Ensemble Learning

CSCI-P556 Applied Machine Learning
Lecture 21

D.S. Williamson

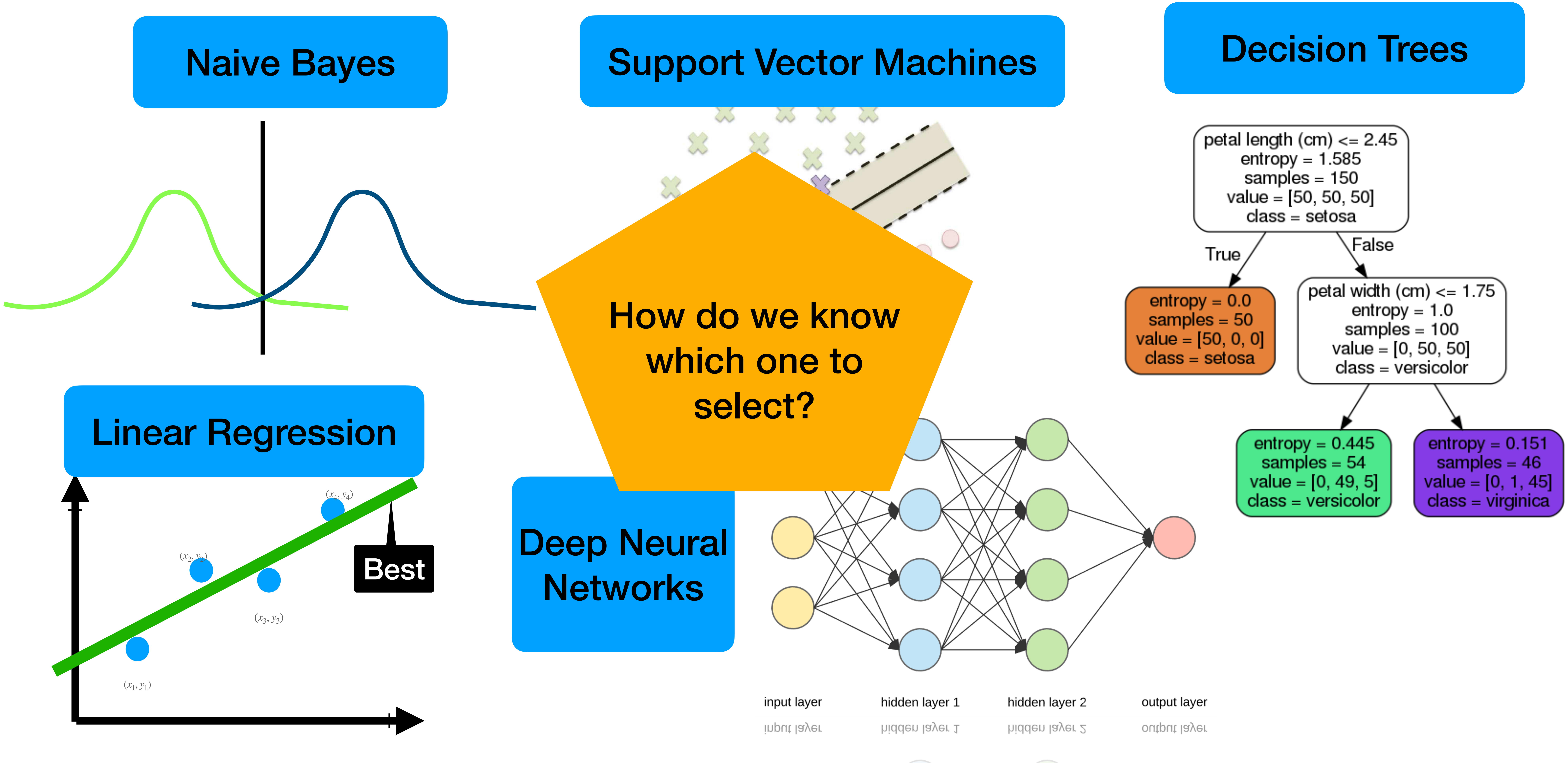
Agenda and Learning Outcomes

Today's Topics

- **Topics**:
  - Quiz #2 review
  - Ensemble Learning
    - Bagging
    - Random Forests
    - Boosting

Recap: Learning Algorithms

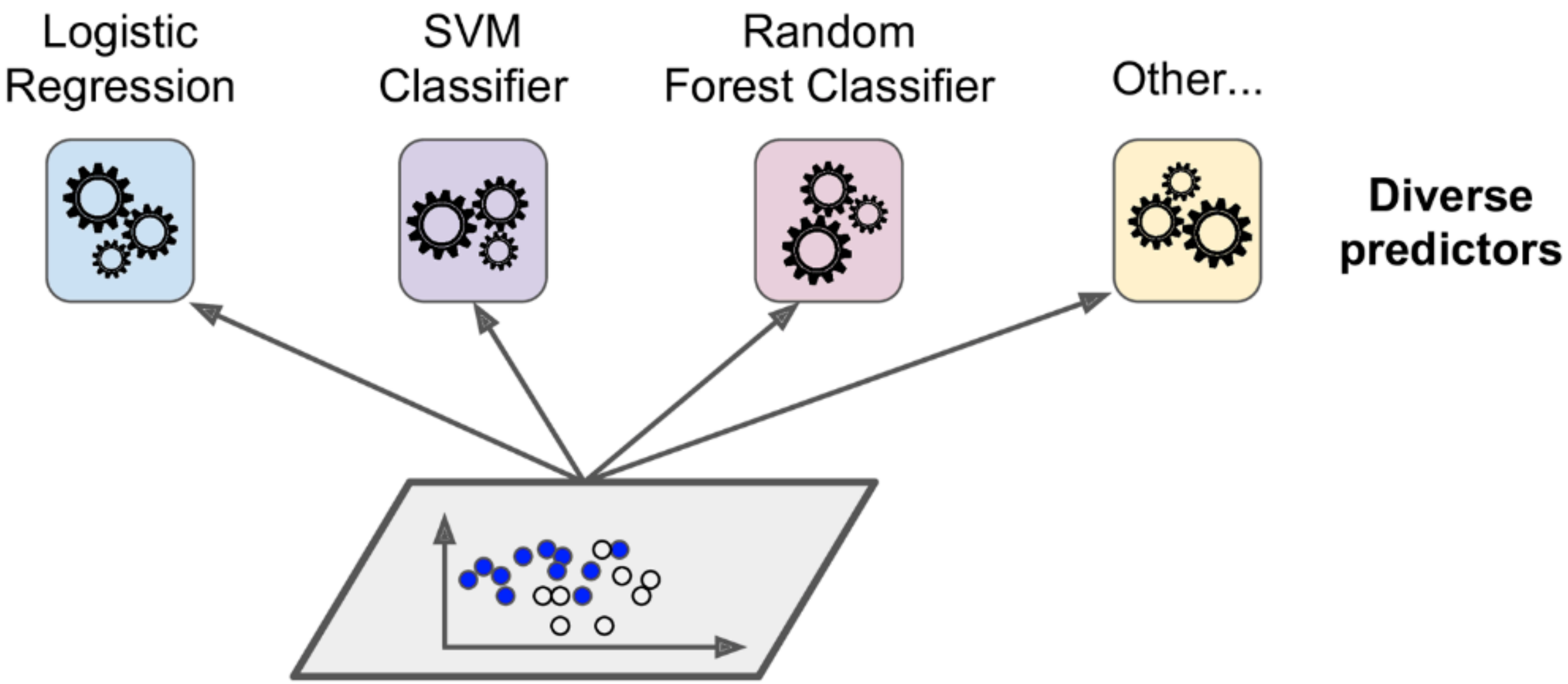
We've discussed several learning algorithms



Which Learning Algorithm Do We Choose?

Ensemble Learning

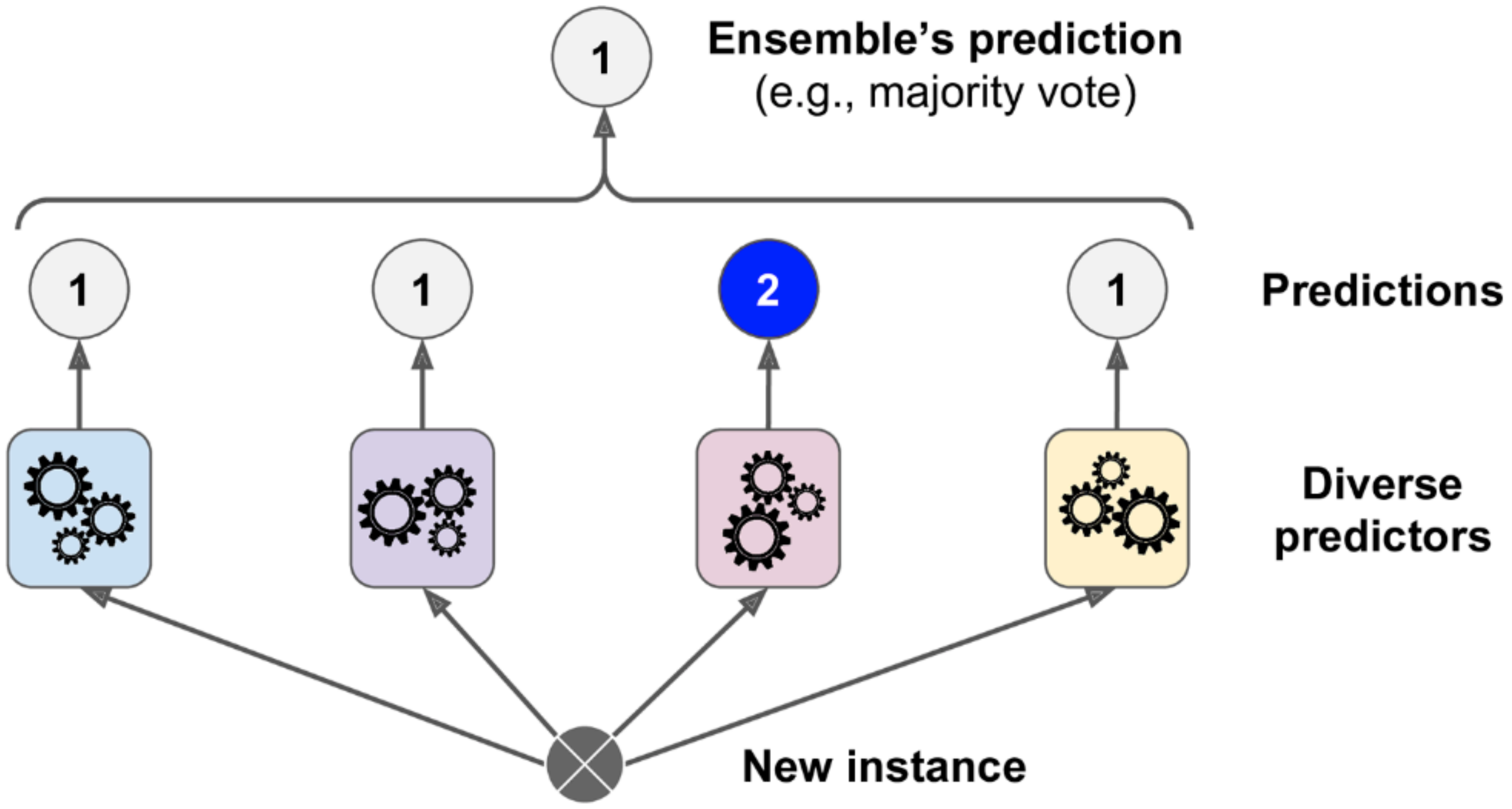
- This is not always clear and obvious
- Hence, we select ALL (or many of them)!
- The idea of considering multiple learning algorithms is known as ***Ensemble Learning***.
    - It is based on the idea that more opinions is better than one.



Ensemble Learning: Voting or Averaging

The Aggregate Experience

- **The simplest ensemble learning strategy involves**:
  - Training multiple algorithms using the same data
  - Generating predictions for each of the algorithms
  - *Aggregate the predictions* of each algorithm to *form a single prediction*
    - **Classification**: *hard-voting* approach (majority wins). *Soft voting* (pick class with highest probability, averaged across)
    - **Regression**: average the predictions



Ensemble Learning for Classification

A Python Example: Hard Voting

- **Consider three learning algorithms**: logistic regression, Random Forest (more on this later), and Support Vector Machine
- Use these to train and test an ensemble voting classifier.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression()
rnd_clf = RandomForestClassifier()
svm_clf = SVC()

voting_clf = VotingClassifier(
        estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
        voting='hard'
    )
voting_clf.fit(X_train, y_train)
```

Individual Accuracy

- Logistic Regression: 86.4%
- Random Forest: 87.2%
- Support Vector Machine: 88.8%

Ensemble Accuracy: 89.6%

Ensemble Learning for Classification

A Python Example: Soft Voting

- **Consider three learning algorithms**: logistic regression, Random Forest (more on this later), and Support Vector Machine

- Use these to train and test an ensemble voting classifier.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

log_clf = LogisticRegression(solver="liblinear", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=10, random_state=42)
svm_clf = SVC(gamma="auto", random_state=42,probability=True)

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf), ('rf', rnd_clf), ('svc', svm_clf)],
    voting='soft')
```

Individual Accuracy

- Logistic Regression: 86.4%
- Random Forest: 87.2%
- Support Vector Machine: 88.8%

Ensemble Accuracy: 91.2%

Why Ensembles?

- When combining multiple ***independent*** and ***diverse*** decisions each of which is at least more accurate than random guessing, <u>***random errors cancel each other out, correct decisions are reinforced***</u>.

- Human ensembles are demonstrably better
  - How many jelly beans in the jar?: Individual estimates vs. group average.
  - Who Wants to be a Millionaire: Expert friend vs. audience vote.

- **Theoretically:** They serve to reduce <u>bias</u> and/or <u>variance</u>

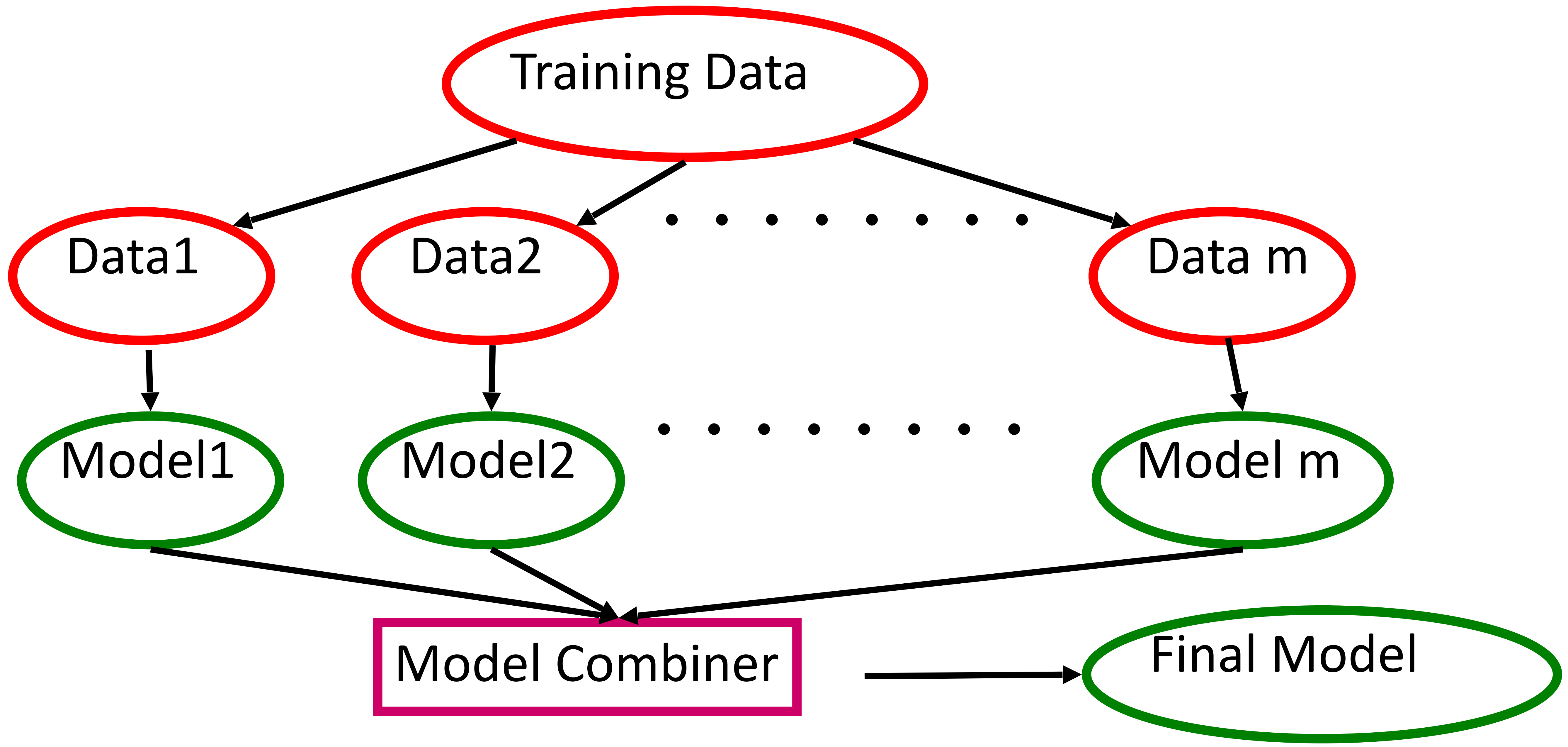
Learning Ensembles

Two approaches

- Perform learning using ***<u>different training data</u>*** or ***<u>different learning algorithms</u>***.
- Combine decisions of multiple definitions, e.g. using weighted voting.

- **When the data varies**, these ensemble learners is either based on **(1) bagging (bootstrap aggregation) or (2) pasting**

- **Key Feature:** They take a single learning algorithm and generate multiple variations (ensembles)



Bagging and Pasting

Only a Subtle Difference

- **Key Feature**: They take a **single** learning algorithm and generate multiple variations (ensembles)
- With ***Bagging***, training samples are randomly selected for each variation, but ***sampling is performed with replacement.*** Hence, there may be data replicates within/across the variations.
- ***Pasting*** performs sampling without replacement.

Bagging Algorithm

Given training set S, bagging works as follows

- Given a training set *S* and *m* different predictors, bagging works as follows:

    1. Randomly select *n* ***bootstrap samples*** from *S* with replacement, *m* different times

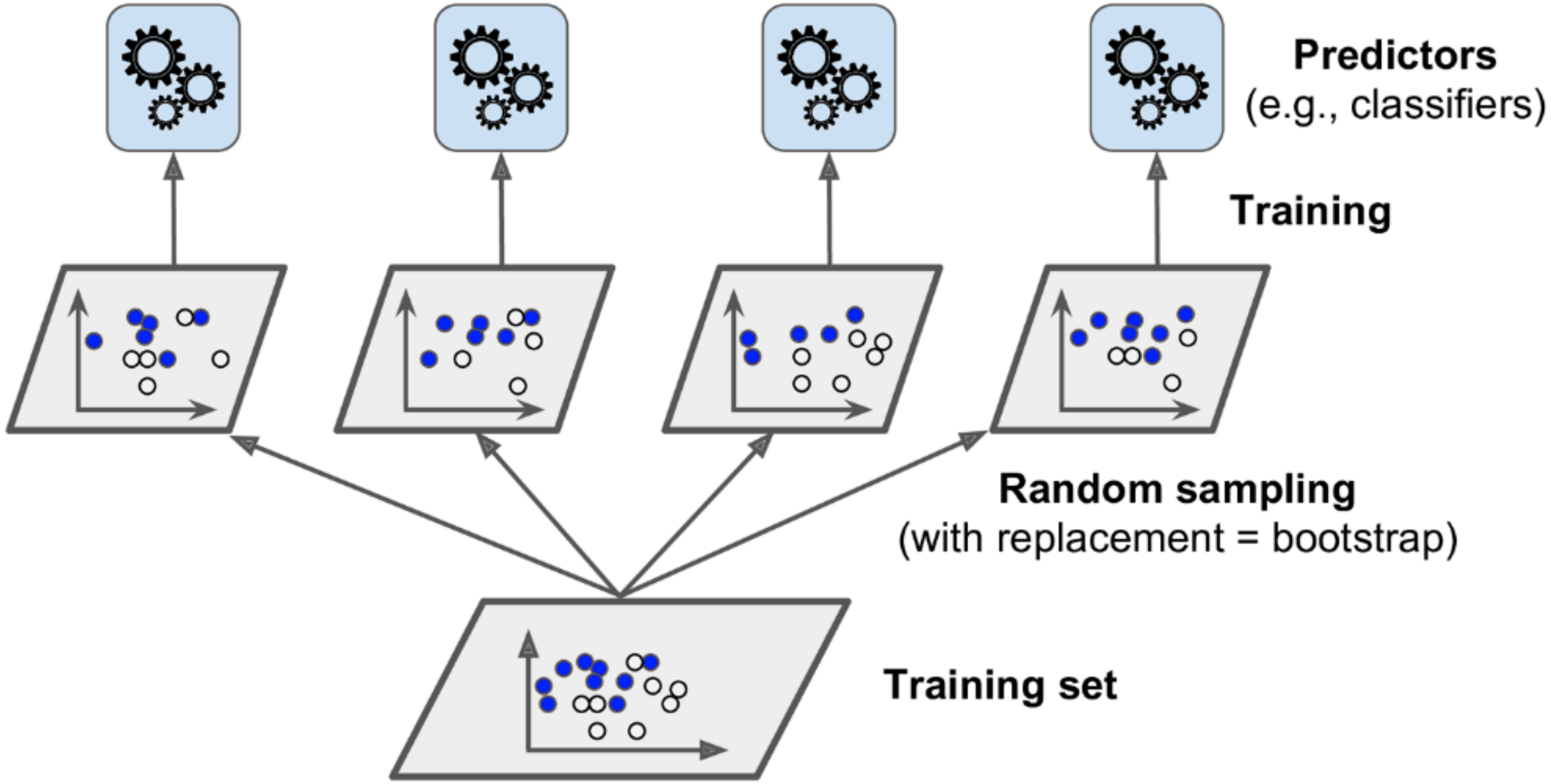
    2. Train the predictor using its corresponding data

    3. Aggregate the results from the *m* resulting models (averaging or hard vote)

- On average each bootstrap sample will contain 63.2% of the unique training examples, the rest (36.8%) are replicas.

A Depiction of Bagging

- With ***Bagging***, training samples are randomly selected for each variation, but ***sampling is performed with replacement.*** Hence, there may or may not be data overlap across the variations



Benefits of Bagging

- Training and predictions may be done in parallel
- ***Individual predictors have a higher bias*** (e.g. underfit the data), but the ***aggregation reduces the variance*** (e.g. less overfitting).
  - Models that fit the data poorly have high bias
  - Models that can fit the data very well have low bias but high variance
- Bagging tends to outperform Pasting

Bagging Classifier in Python

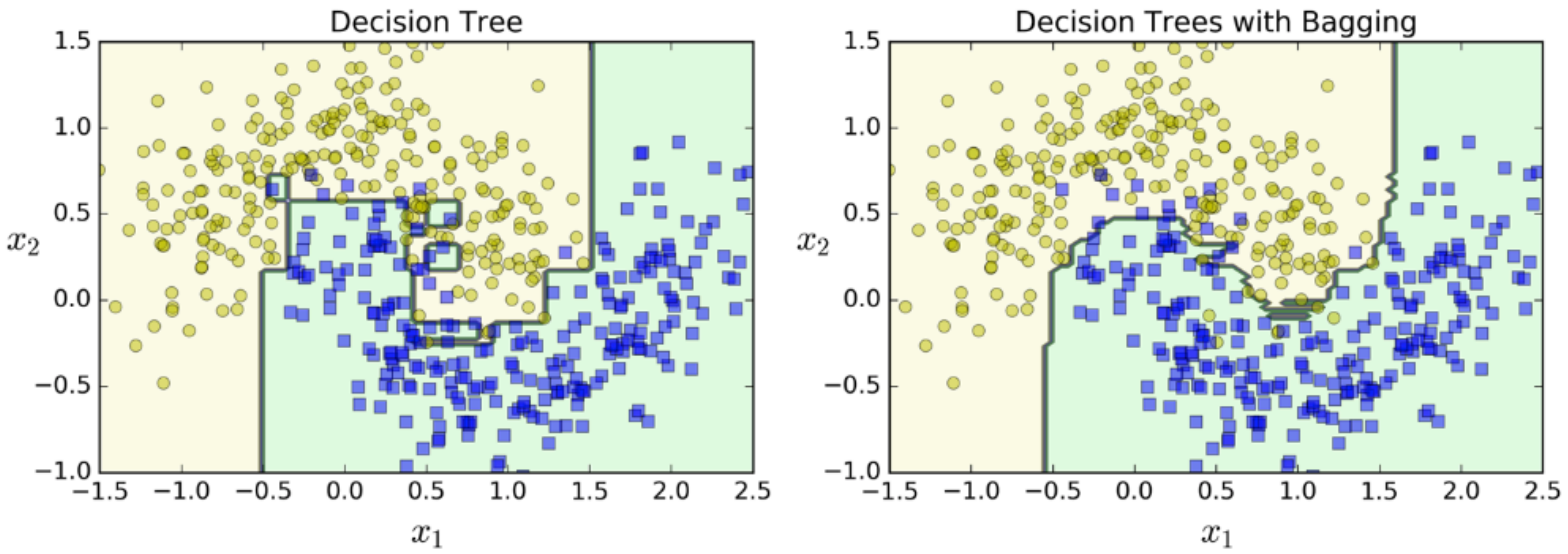
- Train 500 Decision Trees, each trained on 100 training instances using bagging

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

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(random_state=42), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1, random_state=42)
bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

Bagging Classifier in Python

- Train 500 Decision Trees, each trained on 100 training instances using bagging



Random Forest

Bagging Decision Trees

- The previous example use Bagging for an ensemble of Decision Trees. This is called a ***Random Forest.*** They are helpful for feature selection/importance.
- Random Forests can be used for classification or regression.
- When forming trees, it often ***searches for the best feature among a random subset of features*** (e.g. it does not look at all features).
  - This provides better diversity amongst trees
  - Produces a lower variance (but with higher bias).
- Use the *RandomForestClassifier* or *RandomForestRegressor* in Scikit-Learn

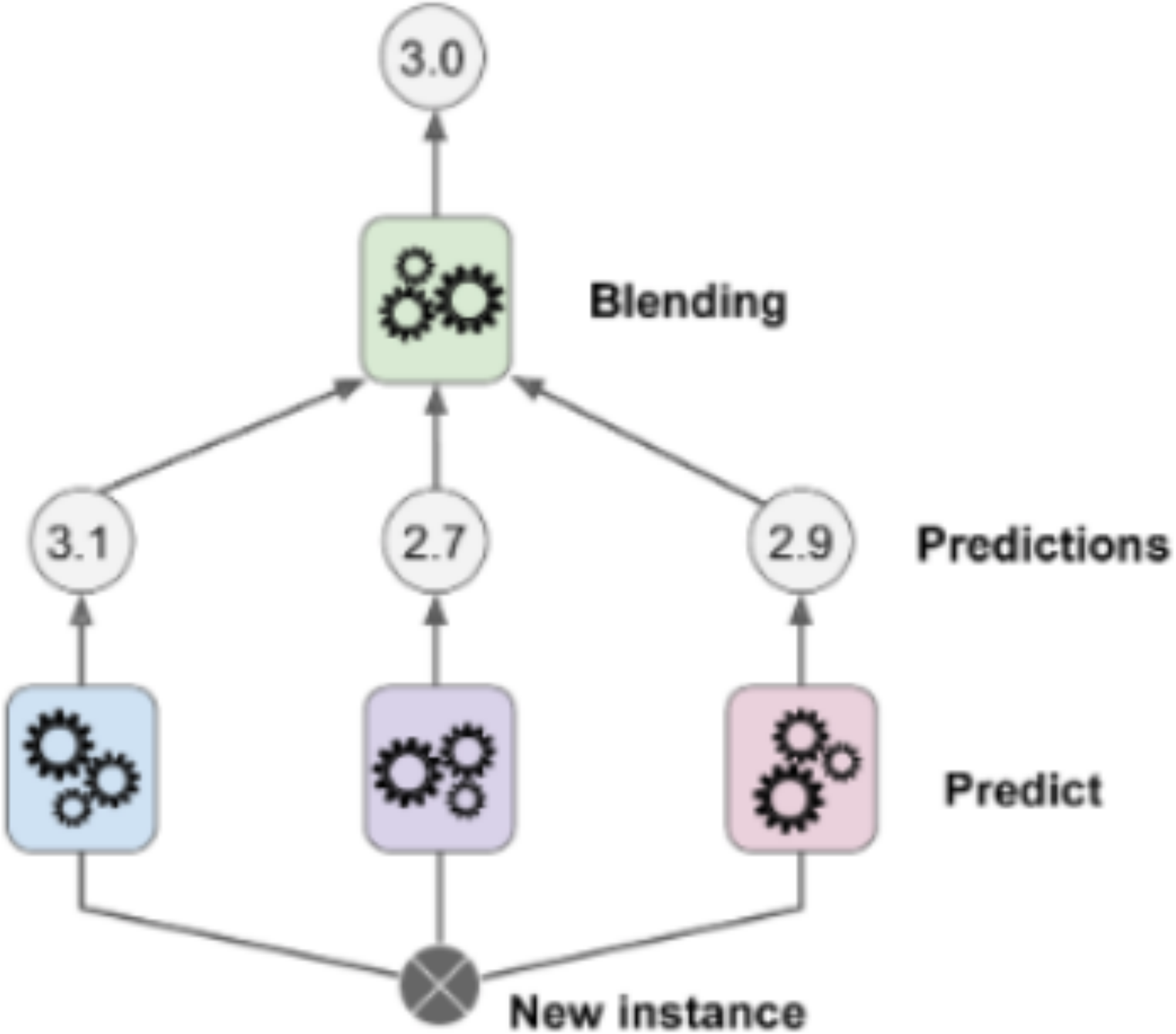
Other Ensemble Learnings

- **Boosting** - sequentially train learning algorithms, where subsequent predictors correct mistakes made by the predecessor. **Two *popular approaches are***
    - *AdaBoost* - Based on sample misclassification/error
    - *Gradient Boost* - Based on learners error
- **Stacking** (stacked generalization) - train a model to perform the aggregation between multiple learners.

Stacking

Another Ensemble approach

- Train a model to perform the aggregation between ensemble learners, instead of using average or majority vote.

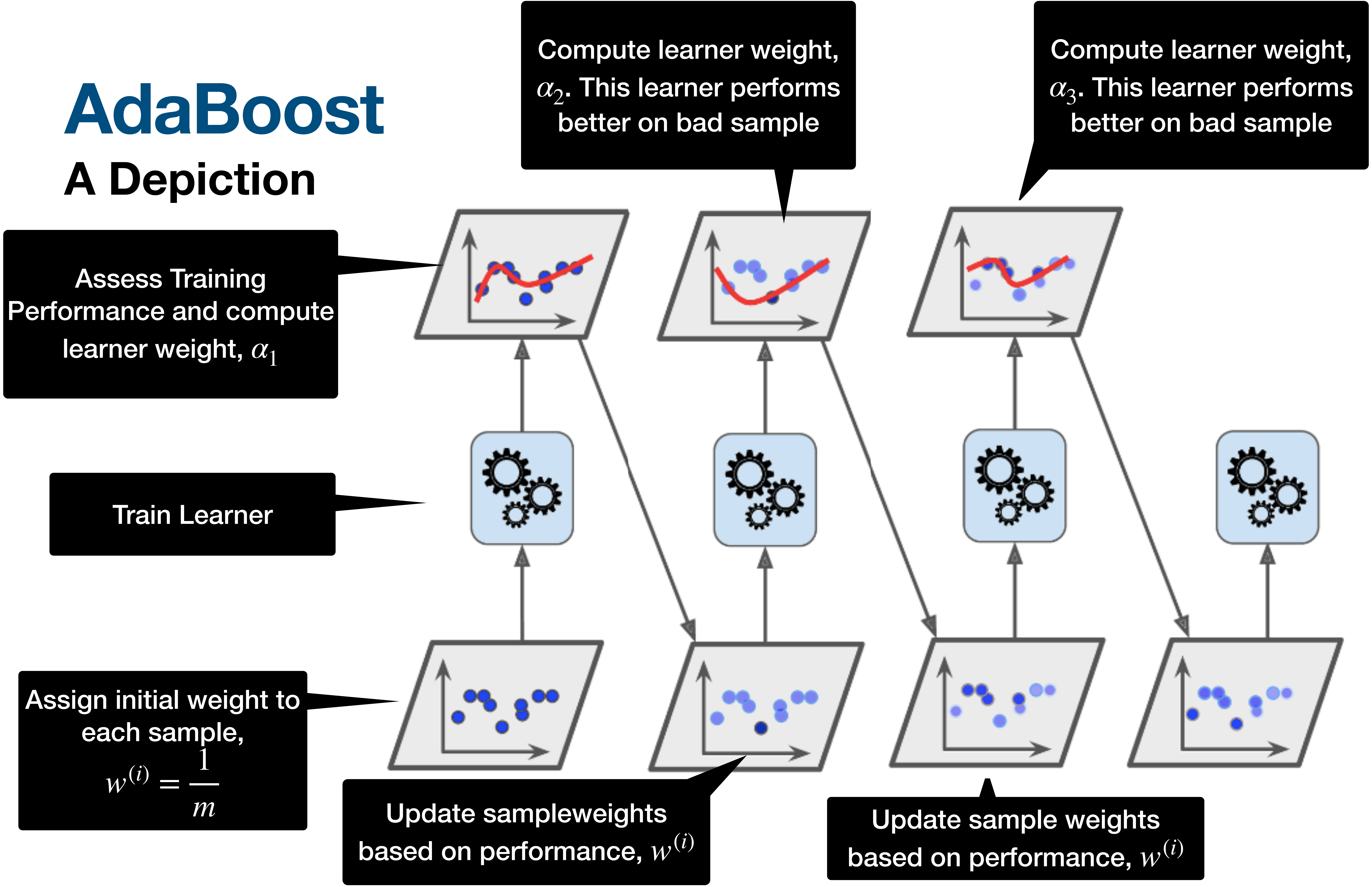


AdaBoost

A Sequential Approach

- **Idea**: Give weights to training instances (or samples)
  - Increase weight along the sequence (after classification/regression) for misclassified samples
  - Train subsequent learner where data is weighted
  - Add more learners and adjust sample weighing each time
- Finally, weigh predictions of each learner in the sequence, based on a learner weight
- Choose class that majority of weighted votes

AdaBoost
A Depiction



AdaBoost

Weight Updates

- **Sample-weight update rule** (assuming m samples)

$$\text{for } i = 1, 2, \cdots, m$$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp\left(\alpha_j\right) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$

- **Learner-weight calculation** for the jth learner in the sequence

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

- **Weighted error rate of jth learner**

$$r_j = \frac{\sum_{i=1, \hat{y}_j^{(i)} \neq y_j^{(i)}}^{m} w^{(i)}}{\sum_{i=1}^{m} w^{(i)}}$$

Gradient Boosting

Fit new learners to residual errors from predescors

- Train first learner:

```
from sklearn.tree import DecisionTreeRegressor

tree_reg1 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg1.fit(X, y)
```

- Compute residual error then train second learner to predict them

```
y2 = y - tree_reg1.predict(X)
tree_reg2 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg2.fit(X, y2)
```

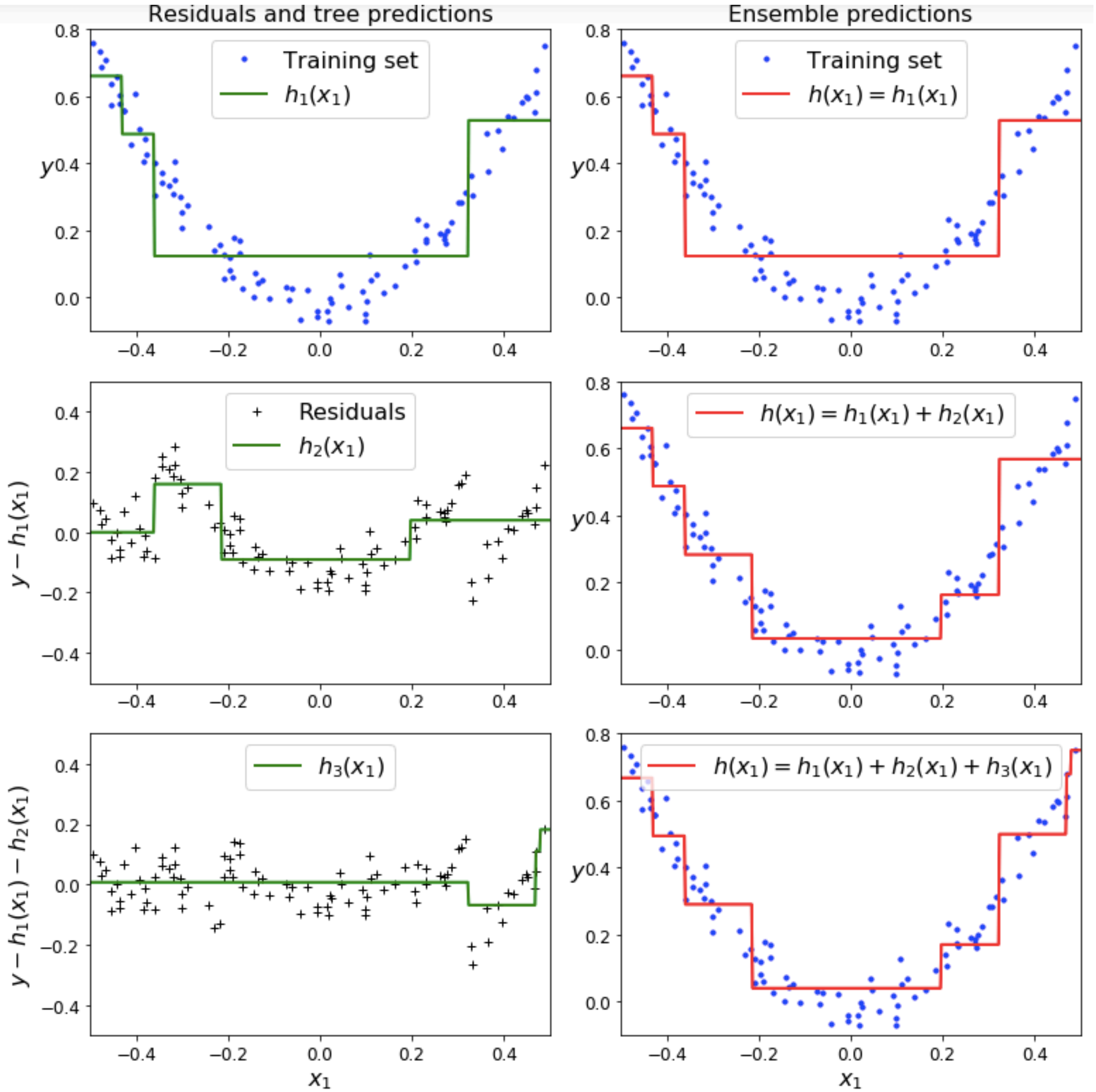
- Compute residual error from second learner and train third learner

```
y3 = y2 - tree_reg2.predict(X)
tree_reg3 = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg3.fit(X, y3)
```

Gradient Boosting

- Make predictions by adding predictions from each learner

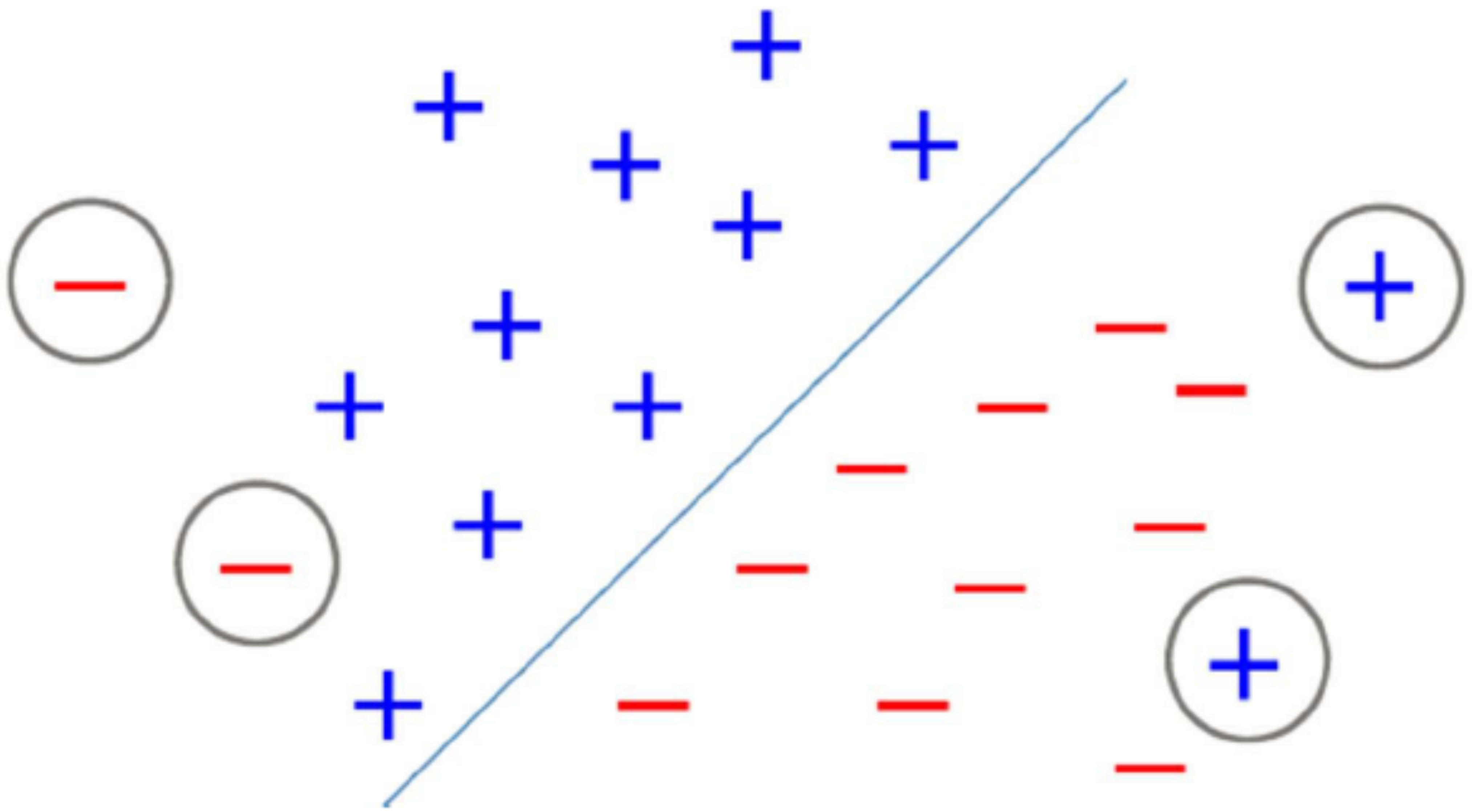
```
y_pred = sum(tree.predict(X_new) for tree in (tree_reg1, tree_reg2, tree_reg3))
```


Residuals and tree predictions
Ensemble predictions
Training set
$h_1(x_1)$
$h(x_1) = h_1(x_1)$
Residuals
$h_2(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1)$
$h_3(x_1)$
$h(x_1) = h_1(x_1) + h_2(x_1) + h_3(x_1)$
y
$y - h_1(x_1)$
$y - h_1(x_1) - h_2(x_1)$
x_1

Pitfall of Boosting

Sensitive to noise and outliers

- The Good: Can identify outliers since focuses on examples that are hard to categorize
- The Bad: Too many outliers can degrade classification performance dramatically increasing time to convergence



Summary: Ensemble Learning

Boosting and Bagging

- Bagging
  - Resample data points
  - Weight of each classifier is the same
  - Only variance reduction
  - Robust to noise and outliers
- Boosting
  - Reweight data points (modify data distribution)
  - Weight of classifier vary depending on accuracy
  - Reduces both bias and variance
  - Can hurt performance with noise and outliers

Next Class