KIG4068 : Machine Learning

Week 8: Ensemble Learning & Random Forest

Semester 2, Session 2023/2024

Ensemble Learning

What is ensemble learning?

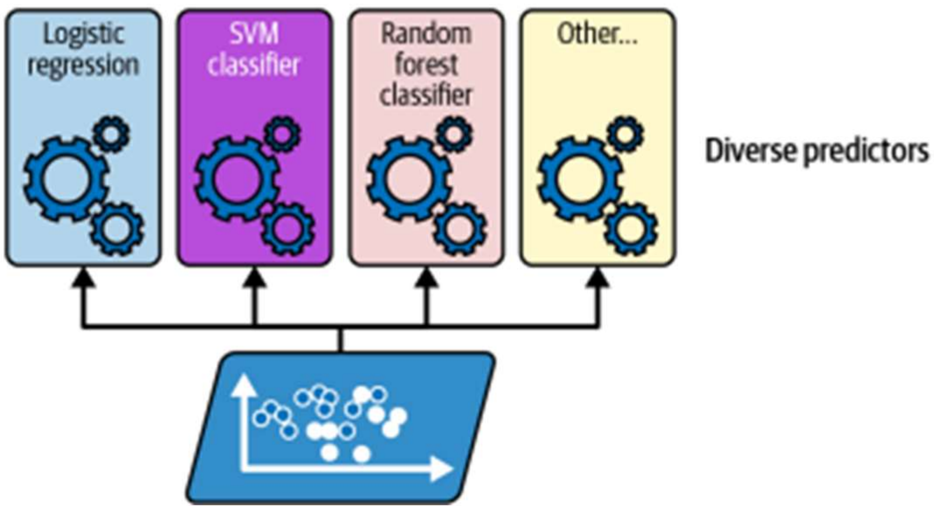
- Suppose you pose a complex question to thousands of random people, then aggregate their answers.
- Aggregated answer is often better than an expert's answer. This is called the *wisdom of the crowd*.
- Aggregate predictions of a group of predictors (such as classifiers or regressors), you will often get better predictions than with the best individual predictor.
- A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method.

What are we going to learn today?

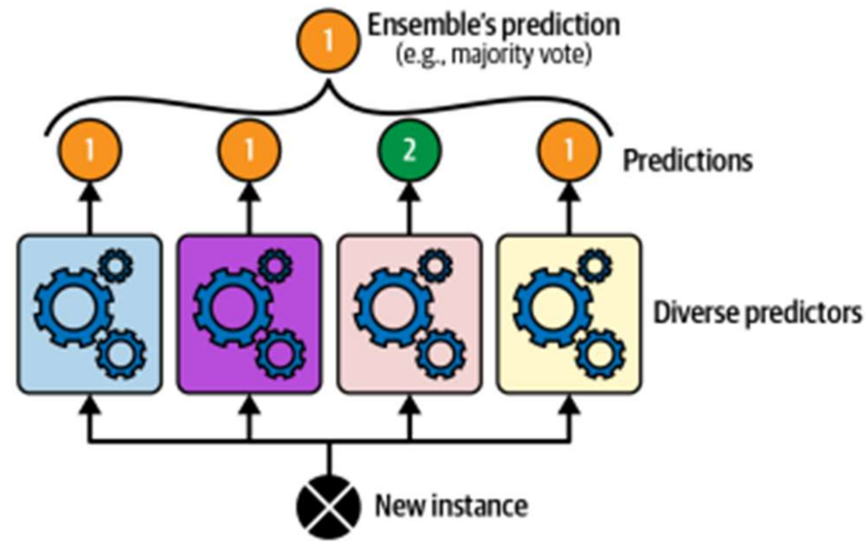
- Voting classifiers
- Bagging and pasting ensembles
- Random Forests
- Boosting
- Stacking ensembles

Voting Classifiers

Training diverse classifiers



Hard voting classifier predictions



- Voting classifiers often obtain better performance than the best individual classifier in the ensemble.
- Voting classifier can be a strong learner (good performance) even if individual classifiers are weak learners (poor performance).
- 2 types of voting: 1) Hard – majority votes, 2) Soft – probability average

Voting Classifier in Scikit-Learn

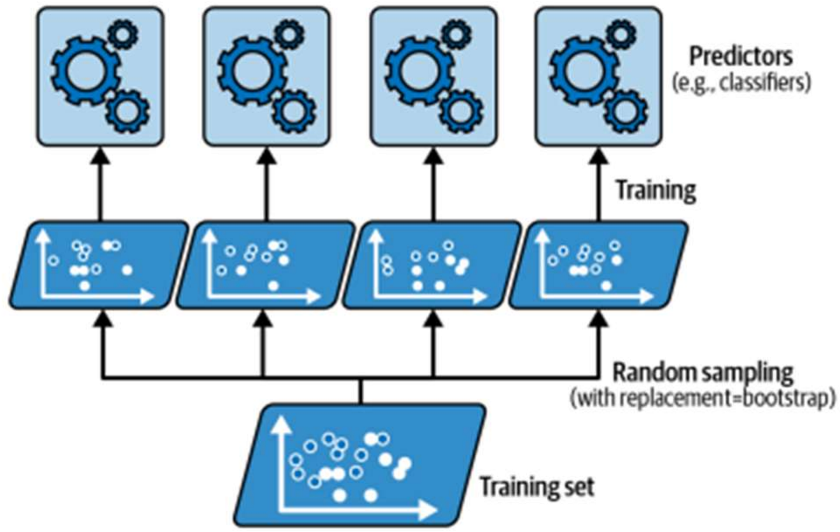
```
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

X_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)
```

Bagging and Pasting

What is Bagging and Pasting?

Example of training several predictors on different random samples

- Same training algorithm for every predictor but train them on different random subsets of the training set.
- Sampling training instances *with replacement* => bagging (boosted aggregating).
- Sampling training instances *without replacement* => pasting.
- With bagging, same training instance might be seen multiple times by the same predictor.



Bagging and Pasting in Scikit Learn

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

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

- n_estimators: number of decision tree classifiers
- max_samples: No. of samples to train each classifier
- n_jobs: number of job to run in parallel for both fitting and predicting. -1 meaning computation will be distributed across available CPU cores.

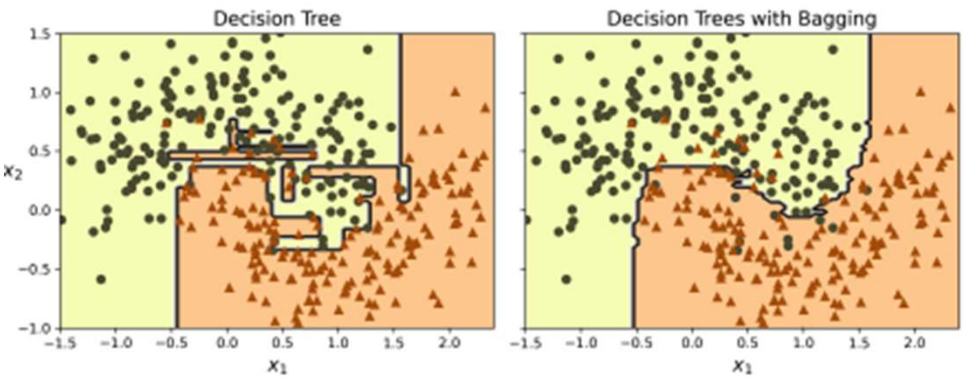
Note: set bootstrap = False in BaggingClassifier() if you want to use pasting instead, e.g. BaggingClassifier(bootstrap=False)

Bagging and Pasting (Cont'd)

Measure generalization error for "free"

Bagging as a form of regularization

- With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all.
- Unsampled instances are called *out-of-bag* instances and can be used to estimate generalization error.
- Bagging generally preferred to pasting; can use cross-validation to compare both approaches.
- Pasting can be advantageous in certain situations, such as when there are concerns about overfitting or when the dataset is relatively small.



The right plot (with bagging ensemble of 500 trees) generalizes better than a standard decision tree

Bagging and Pasting (Cont'd)

Out-of-Bag (OOB) evaluation

- When training each estimator in the ensemble, some samples may not be used to train that particular estimator; these samples are called out-of-bag (OOB) samples.
- These OOB samples will be used as a validation set for the estimator.
- The accuracy of the model can be calculated based on these OOB samples to estimate the model's performance on unseen data.
- Set oob_score=True to request and automatic OOB evaluation after training:

```
bag_clf = BaggingClassifier(DecisionTreeClassifier(), n_estimators=500,
                            oob_score=True, n_jobs=-1, random_state=42)

bag_clf.fit(X_train, y_train)
bag_clf.oob_score_
```

Random Forests

Ensemble of decision trees

- Generally trained using bagging (pasting is also possible).
- Instead of searching for best feature when splitting a node (decision tree in Week 7), algorithm searches for the best feature amongst a random subset of features.
- Extra randomness => greater predictor diversity => trades higher bias for lower variance => better generalization.
- Lots of hyperparameters for tuning!
- Class in Scikit Learn:

  RandomForestClassifier

Extremely randomized trees (Extra-Trees)

- Make trees even more random by also using random thresholds for each feature rather than searching for the best possible thresholds.
- Again, trade higher bias for lower variance and better generalization.
- Much faster to train than Random Forests as finding best threshold for each feature is time consuming.
- Class in Scikit Learn:

  ExtraTreesClassifier

Random Forests in Scikit Learn

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=16,
                                 n_jobs=-1, random_state=42)
rnd_clf.fit(X_train, y_train)

y_pred_rf = rnd_clf.predict(X_test)
```

Random Forest Classifier (using Bagging and Decision Tree Classifiers)

```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(max_features="sqrt", max_leaf_nodes=16),
    n_estimators=500, n_jobs=-1, random_state=42)
```

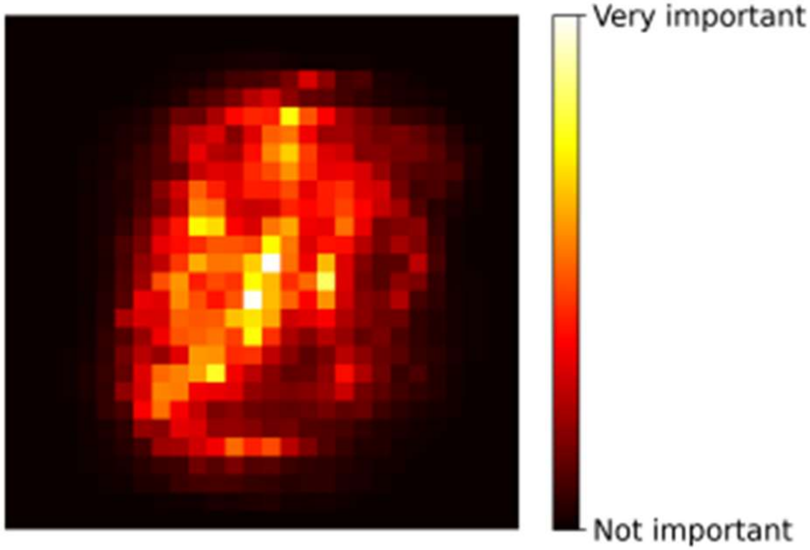
These two codes are equivalent to each other

Random Forest (Cont’d)

Feature importance

Pixel importance using MNIST data

- Easy to measure the relative importance of each feature.
- Measure how much the tree nodes that use a feature reduce impurity on average (across all trees in the forest).
- Weighted average, where each node’s weight is equal to the number of training samples that are associated with it.
- Often useful for feature selection.



Boosting

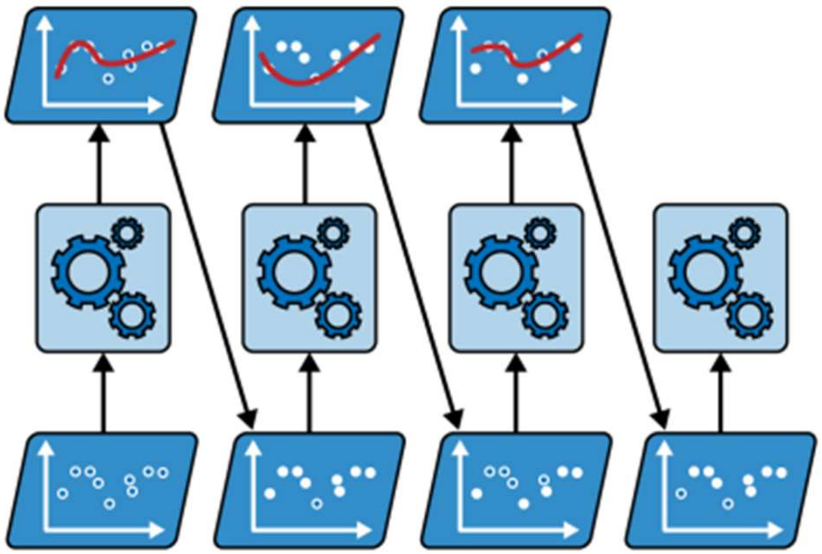
- Ensemble method that combines several weak learners into a strong learner.
- General idea is to train predictors sequentially with each additional predictor trying to correct mistakes its predecessor.
- Two most widely used approaches are AdaBoost and Gradient Boosting.

Boosting (Cont'd)

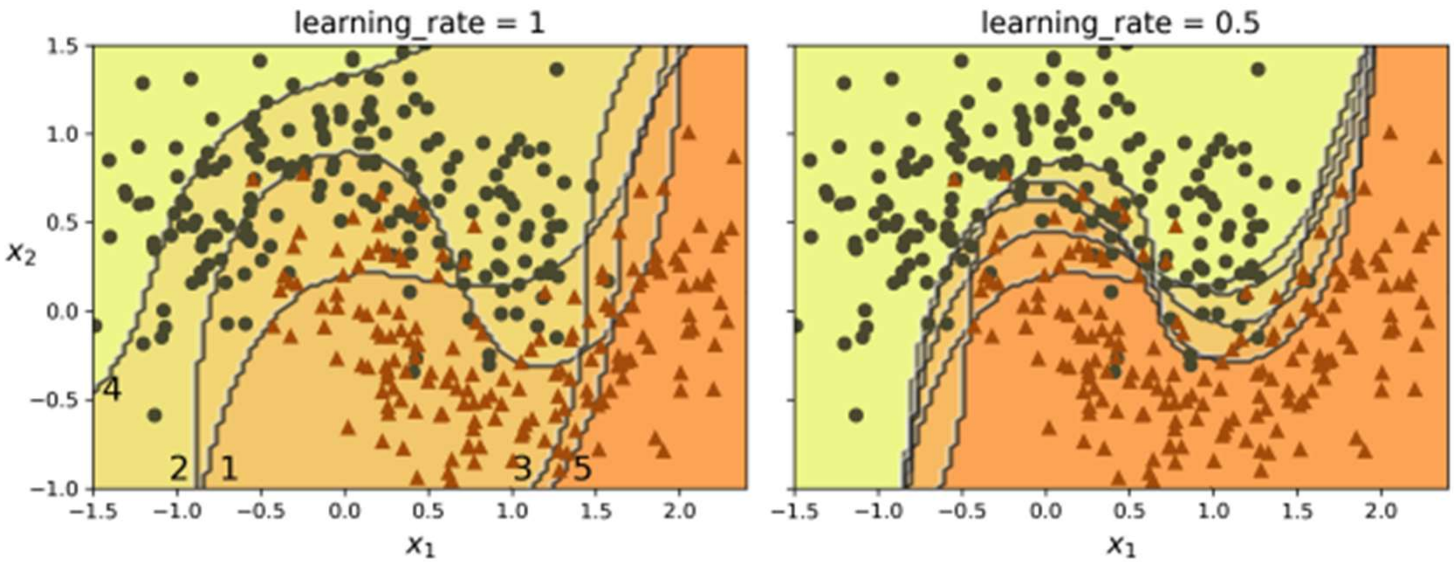
AdaBoost

Sequential training + weighting updates

- Each new predictor pays a bit more attention to the training instances that its predecessor underfitted.
- New predictors increasingly focus on "hard" training instances.
- Ensemble makes predictions like bagging/pasting: predictors have different weights depending on their overall accuracy on the training data.
- Drawback: sequential nature of training => limit opportunity for parallelism.



AdaBoost (Cont'd)



- Decision boundaries of five consecutive predictors on moon dataset.
- On the left, the first classifier gets many instances wrong; weights get boosted.
- The second does a good job on this instances, and so on..
- On the right represents the same sequence of predictors, except that the learning rate is halved (misclassified instance weights are boosted much less at every iteration).
- Sequential learning tech. has some similarities with gradient descent, except instead of tweaking single predictor's parameters, AdaBoost adds predicters

AdaBoost Classifier in Scikit Learn

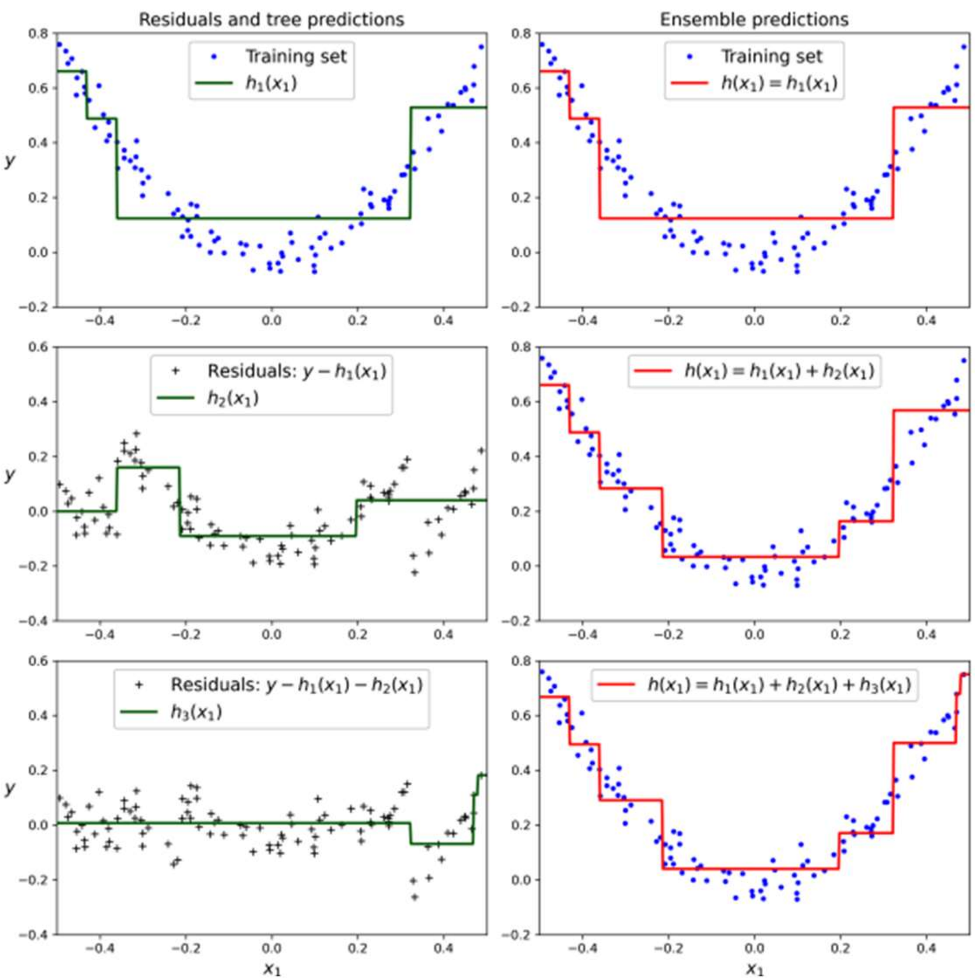
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1), n_estimators=30,
    learning_rate=0.5, random_state=42)
ada_clf.fit(X_train, y_train)
```

Boosting (Cont'd)

Gradient Boosting

Fit new predictors to residual errors

- Works by sequentially adding predictors to an ensemble, each one correcting its predecessor, just like AdaBoost.
- However, instead of tweaking the instance weights at every iteration like AdaBoost does, this method tries to fit new predictor to the *residual errors* made by the previous predictor
- *Residual errors*: the difference between the predicted and actual target values in the training data.



Gradient Boosting in Scikit Learn

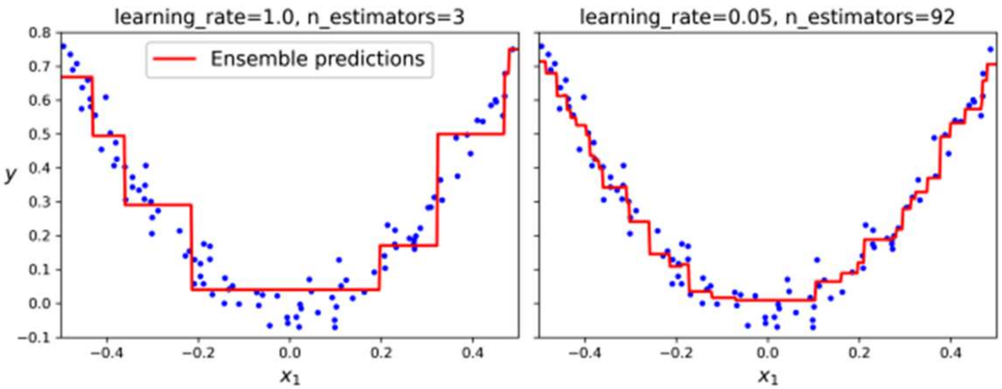
```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3,
                                 learning_rate=1.0, random_state=42)
gbrt.fit(X, y)
```

Boosting (Cont'd)

Gradient Boosting

N=3 too few (underfits), N=200 too many (overfits)

- LR hyperparameter scales the contribution of each tree.
- Low LR => more trees required to fit, but the predictions usually generalize better.
- This is a regularization technique called *shrinkage*.



Set an early stop using n_estimators:

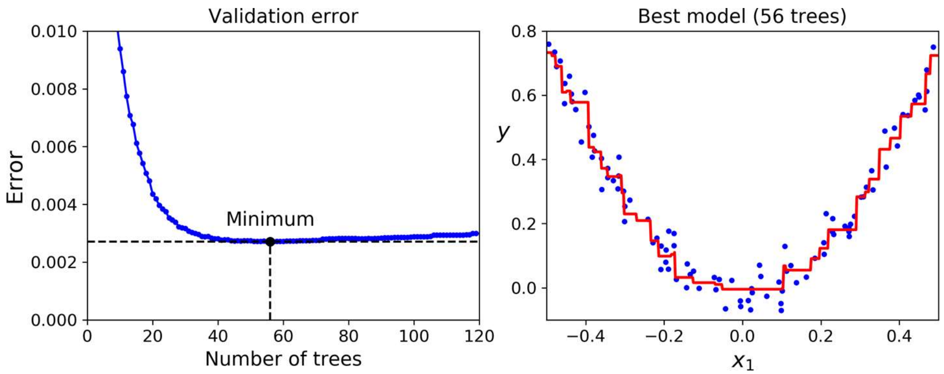
```
gbrt_best = GradientBoostingRegressor(
    max_depth=2, learning_rate=0.05, n_estimators=500,
    n_iter_no_change=10, random_state=42)
gbrt_best.fit(X, y)
```

Boosting (Cont'd)

How to choose the number of estimators?

Looks like N=56 estimators is "just right"!

- Use a form of early stopping.
- Measure the validation error as you add estimators to the ensemble.
- Stop adding estimators when the validation error starts to increase.

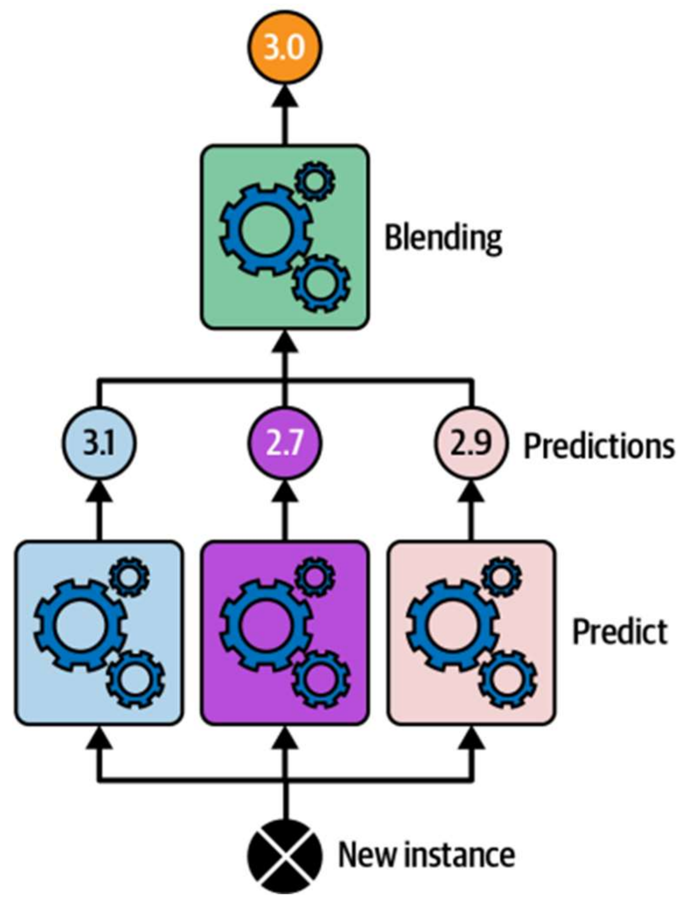


Stacking

Stacked generalization

- Voting classifiers use simple functions to aggregate ensemble predictions.
- Instead of using simple functions, why not train another model to aggregate ensemble predictions?
- Final model is called *blender* or *meta-learner.*

Aggregating predictions using a blender

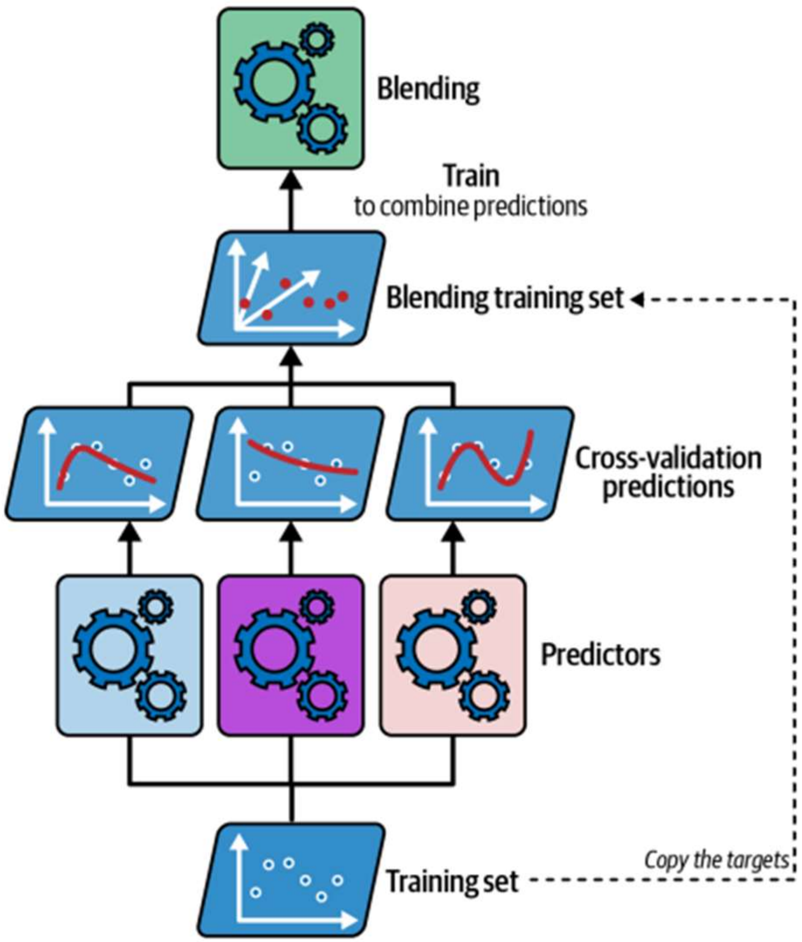


Stacking (Cont'd)

How to train the blender?

- ***Create blending training set***: Generate predictions from each base predictor on the original training data.
- ***Train the blender***: Train the blender using the base predictors' prediction as input feature and the original training set outcomes as targets.
- ***Retrain base predictors***: Retrain the base predictors one final time on the full original training set after training the blender.

Training the blender in a stacking ensemble

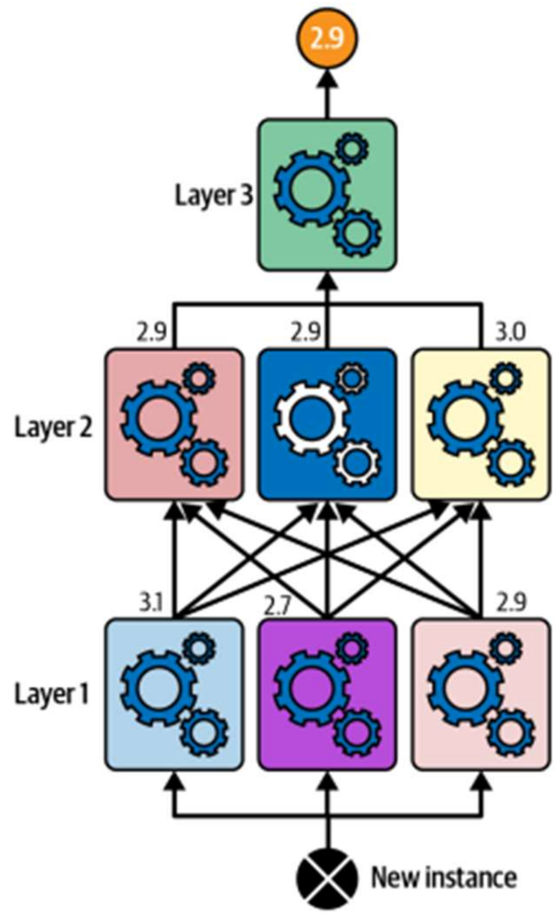


Stacking (Cont’d)

Multi-layer stacking

- ***Multiple blenders in a layer***: training diverse blenders to form a complete layer within the stacking ensemble
- ***Adding another blender for final prediction***: introduce an additional blender on top of the existing layer to produce the final prediction, combining outputs of the previous layer’s blenders
- Multi-layer stacking may slightly enhance performance, but increases training time and system complexity

Predictions in multi-layer stacking ensemble



Stacking in Scikit Learn

```python
from sklearn.ensemble import StackingClassifier

stacking_clf = StackingClassifier(
    estimators=[
        ('lr', LogisticRegression(random_state=42)),
        ('rf', RandomForestClassifier(random_state=42)),
        ('svc', SVC(probability=True, random_state=42))
    ],
    final_estimator=RandomForestClassifier(random_state=43),
    cv=5  # number of cross-validation folds
)
stacking_clf.fit(X_train, y_train)
```