Ensemble Learning

Dr. Ahmad Altarawneh

Department of Data Science

Faculty of information technology, Mutah University

Outline

- Ensemble learning
- Ensemble learning approaches
- Bagging
  - Bag of decision trees
  - Random forest
- Boosting
  - Adaboost
- Stacking

What is Ensemble Learning

- Ensemble learning is a machine learning technique that combines the predictions of multiple models to achieve better accuracy and performance.
- These models, also known as weak learners or base learners, may individually be simple, but together they can form a strong predictive system
  - If we want to perform better on a task, we need more workers.
- Consider ensemble learning like asking multiple experts for an opinion. While each expert may have limitations, combining their insights leads to a more accurate decision.

Why Use Ensemble Learning

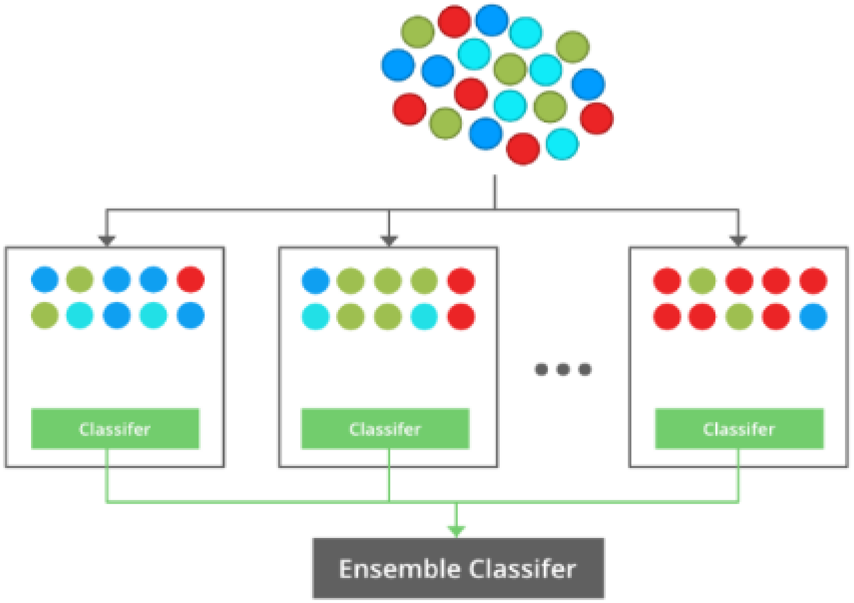
- Reducing Bias and Variance: Individual models often suffer from bias or variance.
  - **bias** means the model makes strong assumptions about the data
  - e.g., A linear regression model used for a highly non-linear relationship will have a high bias because it assumes a linear relationship where none exists (underfitting)
  - variance: high variance means the model makes significant changes in prediction even with small changes in the dataset (overfitting)
  - BOTH LEAD TO POOR GENERALIZATION
- Ensemble learning strikes a balance, creating models that generalize better on unseen data.

Why Use Ensemble Learning

- Error Reduction: By averaging the predictions of multiple models, ensembles help cancel out errors or inconsistencies of individual models, improving reliability.
- Robustness: The variability in weak learners means that if one model performs poorly in a specific scenario, other models can compensate for this weakness.

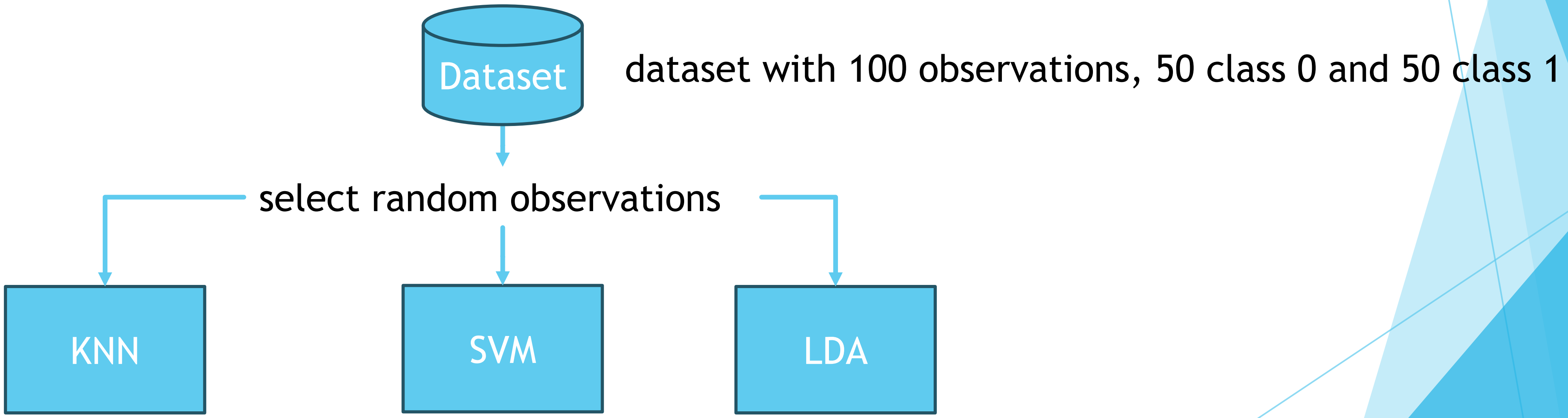
Bagging: Bootstrap aggregating

- Multiple models are trained on random subsets of the data, often with replacement (this is called bootstrapping).
  - Each one of these models is called a **base** or **weak** learner
  - Although, in bagging, they might not be weak
- Each model is trained independently, and their results are averaged (or voted) to make a final prediction



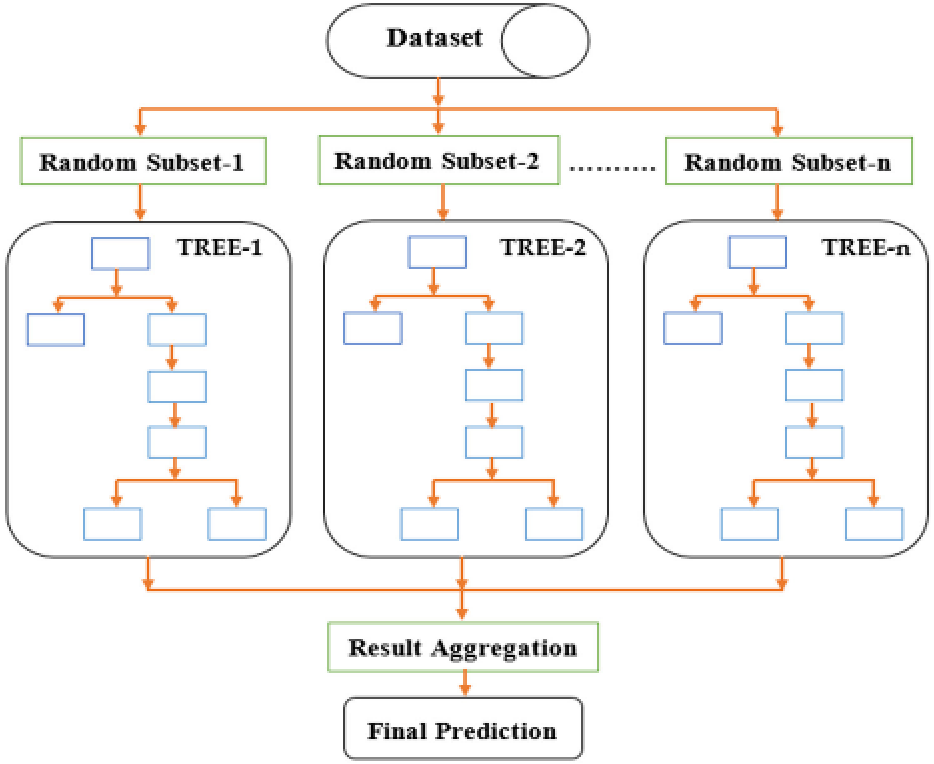
Bagging: Bootstrap aggregating

- Weak learner is a model that performs slightly better than a random guess
  - just over 50%, 51%, for example
- The simplest ensemble learning approach is to aggregate the power of various classifiers like KNN, SVM and LDA (Strong learners)



Bagged trees

- A bag of trees is an ensemble learning method that trains a collection of trees of different subsets (samples) of the dataset
- The samples are selected randomly **with replacement**
- The trees inside this bag are trained using heuristics like *information gain* or *Gini-index*



Code example
Classification

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

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

base_tree = DecisionTreeClassifier()
bagged_trees = BaggingClassifier(base_estimator=base_tree, n_estimators=100,
random_state=42)

bagged_trees.fit(X_train, y_train)

y_pred = bagged_trees.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

Code example
Regression

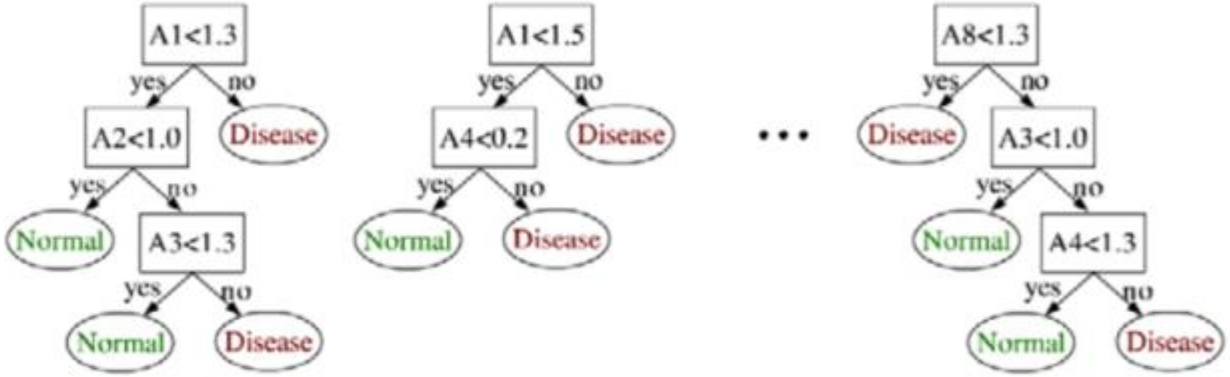
```
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

data = fetch_california_housing()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


base_tree = DecisionTreeRegressor()  bagged_trees_reg =
BaggingRegressor(base_estimator=base_tree, n_estimators=100, random_state=42)

bagged_trees_reg.fit(X_train, y_train)

y_pred = bagged_trees_reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

Bagged trees Cont.

- Bagged trees, reduces variance by training trees on different subsets of the data
- However, in the bagged trees, there is a high chance that different trees may make similar splits
  - as they have access to all features
- Also, the computational cost is high, especially when the dataset is huge with high dimensionality

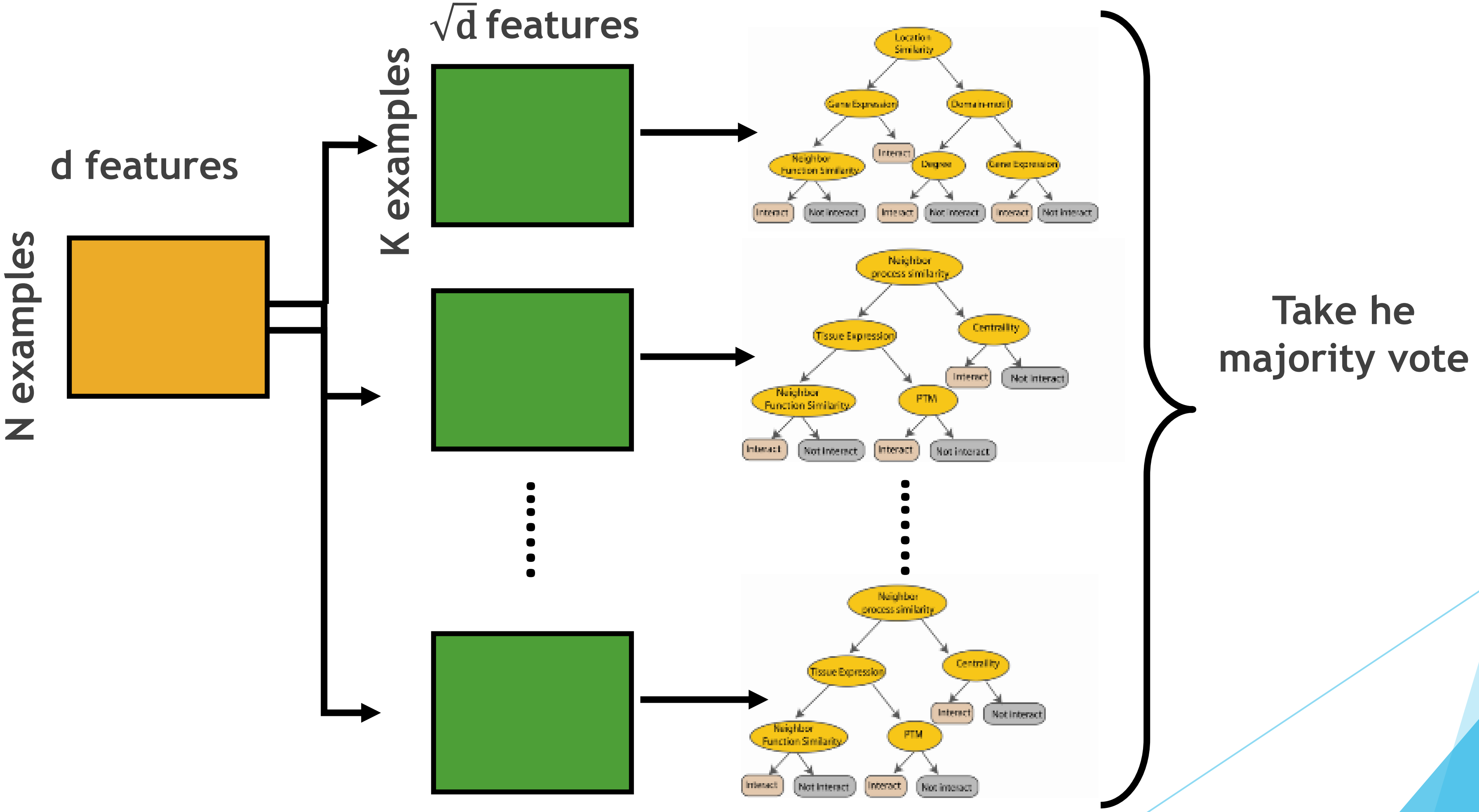
Random Forest

- Random Forest RF, is similar to the bagged trees. However, it adds another layer of randomness
- The other layer of randomness is at the feature level.
- It takes $\sqrt{d}$ features for each subset, $\frac{d}{3}$ for regression problems
- This layer of randomness increases the diversity of the model and improves generalization to new unseen data



Each tree uses a random selection of $m \approx \sqrt{d}$ features $\{A_{i_j}\}_{j=1}^{m}$ chosen from *all* features $A_1, A_2, \ldots, A_d$

Random Forest Cont.



Code Example
Random Forest

Classification

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

RF = RandomForestClassifier(n_estimators=100,
random_state=42)
class_weight='balanced'
RF.fit(X_train, y_train)

y_pred = RF.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
```

Regression

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


data = fetch_california_housing()
X, y = data.data, data.target


X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

rf_regressor = RandomForestRegressor(n_estimators=100,
random_state=42)

rf_regressor.fit(X_train, y_train)

y_pred = rf_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.4f}")
```

Important notes

- RF and bagged trees choose the samples (bootstrapped subsets) randomly with replacement
  - ***Some samples might appear multiple times and some might not appear in the subset***
- It might happen by chance that the majority of examples are drawn from 1 class, which might affect the performance
  - maybe all samples come from 1 class
  - The high number of estimators (weak learners mitigates the bad effect
- to further ensure this will not affect the performance, use balance subsets
  - This applies to classification tasks

```
RF = RandomForestClassifier(class_weight='balanced' n_estimators=100, random_state=42)
```

  - In bagged trees the class_weight is added to the estimator it self

```
base_tree = DecisionTreeClassifier(class_weight='balanced', random_state=42)

bagging_classifier = BaggingClassifier(base_estimator=base_tree, n_estimators=100, random_state=42)
```

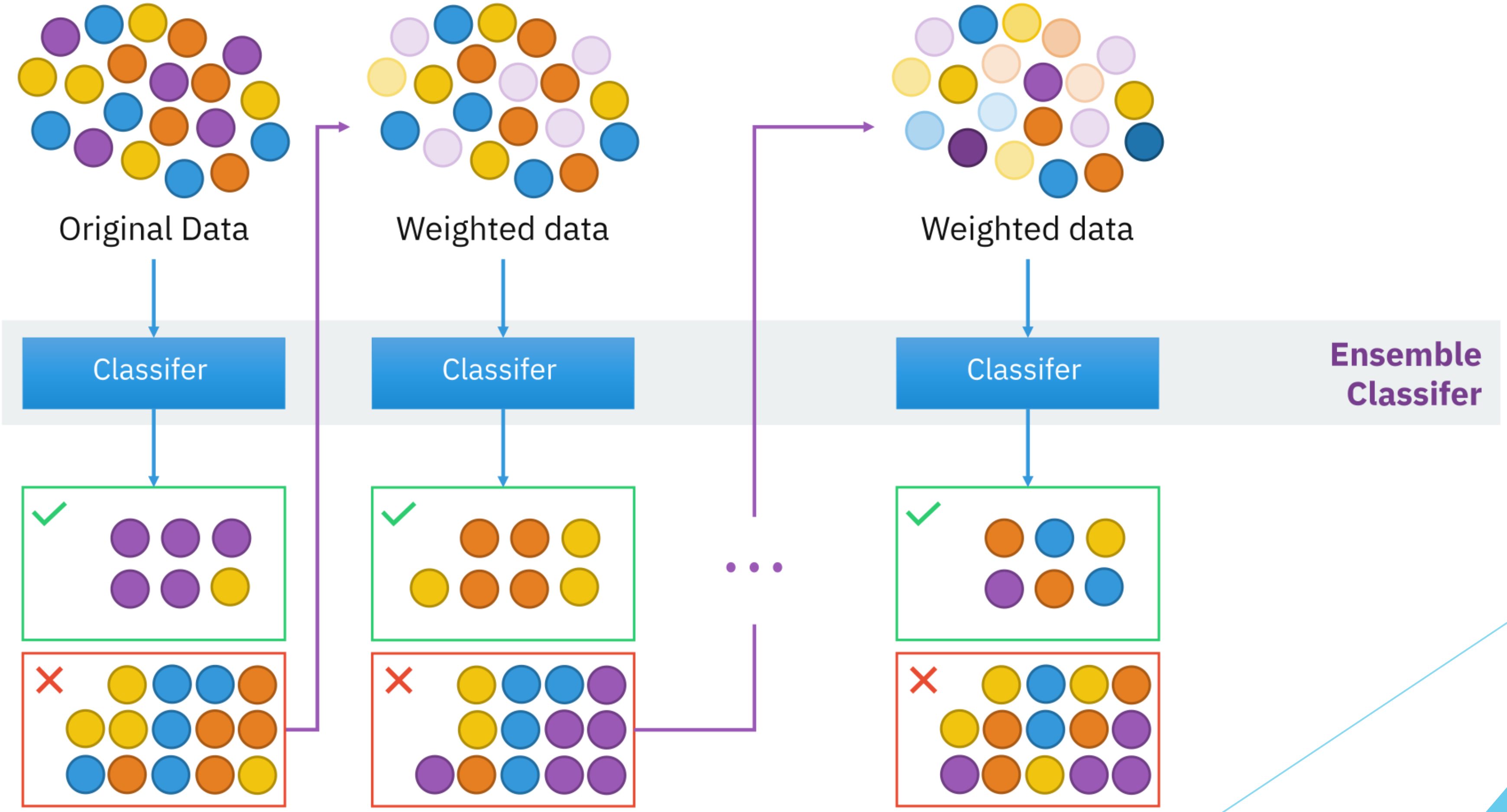
Important notes Cont.

- max_samples is another parameter to play with using these methods
- its default value is between 0-1, 1 means all original datasets will be used for the bootstrap sampling, and the size of each subset will be the same as the original dataset
  - Although they are not the same
  - the subset might include some samples multiple times, while other samples do not appear
- If max_samples is assigned an integer value, e.g., 100, Each base learner will be trained on exactly 100 samples, regardless of the total number of samples in the original dataset

Boosting

- **Boosting** is a powerful ensemble learning technique aimed at improving model performance by **combining several weak learners** to form a strong learner.
- Boosting focuses on **sequentially correcting errors** made by previous models.
- Pmethods in this approach:
  - AdaBoost
  - Gradient Boosting
  - XGboost

Boosting Cont.



AdaBoost

- **AdaBoost** (Adaptive Boosting) is a widely used boosting algorithm that combines multiple weak learners to form a strong classifier.
- **How It Works:**

1. **Weak Learners:** Typically uses **decision stumps** (single-level decision trees) as weak learners.
2. **Sequential Learning:** Each weak learner is trained on the dataset, focusing on **misclassified instances** from the previous model.
3. **Weighting:** After each round, AdaBoost:
    1. **Increases weights** of misclassified points, so the next learner pays more attention to them.
    2. **Decreases weights** of correctly classified points, reducing their influence.
4. **Final Prediction:** A weighted vote of all weak learners' predictions forms the final output.

Adaboost
Procedure

Steps of Adaboost learning

1- Create w vector that determine each sample to be considered in the classification

$$\text{Set } w_i = \frac{1}{n} \text{ for each sample } i, \text{ where } n \text{ is the total number of training samples}$$

2- Create a weak classifier C_t, In AdaBoost the weak learner is a 1-depth decision tree (decision stamp)

3- Classify the data using this stamp

4- Calculate the error

The error is given by $\boldsymbol{\epsilon_t = \sum_{i=1}^{n} w_i \,.\, I(\bar{y}_i \neq y_i)}$ *=> I(.) = 1 if* $\bar{y}_i \neq y_i$ *otherwise 0*

5- Calculate $\boldsymbol{\alpha_t}$ *(the amount of say)*

$$\boldsymbol{\alpha_t = \frac{1}{2} log(\frac{(1-\epsilon_t)}{\epsilon_t})}$$

6- Update the weights

$w_i = w_i . e^{-\alpha_t}$ *if the sample is correcly classified, and* $w_i = w_i . e^{\alpha_t}$ otherwise

7- Normalize the weights so that they sum to 1

$$w_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$$

8- Create the new dataset for the next stamp (resampling based on the weights)

Adaboost Example

- Assume you have the following data

#	X1	X2	X3	Class	
1	83	0.3	73	+	
2	91	0.06	7	+	
3	98	0.41	42	+	
4	95	0.16	29	+	
5	89	0.71	99	+	
6	73	0.81	37	-	
7	58	0.66	82	-	
8	32	0.65	36	-	
9	13	0.11	91	-	
10	82	0.28	91	-	

Adaboost
Initialize the weights

- Assign an initial weights for each sample in the dataset
  - $\frac{1}{10} = 0.1$

#	X1	X2	X3	Class	weight
1	83	0.3	73	+	0.1
2	91	0.06	7	+	0.1
3	98	0.41	42	+	0.1
4	95	0.16	29	+	0.1
5	89	0.71	99	+	0.1
6	73	0.81	37	-	0.1
7	58	0.66	82	-	0.1
8	32	0.65	36	-	0.1
9	13	0.11	91	-	0.1
10	82	0.28	91	-	0.1

Adaboost
Predict

- Now use the decision stamp to classify the data and find the predictions

#	X1	X2	X3	Class	weight	P
1	83	0.3	73	+	0.1	+
2	91	0.06	7	+	0.1	+
3	98	0.41	42	+	0.1	-
4	95	0.16	29	+	0.1	-
5	89	0.71	99	+	0.1	-
6	73	0.81	37	-	0.1	-
7	58	0.66	82	-	0.1	-
8	32	0.65	36	-	0.1	-
9	13	0.11	91	-	0.1	-
10	82	0.28	91	-	0.1	-

Adaboost
Calculate the error

- Calculate the errors

#	X1	X2	X3	Class	weight	P	E
1	83	0.3	73	+	0.1	+	0
2	91	0.06	7	+	0.1	+	0
3	98	0.41	42	+	0.1	-	1
4	95	0.16	29	+	0.1	-	1
5	89	0.71	99	+	0.1	-	1
6	73	0.81	37	-	0.1	-	0
7	58	0.66	82	-	0.1	-	0
8	32	0.65	36	-	0.1	-	0
9	13	0.11	91	-	0.1	-	0
10	82	0.28	91	-	0.1	-	0

Adaboost
Calculate the error

The error is given by $\epsilon_t = \sum_{i=1}^{n} w_i \, . I(\overline{y}_i \neq y_i)$ => $I(.) = 1$ *if* $\overline{y}_i \neq y_i$ *otherwise 0*

#	X1	X2	X3	Class	weight	P	E
1	83	0.3	73	+	0.1	+	0
2	91	0.06	7	+	0.1	+	0
3	98	0.41	42	+	0.1	-	1
4	95	0.16	29	+	0.1	-	1
5	89	0.71	99	+	0.1	-	1
6	73	0.81	37	-	0.1	-	0
7	58	0.66	82	-	0.1	-	0
8	32	0.65	36	-	0.1	-	0
9	13	0.11	91	-	0.1	-	0
10	82	0.28	91	-	0.1	-	0
Error = 1*0.1 + 1*0.1 + 1*0.1							0.3

Adaboost
Amount to say

- *Calculate* $\boldsymbol{\alpha_t}$ *(the amount of say)* $\boldsymbol{\alpha_t = \frac{1}{2} log(\frac{(1-\epsilon_t)}{\epsilon_t})}$

#	X1	X2	X3	Class	weight	P	E
1	83	0.3	73	+	0.1	+	0
2	91	0.06	7	+	0.1	+	0
3	98	0.41	42	+	0.1	-	1
4	95	0.16	29	+	0.1	-	1
5	89	0.71	99	+	0.1	-	1
6	73	0.81	37	-	0.1	-	0
7	58	0.66	82	-	0.1	-	0
8	32	0.65	36	-	0.1	-	0
9	13	0.11	91	-	0.1	-	0
10	82	0.28	91	-	0.1	-	0
Error = 1*0.1 + 1*0.1 + 1*0.1							0.3
α =							0.42

To avoid log 0, which is undefined, we add a very small number called epsilon (EPS). E.g., EPS=0.0001

Adaboost
Update the weights

- $w_i = w_i . e^{-\alpha_t}$ *if the sample is correctly classified, and* $w_i = w_i . e^{\alpha_t}$ otherwise

$\boldsymbol{\alpha_t}$	0.42
Correct w	0.65
Incorrect w	1.53

Adaboost

Update the weights

#	X1	X2	X3	Class	weight	P	E	New w
1	83	0.3	73	+	0.1	+	0	0.065
2	91	0.06	7	+	0.1	+	0	0.065
3	98	0.41	42	+	0.1	-	1	0.153
4	95	0.16	29	+	0.1	-	1	0.153
5	89	0.71	99	+	0.1	-	1	0.153
6	73	0.81	37	-	0.1	-	0	0.065
7	58	0.66	82	-	0.1	-	0	0.065
8	32	0.65	36	-	0.1	-	0	0.065
9	13	0.11	91	-	0.1	-	0	0.065
10	82	0.28	91	-	0.1	-	0	0.065

Adaboost
Normalize the weights

- *Normalize the weights so that they sum to 1* $w_i = \frac{w_i}{\sum_{i=1}^{n} w_i}$

#	X1	X2	X3	Class	weight	P	E	New w	Nor. w
1	83	0.3	73	+	0.1	+	0	0.065	0.071
2	91	0.06	7	+	0.1	+	0	0.065	0.071
3	98	0.41	42	+	0.1	-	1	0.153	0.167
4	95	0.16	29	+	0.1	-	1	0.153	0.167
5	89	0.71	99	+	0.1	-	1	0.153	0.167
6	73	0.81	37	-	0.1	-	0	0.065	0.071
7	58	0.66	82	-	0.1	-	0	0.065	0.071
8	32	0.65	36	-	0.1	-	0	0.065	0.071
9	13	0.11	91	-	0.1	-	0	0.065	0.071
10	82	0.28	91	-	0.1	-	0	0.065	0.071

Adaboost
Create the new dataset

- Create a new resampled dataset.
- In the new data the wrongly classified samples are more likely to appear
- You can do this by performing a cumulative sum and picking uniformly distribution random values in the range [0-1]

Adaboost
Create the new dataset

Cumulative distribution

#	X1	X2	X3	Class	weight	P	E	New w	Nor. w	Low	Up
1	83	0.3	73	+	0.1	+	0	0.065	0.071	0	0.071
2	91	0.06	7	+	0.1	+	0	0.065	0.071	0.071	0.142
3	98	0.41	42	+	0.1	-	1	0.153	0.167	0.142	0.307
4	95	0.16	29	+	0.1	-	1	0.153	0.167	0.307	0.467
5	89	0.71	99	+	0.1	-	1	0.153	0.167	0.467	0.634
6	73	0.81	37	-	0.1	-	0	0.065	0.071	0.634	0.705
7	58	0.66	82	-	0.1	-	0	0.065	0.071	0.705	0.776
8	32	0.65	36	-	0.1	-	0	0.065	0.071	0.776	0.847
9	13	0.11	91	-	0.1	-	0	0.065	0.071	0.847	0.918
10	82	0.28	91	-	0.1	-	0	0.065	0.071	0.918	0.99

Adaboost
Create the new dataset

- Now pick Uniformly distributed random number [0-1]
- As the wrong examples have bigger interval, they are more likely to present in the new data

Adaboost

Create the new dataset

UDRNumber								
0.5	0.2	0.3	0.2	0.1	0.9	0.47	0.1	0.95

#	X1	X2	X3	Class	weight	P	E	New w	Nor. w	Low	Up
1	83	0.3	73	+	0.1	+	0	0.065	0.071	0	0.071
2	91	0.06	7	+	0.1	+	0	0.065	0.071	0.071	0.142
3	98	0.41	42	+	0.1	-	1	0.153	0.167	0.142	0.307
4	95	0.16	29	+	0.1	-	1	0.153	0.167	0.307	0.467
5	89	0.71	99	+	0.1	-	1	0.153	0.167	0.467	0.634
6	73	0.81	37	-	0.1	-	0	0.065	0.071	0.634	0.705
7	58	0.66	82	-	0.1	-	0	0.065	0.071	0.705	0.776
8	32	0.65	36	-	0.1	-	0	0.065	0.071	0.776	0.847
9	13	0.11	91	-	0.1	-	0	0.065	0.071	0.847	0.918
10	82	0.28	91	-	0.1	-	0	0.065	0.071	0.918	0.99

Adaboost

Create the new dataset

UDRNumber	0.5	0.2	0.3	0.2	0.1	0.9	0.47	0.1	0.95

#	X1	X2	X3	Class	weight	P	E	New w	Nor. w	Low	Up
1	83	0.3	73	+	0.1	+	0	0.065	0.071	0	0.071
2	91	0.06	7	+	0.1	+	0	0.065	0.071	0.071	0.142
3	98	0.41	42	+	0.1	-	1	0.153	0.167	0.142	0.307
4	95	0.16	29	+	0.1	-	1	0.153	0.167	0.307	0.467
5	89	0.71	99	+	0.1	-	1	0.153	0.167	0.467	0.634
6	73	0.81	37	-	0.1	-	0	0.065	0.071	0.634	0.705
7	58	0.66	82	-	0.1	-	0	0.065	0.071	0.705	0.776
8	32	0.65	36	-	0.1	-	0	0.065	0.071	0.776	0.847
9	13	0.11	91	-	0.1	-	0	0.065	0.071	0.847	0.918
10	82	0.28	91	-	0.1	-	0	0.065	0.071	0.918	0.99

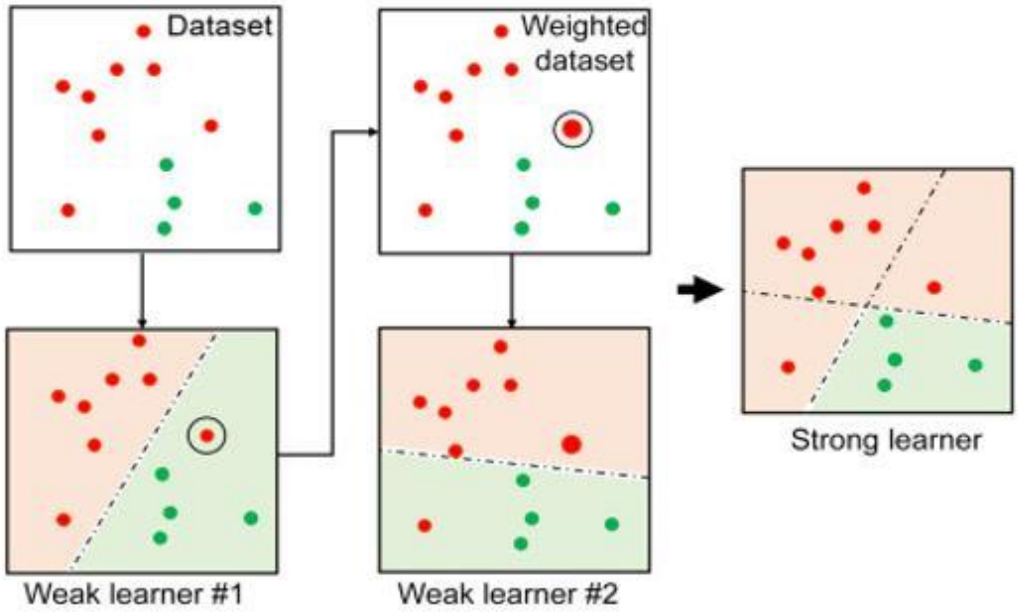
Add this sample, 5, to the new resampled dataset

Adaboost
resampled dataset

#	X1	X2	X3	Class
5	89	0.71	99	+
3	98	0.41	42	+
3	98	0.41	42	+
4	95	0.16	29	+
5	89	0.71	99	+
2	91	0.06	7	+
9	13	0.11	91	-
5	89	0.71	99	+
2	91	0.06	7	+
10	82	0.28	91	-

Repeat and prediction

- Repeat the previous steps for the number of defined classifiers.
- The final prediction is achieved using the following formula.
  - $prediction = Sign(\sum \alpha_t . y_t)$
  - y_t is the prediction of the t weak classifier (estimator)



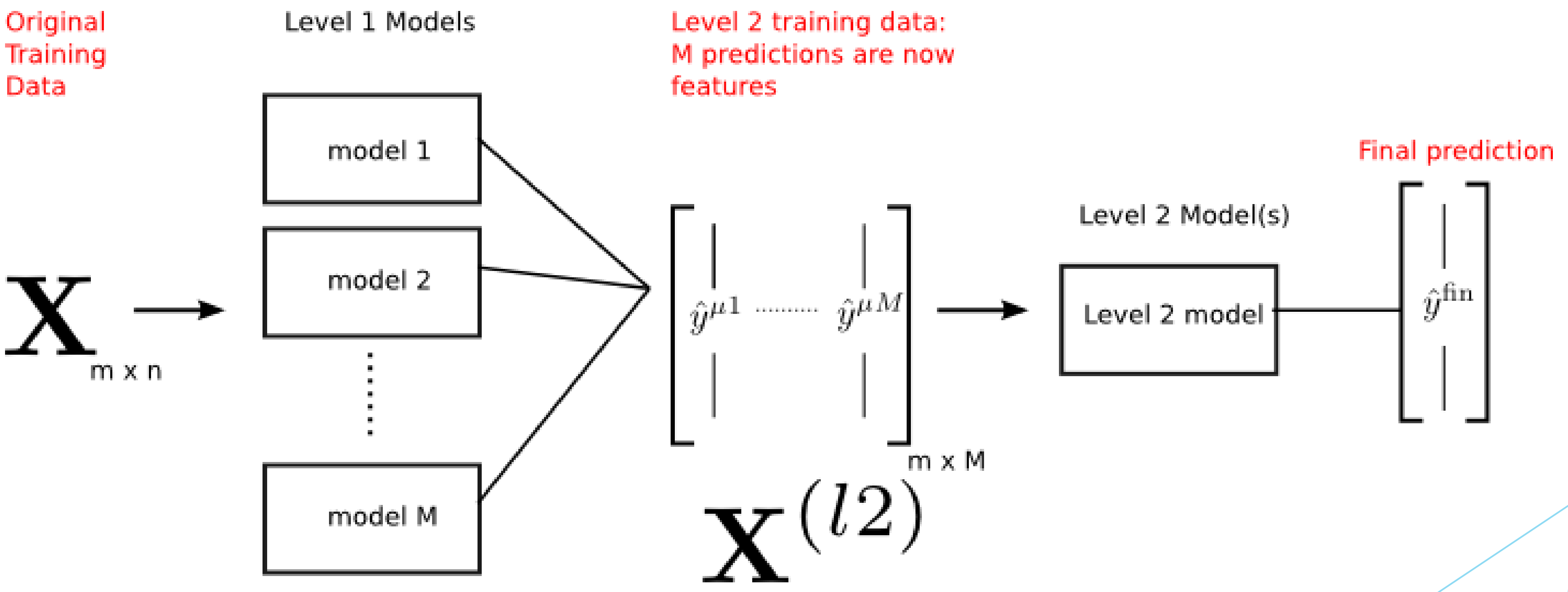
Stacking

- **Stacking** is an ensemble learning technique that combines multiple machine learning models to achieve better predictive performance than any individual model
- How it works
  - Multiple Base Learners: Several models (e.g., decision trees, logistic regression, SVM, etc.) are trained on the same dataset.
  - Meta-Learner (Blender): The predictions from base learners are then passed to a meta-learner, trained to make the final prediction.
  - Diverse Models: Unlike other ensembles like bagging or boosting, stacking typically uses models of different types to capture different patterns in the data

Stacking
Cont.

- After the base classifiers make their predictions, their outputs become the meta-learner's input features.
- Meta-Learner Input: Each base classifier outputs either:
  1. Class predictions (for classification tasks).
  2. Probability estimates (e.g., the probability of belonging to each class).
  3. Regression outputs (for regression tasks).
- These outputs form a new dataset.
- Each row in this dataset consists of the predictions made by the base classifiers for a particular sample.

Stacking
Cont.



Notes

- Ensemble learning provides powerful learning abilities by combining the outputs of multiple models
  - This combination allows the model to divide complex tasks between several learners, each focusing on different aspects of the data.
- Bagging trains independent classifiers in parallel because each base learner is trained on a different subset of the data, often sampled with replacement.
  - This independence makes it easy to parallelize bagging across multiple machines or nodes in a cluster, enabling faster training.
- Boosting, on the other hand, cannot be parallelized easily because it trains sequential classifiers.
  - Each subsequent model is built to correct the errors made by the previous models, making it inherently sequential.
  - Limits the ability to parallelize the process effectively.
- Ensemble learning increases computational requirements
  - This is because you are training multiple models, which can increase memory and processing time requirements.
  - However, this tradeoff is often justified by the significant improvement in performance for complex problems where a single model might underperform.
  - Inevitable for some tasks.