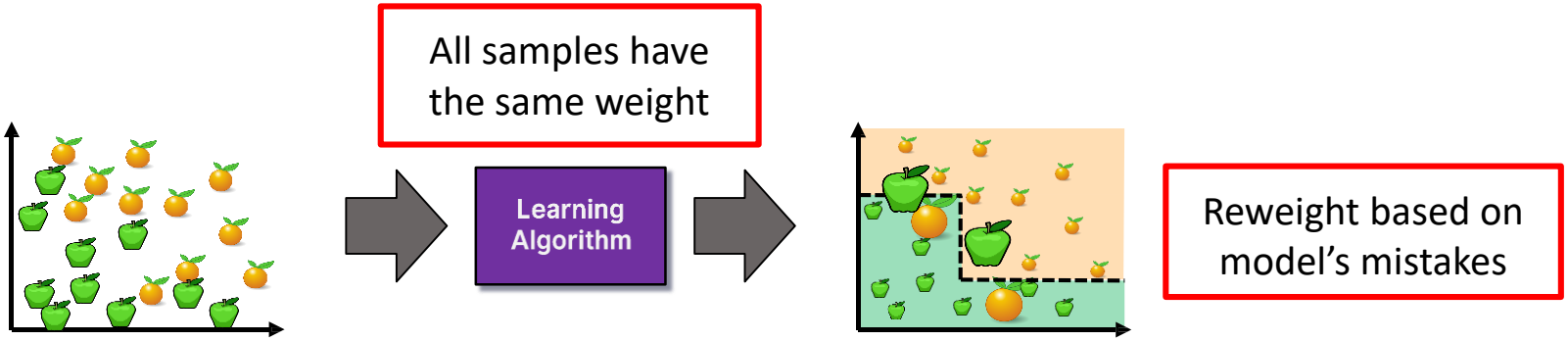
Ensemble Learning

- Ensemble Learning:
  - Method that combines multiple learning algorithms to obtain performance improvements over its components
- **Random Forests** are one of the most common examples of ensemble learning
- Other commonly-used ensemble methods:
  - Bagging: multiple models on random subsets of data samples
  - Random Subspace Method: multiple models on random subsets of features
  - Boosting: train models iteratively, while making the current model focus on the mistakes of the previous ones by increasing the weight of misclassified samples

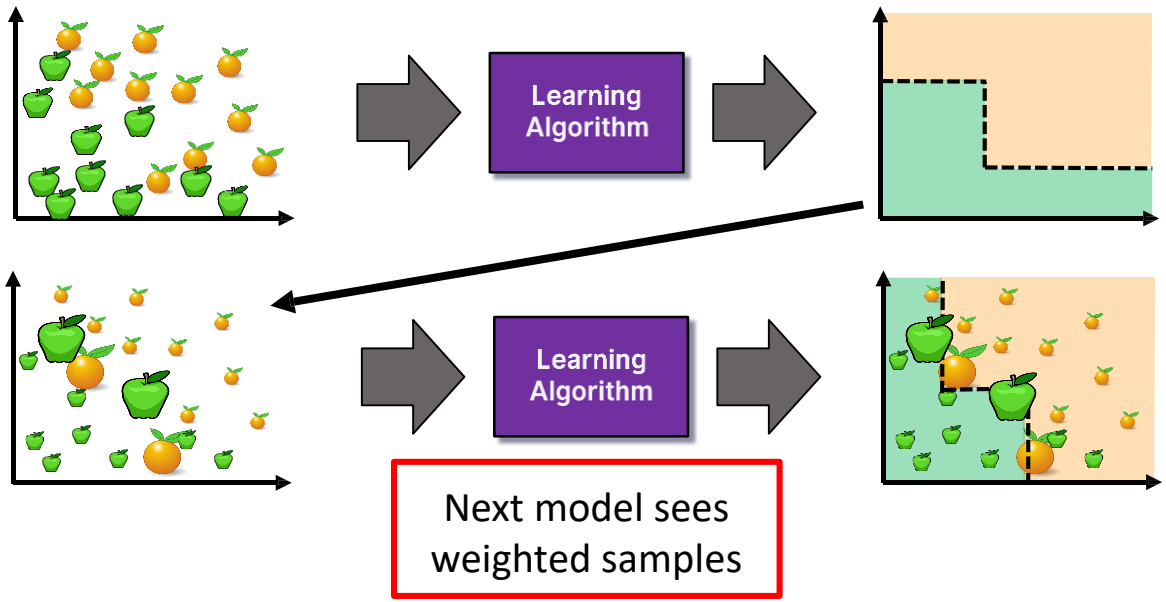
Boosting



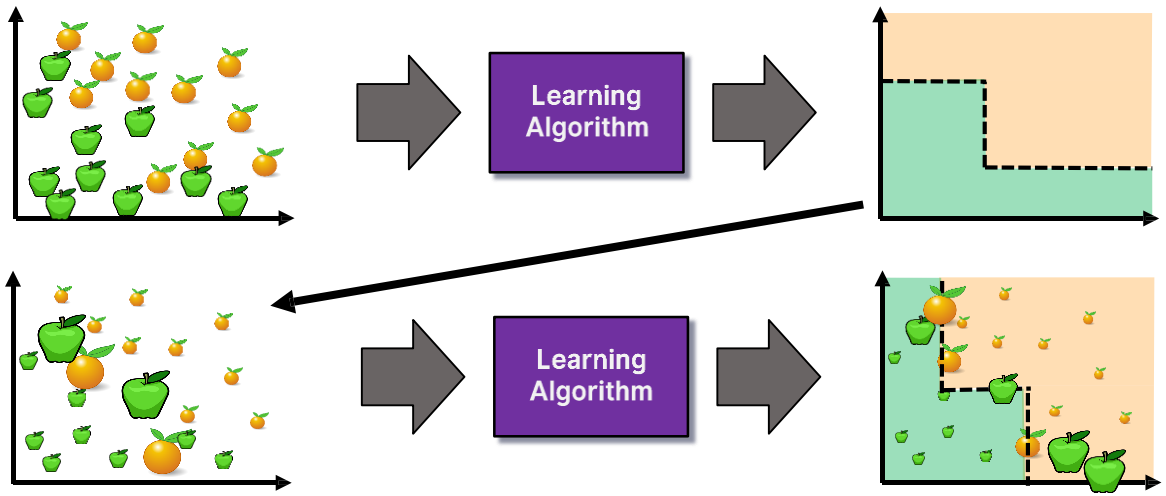
Boosting


All samples have the same weight
Learning Algorithm
Reweight based on model's mistakes

Boosting

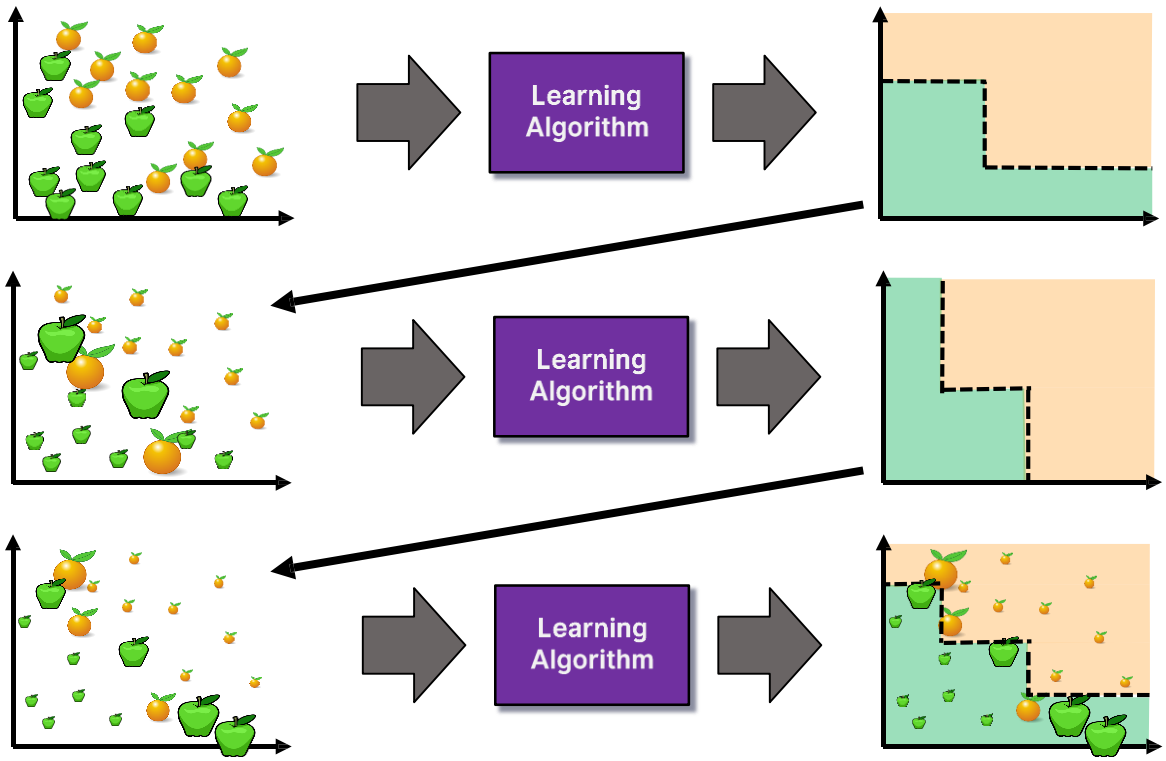

Learning Algorithm
Learning Algorithm
Next model sees weighted samples

Boosting

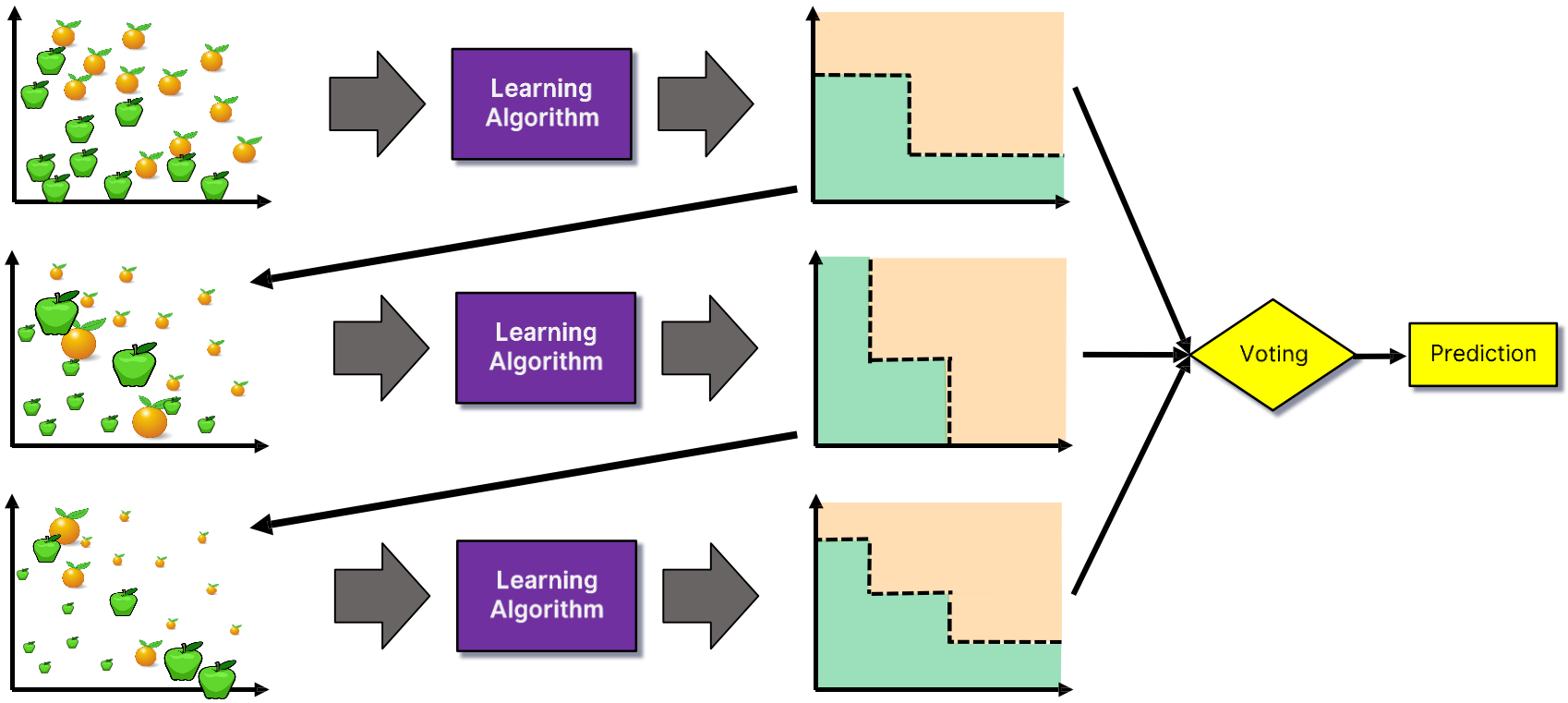


Reweight based on current model's mistakes

Boosting


Learning Algorithm
Learning Algorithm
Learning Algorithm

Boosting


Learning Algorithm
Learning Algorithm
Learning Algorithm
Voting
Prediction

Summary

- Ensemble Learning methods combine multiple learning algorithms to obtain performance improvements over its components
- Commonly-used ensemble methods:
  - Bagging (multiple models on random subsets of data samples)
  - Random Subspace Method (multiple models on random subsets of features)
  - Boosting (train models iteratively, while making the current model focus on the mistakes of the previous ones by increasing the weight of misclassified samples)
- **Random Forests** are an ensemble learning method that employ decision tree learning to build multiple trees through **bagging** and **random subspace method**.
  - They rectify the overfitting problem of decision trees!

Decision Trees and Random Forest (Python)

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

clf = DecisionTreeClassifier(criterion = "entropy",
min_samples_leaf = 3)
# Lots of parameters: criterion = "gini" / "entropy";
#                     max_depth;
#                     min_impurity_split;

clf.fit(X, y) # It can only handle numerical attributes!
# Categorical attributes need to be encoded, see LabelEncoder
and OneHotEncoder

clf.predict([x]) # Predict class for x

clf.feature_importances_ # Importance of each feature
clf.tree_ # The underlying tree object

clf = RandomForestClassifier(n_estimators = 20) # Random Forest
with 20 trees
```

- **Thank You**