Chapter 7. Ensemble Leaning and Random Forests

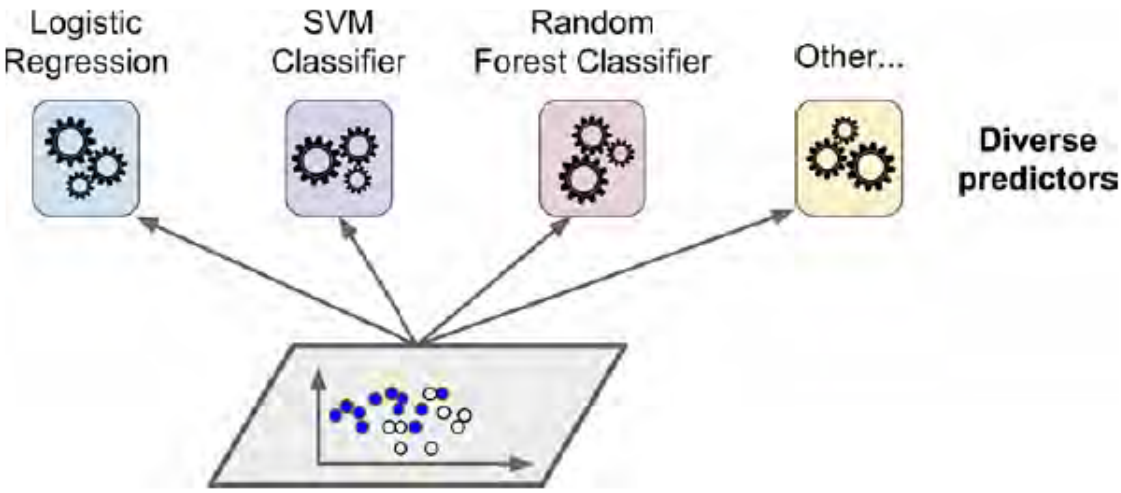
1. What is Ensemble Learning?

The idea is that if we aggegate the predictions of a group of predictors such as classifiers or regressors, we will often get better predictions than with the best individual predictor. A group of predictors is called an *ensemble* and this technique is called Ensemble Learning. An Ensemble Learning algorithm is called an Ensemble Method

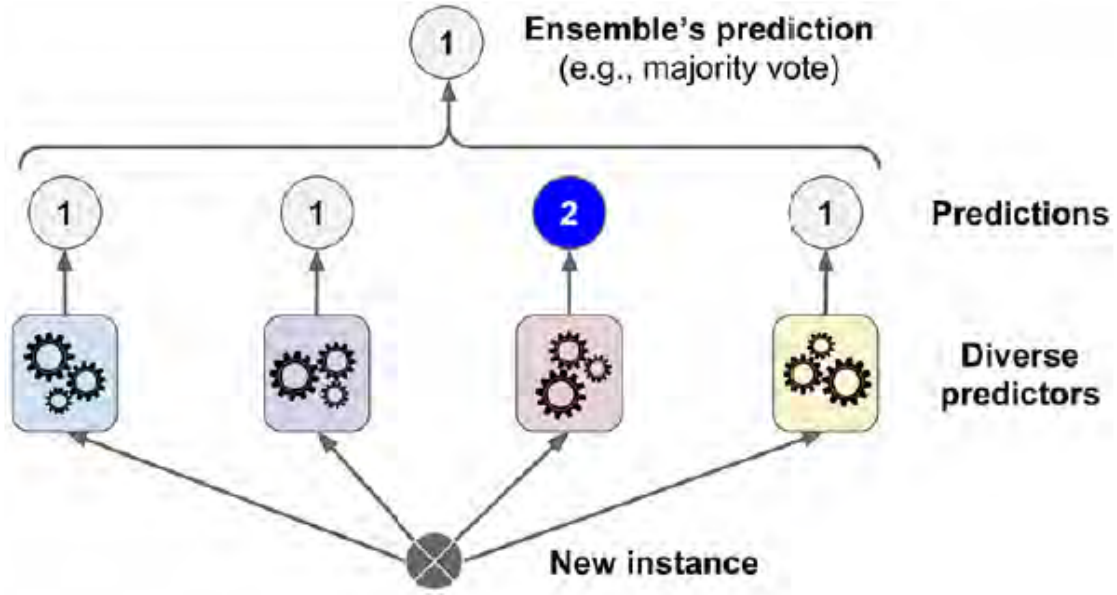
An example of Ensemble Method is the Random Forest. We train a group of Decision Tree classifiers, each on a different random subset of the training set and take the majority vote class as the final prediction.

2. Voting Classifiers

2.1 Hard Voting Classifier



To create an even better classifier, we can aggregate the predictions of each classifier and predict the class that gets the most votes. This is called a hard voting classifier



- Even if each classifier is a weak learner (does only slightly better than randomly guessing), the ensemble can still be a strong learner (achieving high accuracy), provided there are a sufficient number of weak learners and they are sufficiently diverse
- Ensemble methods work best when the predictors are as independent from one another as possible. One way to get diverse classifiers is to train them using very different algorithms. This increases the chance that they will make very different types of errors hence improving the ensemble's accuracy

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_moons

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)
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC

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

voting_clf = VotingClassifier(
    estimators=[('lr', log_clf),
                ('rf_clf', rf_clf),
                ('svc', svm_clf)],
    voting="hard")

voting_clf.fit(X_train, y_train)
```

Out[ ]:
```
VotingClassifier(estimators=[('lr', LogisticRegression()),
                             ('rf_clf', RandomForestClassifier()),
                             ('svc', SVC())])
```

Indivdual classifier's accuracy

In [ ]:
```
from sklearn.metrics import accuracy_score
```

```
for clf in (log_clf, rf_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.888
SVC 0.896
VotingClassifier 0.896
```

2.2 Soft Voting Classifier

If all classifiers are able to estimate class probabilties, we can predict the class with the highest class probability, averaged over all the individual classifiers.

Soft voting classifier often achieves higher performance than hard voting because it gives more weight to highly confident votes.

In [ ]:
```
log_clf = LogisticRegression(solver="lbfgs", random_state=42)
rnd_clf = RandomForestClassifier(n_estimators=100, random_state=42)
svm_clf = SVC(gamma="scale", probability=True, random_state=42)

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

Out[ ]:
```
VotingClassifier(estimators=[('lr', LogisticRegression(random_state=42)),
                             ('rf', RandomForestClassifier(random_state=42)),
                             ('svc', SVC(probability=True, random_state=42))],
                 voting='soft')
```

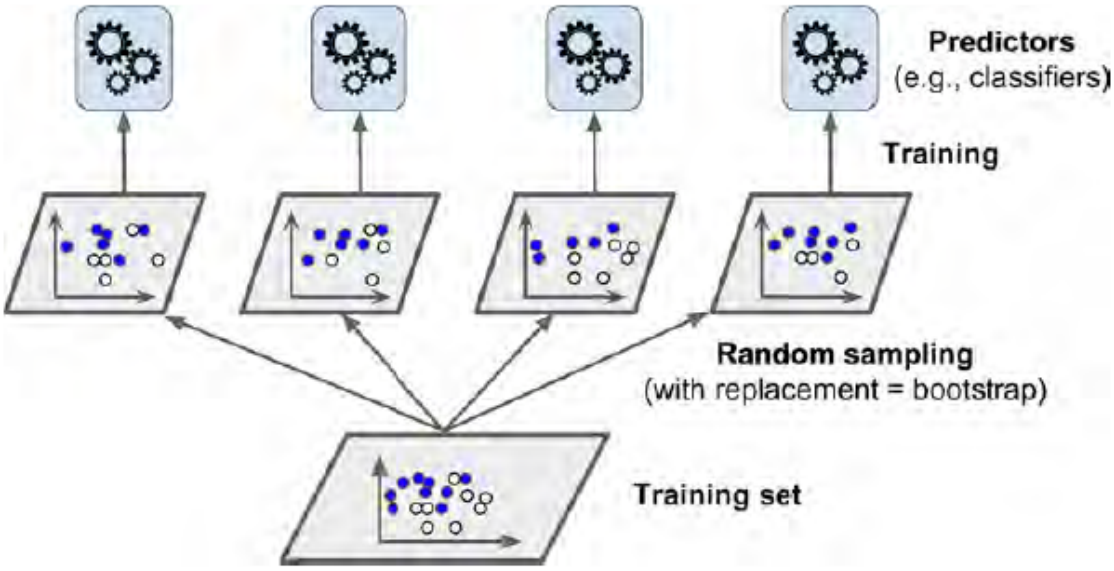
In [ ]:
```
from sklearn.metrics import accuracy_score

for clf in (log_clf, rnd_clf, svm_clf, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
```

```
LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.896
VotingClassifier 0.92
```

3. Bagging and Pasting

- **Bagging (Bootstrap Aggregating)**: Sampling with replacement
- **Pasting**: Sampling without replacement



The ensemble can make a prediction for a new instance by aggregating the predictions of all predcitors.

- Classification: Majority Vote
- Regression: Average value

Aggregating reduces both bias and variance. The net result is that the ensemble has a similar bias but a lower variance than a single predcitor trained on the original training set.

Bagging Classifier automatically performs soft voting instead of hard voting if the base classifier can estimate probabiltiies.

In [ ]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    max_samples=100, bootstrap=True, n_jobs=-1) # for pasting, bootstrap=False. n_

bag_clf.fit(X_train, y_train)
y_pred = bag_clf.predict(X_test)
```

In [ ]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
0.92
```

In [ ]:
```
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))
```

```
0.856
```

3.1 Out-of-Bag Evaluation

Since a predictor never sees the oob instances during training, it can be evaluated on these instances without the need for a separate validation set.

In [ ]:
```
bag_clf = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators=500,
    bootstrap=True, n_jobs=-1, oob_score=True)

bag_clf.fit(X_train, y_train)

bag_clf.oob_score_
```

Out[ ]:
```
0.896
```

In [ ]:
```
from sklearn.metrics import accuracy_score
y_pred = bag_clf.predict(X_test)
accuracy_score(y_test, y_pred) # close enough to the oob score
```

Out[ ]:
```
0.912
```

In [ ]:
```
bag_clf.oob_decision_function_
```

Out[ ]:
```
array([[0.36507937, 0.63492063],
       [0.38043478, 0.61956522],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.14942529, 0.85057471],
       [0.32960894, 0.67039106],
       [0.01522843, 0.98477157],
       [0.99468085, 0.00531915],
       [0.98461538, 0.01538462],
       [0.73023256, 0.26976744],
       [0.        , 1.        ],
       [0.85714286, 0.14285714],
       [0.8423913 , 0.1576087 ],
       [0.97282609, 0.02717391],
       [0.07142857, 0.92857143],
       [0.        , 1.        ],
       [0.98039216, 0.01960784],
       [0.96089385, 0.03910615],
       [0.99435028, 0.00564972],
       [0.04891304, 0.95108696],
       [0.37755102, 0.62244898],
       [0.92771084, 0.07228916],
       [1.        , 0.        ],
       [0.97350993, 0.02649007],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.68648649, 0.31351351],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.18324607, 0.81675393],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.41116751, 0.58883249],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [0.25128205, 0.74871795],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.02139037, 0.97860963],
       [1.        , 0.        ],
       [0.02808989, 0.97191011],
       [0.99438202, 0.00561798],
       [0.89830508, 0.10169492],
       [0.97175141, 0.02824859],
       [0.96585366, 0.03414634],
       [0.        , 1.        ],
       [0.05464481, 0.94535519],
       [0.97860963, 0.02139037],
       [0.        , 1.        ],
       [0.        , 1.        ],
       [0.01098901, 0.98901099],
       [0.98461538, 0.01538462],
       [0.77011494, 0.22988506],
       [0.43715847, 0.56284153],
       [1.        , 0.        ],
```

```
[0.        , 1.        ],
[0.69767442, 0.30232558],
[1.        , 0.        ],
[1.        , 0.        ],
[0.84530387, 0.15469613],
[1.        , 0.        ],
[0.63687151, 0.36312849],
[0.14634146, 0.85365854],
[0.66161616, 0.33838384],
[0.9047619 , 0.0952381 ],
[0.        , 1.        ],
[0.17877095, 0.82122905],
[0.86979167, 0.13020833],
[1.        , 0.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.03465347, 0.96534653],
[0.03108808, 0.96891192],
[0.32386364, 0.67613636],
[1.        , 0.        ],
[0.00578035, 0.99421965],
[0.85263158, 0.14736842],
[0.00483092, 0.99516908],
[0.        , 1.        ],
[0.        , 1.        ],
[0.23728814, 0.76271186],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.93969849, 0.06030151],
[0.79227053, 0.20772947],
[0.0049505 , 0.9950495 ],
[1.        , 0.        ],
[0.20689655, 0.79310345],
[0.61827957, 0.38172043],
[0.        , 1.        ],
[0.05202312, 0.94797688],
[0.5       , 0.5       ],
[1.        , 0.        ],
[0.01694915, 0.98305085],
[0.99390244, 0.00609756],
[0.26237624, 0.73762376],
[0.54347826, 0.45652174],
[1.        , 0.        ],
[0.01630435, 0.98369565],
[0.9950495 , 0.0049505 ],
[0.28571429, 0.71428571],
[0.86868687, 0.13131313],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.85227273, 0.14772727],
[1.        , 0.        ],
[0.00555556, 0.99444444],
[1.        , 0.        ],
[1.        , 0.        ],
[1.        , 0.        ],
[0.98181818, 0.01818182],
[1.        , 0.        ],
[0.        , 1.        ],
[0.953125  , 0.046875  ],
```

```
[1.        , 0.        ],
[0.01538462, 0.98461538],
[0.27918782, 0.72081218],
[0.95833333, 0.04166667],
[0.28494624, 0.71505376],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.70760234, 0.29239766],
[0.36627907, 0.63372093],
[0.45454545, 0.54545455],
[0.83977901, 0.16022099],
[0.94623656, 0.05376344],
[0.06629834, 0.93370166],
[0.84065934, 0.15934066],
[0.01098901, 0.98901099],
[0.        , 1.        ],
[0.0199005 , 0.9800995 ],
[0.95897436, 0.04102564],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.03414634, 0.96585366],
[0.00510204, 0.99489796],
[1.        , 0.        ],
[1.        , 0.        ],
[0.9558011 , 0.0441989 ],
[1.        , 0.        ],
[1.        , 0.        ],
[0.9895288 , 0.0104712 ],
[0.        , 1.        ],
[0.32369942, 0.67630058],
[0.2392638 , 0.7607362 ],
[0.00526316, 0.99473684],
[0.        , 1.        ],
[0.25698324, 0.74301676],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.98421053, 0.01578947],
[0.        , 1.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.01219512, 0.98780488],
[0.63636364, 0.36363636],
[0.91666667, 0.08333333],
[0.        , 1.        ],
[0.98947368, 0.01052632],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.05128205, 0.94871795],
[1.        , 0.        ],
[0.03821656, 0.96178344],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.03867403, 0.96132597],
```

```
[0.99435028, 0.00564972],
[0.95505618, 0.04494382],
[0.74331551, 0.25668449],
[0.60294118, 0.39705882],
[0.        , 1.        ],
[0.12777778, 0.87222222],
[1.        , 0.        ],
[0.92553191, 0.07446809],
[0.96842105, 0.03157895],
[1.        , 0.        ],
[0.00518135, 0.99481865],
[0.        , 1.        ],
[0.35869565, 0.64130435],
[0.9039548 , 0.0960452 ],
[0.        , 1.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.00995025, 0.99004975],
[0.        , 1.        ],
[0.92655367, 0.07344633],
[0.        , 1.        ],
[0.25925926, 0.74074074],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.98342541, 0.01657459],
[0.77777778, 0.22222222],
[1.        , 0.        ],
[0.        , 1.        ],
[0.12068966, 0.87931034],
[0.99462366, 0.00537634],
[0.02604167, 0.97395833],
[0.        , 1.        ],
[0.05376344, 0.94623656],
[1.        , 0.        ],
[0.76608187, 0.23391813],
[0.        , 1.        ],
[0.90954774, 0.09045226],
[1.        , 0.        ],
[0.15662651, 0.84337349],
[0.22916667, 0.77083333],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.1827957 , 0.8172043 ],
[0.96315789, 0.03684211],
[0.        , 1.        ],
[1.        , 0.        ],
[0.98224852, 0.01775148],
[0.        , 1.        ],
[0.52542373, 0.47457627],
[1.        , 0.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.14619883, 0.85380117],
[0.0862069 , 0.9137931 ],
[0.97714286, 0.02285714],
[0.04      , 0.96      ],
[1.        , 0.        ],
[0.38728324, 0.61271676],
```

```
[0.1299435 , 0.8700565 ],
[0.55675676, 0.44324324],
[0.60427807, 0.39572193],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.58602151, 0.41397849],
[0.        , 1.        ],
[1.        , 0.        ],
[0.23295455, 0.76704545],
[0.83248731, 0.16751269],
[0.07      , 0.93      ],
[1.        , 0.        ],
[0.82080925, 0.17919075],
[0.        , 1.        ],
[0.        , 1.        ],
[0.12290503, 0.87709497],
[0.01219512, 0.98780488],
[0.        , 1.        ],
[1.        , 0.        ],
[0.90229885, 0.09770115],
[0.15217391, 0.84782609],
[0.96703297, 0.03296703],
[0.00552486, 0.99447514],
[0.58045977, 0.41954023],
[0.07009346, 0.92990654],
[0.98429319, 0.01570681],
[0.80729167, 0.19270833],
[0.        , 1.        ],
[1.        , 0.        ],
[0.95212766, 0.04787234],
[0.        , 1.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[0.22916667, 0.77083333],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.87165775, 0.12834225],
[0.        , 1.        ],
[1.        , 0.        ],
[0.7173913 , 0.2826087 ],
[0.96174863, 0.03825137],
[1.        , 0.        ],
[0.70175439, 0.29824561],
[0.57837838, 0.42162162],
[0.00561798, 0.99438202],
[0.90547264, 0.09452736],
[0.        , 1.        ],
[1.        , 0.        ],
[0.85106383, 0.14893617],
[1.        , 0.        ],
[1.        , 0.        ],
[0.78804348, 0.21195652],
[0.0875    , 0.9125    ],
[0.47643979, 0.52356021],
[0.22994652, 0.77005348],
[0.        , 1.        ],
[0.88888889, 0.11111111],
[0.85632184, 0.14367816],
```

```
[0.00483092, 0.99516908],
[1.        , 0.        ],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.02525253, 0.97474747],
[0.97237569, 0.02762431],
[0.92670157, 0.07329843],
[1.        , 0.        ],
[0.56565657, 0.43434343],
[1.        , 0.        ],
[0.        , 1.        ],
[0.99441341, 0.00558659],
[0.02898551, 0.97101449],
[1.        , 0.        ],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.98876404, 0.01123596],
[0.        , 1.        ],
[0.08823529, 0.91176471],
[0.        , 1.        ],
[0.        , 1.        ],
[1.        , 0.        ],
[1.        , 0.        ],
[0.        , 1.        ],
[0.99497487, 0.00502513],
[0.00613497, 0.99386503],
[1.        , 0.        ],
[0.14367816, 0.85632184],
[0.        , 1.        ],
[0.        , 1.        ],
[0.        , 1.        ],
[0.44382022, 0.55617978],
[0.05294118, 0.94705882],
[0.25988701, 0.74011299],
[1.        , 0.        ],
[0.98958333, 0.01041667],
[0.22404372, 0.77595628],
[0.99428571, 0.00571429],
[0.00546448, 0.99453552],
[0.        , 1.        ],
[1.        , 0.        ],
[0.97948718, 0.02051282],
[0.32642487, 0.67357513],
[0.98924731, 0.01075269],
[1.        , 0.        ],
[0.        , 1.        ],
[0.98924731, 0.01075269],
[0.        , 1.        ],
[0.03389831, 0.96610169],
[0.99038462, 0.00961538],
[1.        , 0.        ],
[0.01612903, 0.98387097],
[0.72413793, 0.27586207]])
```

4. Random Patches and Random Subspaces

BaggingClassifier class supports sampling the features. Sampling is controlled by two hyperparameters: `max_features` and `bootstrap-features`

- **Random Patches**: Sampling both training instances and features
- **Random Subspaces**: Keeping all training instances but sampling features

5. Random Forests

Random Forests is an ensemble of Decision Trees, generally trained via the bagging method.

Random Forest algorithm produces extra randomness when growing trees.

- At each node, only a random subset of features is considered for splitting.
- This results in greater tree diversity which trades for higher bias for a lower variance.

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

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

y_pred_rf = rnd_clf.predict(X_test)
```

5.1. Extremely Randomized Trees (Extra-Trees)

Make trees even more random by using random thresholds for each feature rather than searching for the best possible thresholds.

- This technique trades more bias for a lower variance.
- Much faster algorithm to train than regular Random Forests because finding the best possible threshold for each feature at every node is one of the most time-consuming tasks of growing a tree

5.2 Feature Importance

Random Forests are very handy to get a quick understanding of what features matter, if we need to perform feature selection

In [ ]:
```
from sklearn.datasets import load_iris
iris = load_iris()
rnd_clf = RandomForestClassifier(n_estimators=500, random_state=42)
rnd_clf.fit(iris["data"], iris["target"])
for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
    print(name, score)
```

```
sepal length (cm) 0.11249225099876375
sepal width (cm) 0.02311928828251033
petal length (cm) 0.4410304643639577
petal width (cm) 0.4233579963547682
```

Most important features are the petal length (44%) and width (42%)

In [ ]:
```
rnd_clf.feature_importances_
```

Out[ ]:
```
array([0.11249225, 0.02311929, 0.44103046, 0.423358  ])
```

6. Boosting

Boosting: refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.

Types of boosting methods:

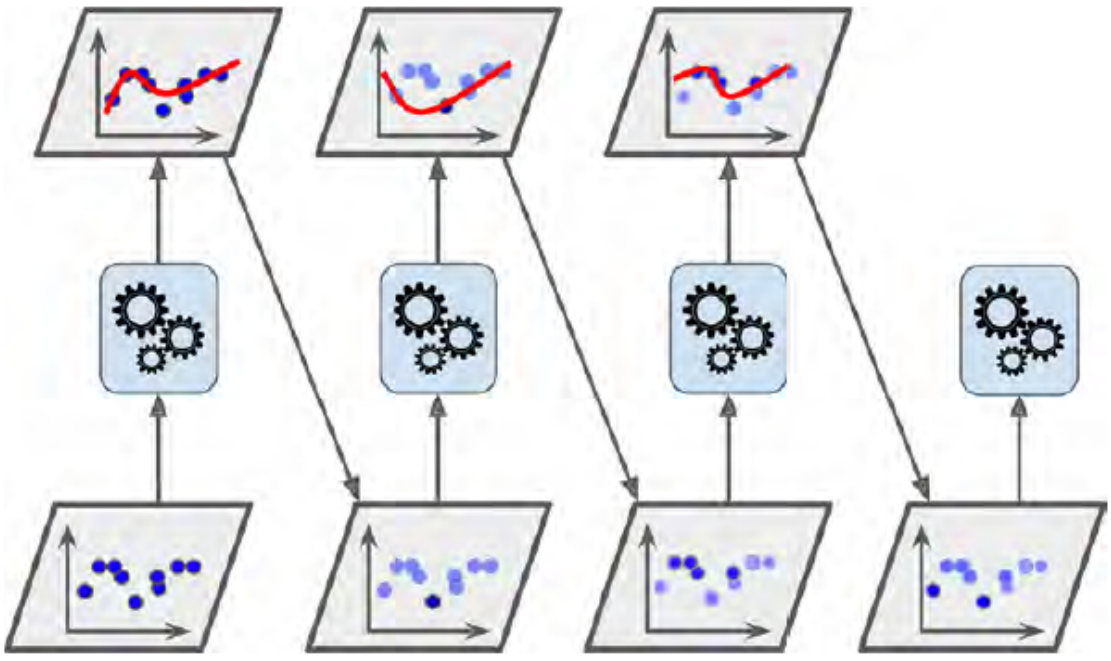
- AdaBoost (Adaptive Boosting)
- Gradient Boosting

6.1 AdaBoost

AdaBoost: Pay more attention to the training instances that the predecessor underfitted. This results in new predictors focusing more and more on the hard cases.

AdaBoost can be used for classification and regression

- AdaBoostClassifier()
- AdaBoostRegressor()



When training the AdaBoost classifier,

- algorithm first rains a base classifier (such as decision tree) and uses it to make predictions on the training set.
- then the algorithm increases the relative weight of misclassified training instances.
- then it trains a second classifier, using the updated weights and make predictions on the training set and updates weights and so on

A drawback to this sequential learning technique is that it cannot be parallelized since each predictor can only be trained after the previous predictor has been trained and evaluated. Hence it does not scale as well as bagging or pasting

- Scikit-Learn uses a multiclass version of AdaBoost called SAMME (Stagewise Additive Modeling using a Multiclass Exponential loss function).
- Scikit-Learn can use a variant of SAMME called SAMME.R (R="real") which relies on class probabilities rather than predictions and generally performs better.

In the code below, we train an AdaBoost classifier based on 200 Decision stumps (max_depth=1). Decision stump tree is composed of a single decision node plus two leaf nodes. This is the default base estimator for the AdaBoostClassifier class.

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier

ada_clf = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200,
    algorithm="SAMME.R",
    learning_rate=0.5)

ada_clf.fit(X_train, y_train)
```

Out[ ]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                   learning_rate=0.5, n_estimators=200)
```

If AdaBoost ensemble is overfitting, reduce number of estimators or more strongly regularizeing the base estimator

6.2 Gradient Boosting

Gradient Boosting: Sequentially adding predictors to an ensemble, each one corresponding its predecessor. However instead of tweaking the instance weight at every iteration like AdaBoost, Gradient Boosting tries to fit the new predictor to the residual errors made by the previous predictor

Gradient Boosting also works with regression tasks

- Gradient Tree Boosting
- Gradient Boosted Regression Trees (GBRT)

In [ ]:
```
import numpy as np

np.random.seed(42)
X = np.random.rand(100, 1) - 0.5
y = 3*X[:, 0]**2 + 0.05 * np.random.randn(100)

X_new = np.array([[0.8]])
```

In [ ]:
```
from sklearn.tree import DecisionTreeRegressor

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

Out[ ]:
```
DecisionTreeRegressor(max_depth=2)
```

We will now train a second DecisionTreeRegressor on the residual errors made by the first

predcitor

In [ ]:
```
y2 = y - tree_reg1.predict(X)

tree_reg2 = DecisionTreeRegressor(max_depth=2)
tree_reg2.fit(X, y2)
```

Out[ ]:
```
DecisionTreeRegressor(max_depth=2)
```

We will now train a third DecisionTreeRegressor on the residual errors made by the fsecondirst predcitor

In [ ]:
```
y3 = y2 - tree_reg2.predict(X)

tree_reg3 = DecisionTreeRegressor(max_depth=2)
tree_reg3.fit(X, y3)
```

Out[ ]:
```
DecisionTreeRegressor(max_depth=2)
```

With an ensemble containing three trees, we can make predictions on a new instance by adding up the predictions of all the trees

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

In [ ]:
```
y_pred
```

Out[ ]:
```
array([0.75026781])
```

In [ ]:
```
from sklearn.ensemble import GradientBoostingRegressor

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

Out[ ]:
```
GradientBoostingRegressor(learning_rate=1.0, max_depth=2, n_estimators=3)
```

6.2.1 Finding optimal number of trees

In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

X_train, X_val, y_train, y_val = train_test_split(X, y)

gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=120)
gbrt.fit(X_train, y_train)

errors = [mean_squared_error(y_val, y_pred) for y_pred in gbrt.staged_predict(X_val

bst_n_estimators = np.argmin(errors) + 1

gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimators)
gbrt_best.fit(X_train, y_train)
```

Out[ ]:
```
GradientBoostingRegressor(max_depth=2, n_estimators=85)
```

In [ ]:
```
min_error = np.min(errors)
```

In [ ]:
```
def plot_predictions(regressors, X, y, axes, label=None, style="r-", data_style="b
    x1 = np.linspace(axes[0], axes[1], 500)
    y_pred = sum(regressor.predict(x1.reshape(-1, 1)) for regressor in regressors)
    plt.plot(X[:, 0], y, data_style, label=data_label)
    plt.plot(x1, y_pred, style, linewidth=2, label=label)
    if label or data_label:
        plt.legend(loc="upper center", fontsize=16)
    plt.axis(axes)
```

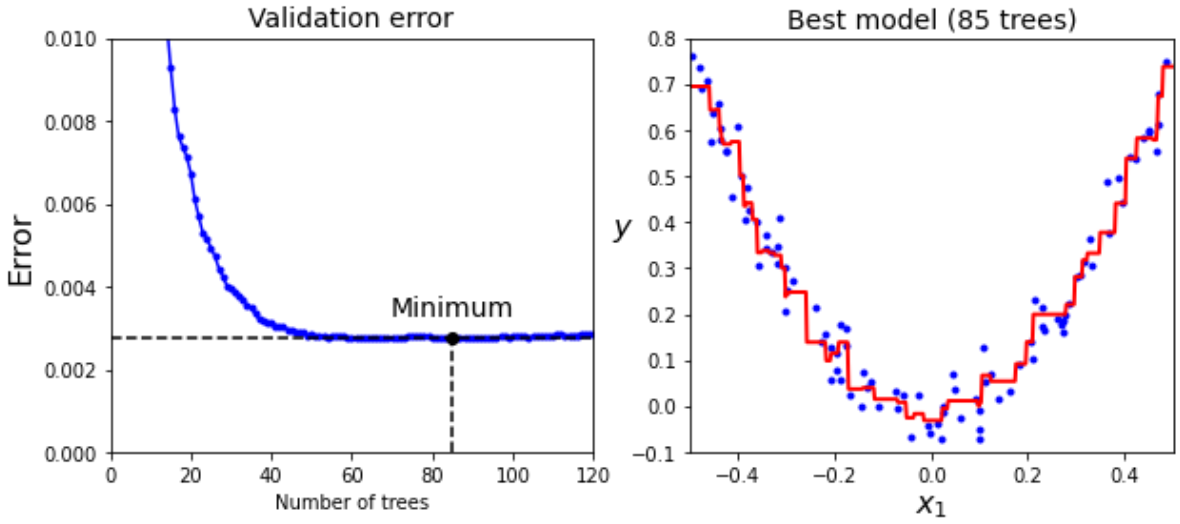
In [ ]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 4))

plt.subplot(121)
plt.plot(np.arange(1, len(errors) + 1), errors, "b.-")
plt.plot([bst_n_estimators, bst_n_estimators], [0, min_error], "k--")
plt.plot([0, 120], [min_error, min_error], "k--")
plt.plot(bst_n_estimators, min_error, "ko")
plt.text(bst_n_estimators, min_error*1.2, "Minimum", ha="center", fontsize=14)
plt.axis([0, 120, 0, 0.01])
plt.xlabel("Number of trees")
plt.ylabel("Error", fontsize=16)
plt.title("Validation error", fontsize=14)

plt.subplot(122)
plot_predictions([gbrt_best], X, y, axes=[-0.5, 0.5, -0.1, 0.8])
plt.title("Best model (%d trees)" % bst_n_estimators, fontsize=14)
plt.ylabel("$y$", fontsize=16, rotation=0)
plt.xlabel("$x_1$", fontsize=16)

plt.show()
```



Early stopping with some patience (interrupts training only after there's no improvement for 5 epochs)

In [ ]:
```
gbrt = GradientBoostingRegressor(max_depth=2, warm_start=True, random_state=42)

min_val_error = float("inf")
error_going_up = 0
for n_estimators in range(1, 120):
    gbrt.n_estimators = n_estimators
    gbrt.fit(X_train, y_train)
    y_pred = gbrt.predict(X_val)
    val_error = mean_squared_error(y_val, y_pred)
    if val_error < min_val_error:
```

```
        min_val_error = val_error
        error_going_up = 0
    else:
        error_going_up += 1
        if error_going_up == 5:
            break  # early stopping
```

In [ ]:
```
print(gbrt.n_estimators)
```

```
69
```

In [ ]:
```
print("Minimum validation MSE:", min_val_error)
```

```
Minimum validation MSE: 0.002750279033345716
```

6.2.2 Extreme Gradient Boosting (XGBoost)

- Optimizd implementation of Gradient Boosting.
- It aims to be extremely fast, scalable and portable.
- XGBoost automatically take care of early stopping

In [ ]:
```
import xgboost

xgb_reg = xgboost.XGBRegressor()
xgb_reg.fit(X_train, y_train)
y_pred = xgb_reg.predict(X_val)
```

In [ ]:
```
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)],
            early_stopping_rounds=2)

y_pred = xgb_reg.predict(X_val)
```

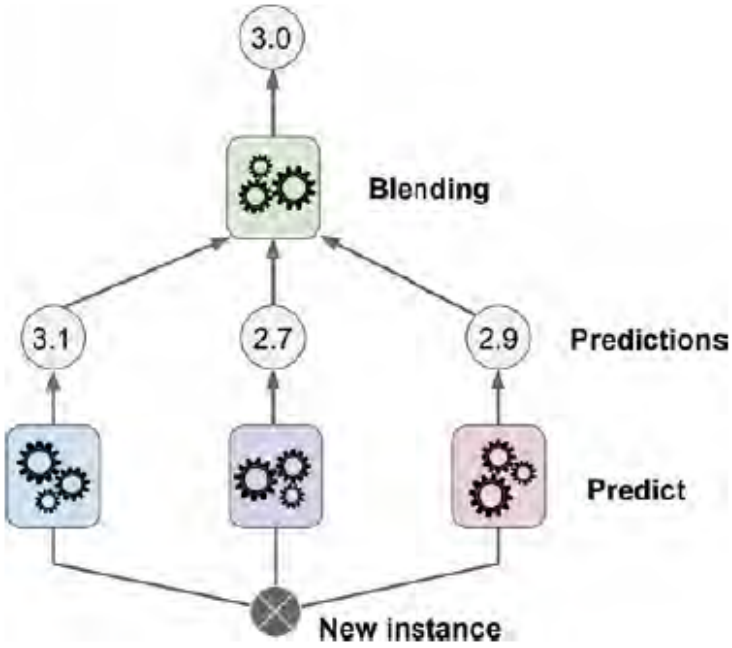
```
[0]     validation_0-rmse:0.22055
[1]     validation_0-rmse:0.16547
[2]     validation_0-rmse:0.12243
[3]     validation_0-rmse:0.10044
[4]     validation_0-rmse:0.08467
[5]     validation_0-rmse:0.07344
[6]     validation_0-rmse:0.06728
[7]     validation_0-rmse:0.06383
[8]     validation_0-rmse:0.06125
[9]     validation_0-rmse:0.05959
[10]    validation_0-rmse:0.05902
[11]    validation_0-rmse:0.05852
[12]    validation_0-rmse:0.05844
[13]    validation_0-rmse:0.05801
[14]    validation_0-rmse:0.05747
[15]    validation_0-rmse:0.05772
c:\Users\joann\Anaconda3\envs\tensorflow-gpu\lib\site-packages\xgboost\sklearn.py:
793: UserWarning: `early_stopping_rounds` in `fit` method is deprecated for better
compatibility with scikit-learn, use `early_stopping_rounds` in constructor or`set
_params` instead.
  warnings.warn(
```

7. Stacking (Stacked Generalization)

- Based on the idea that instead of using trivial functions such as hard voting to

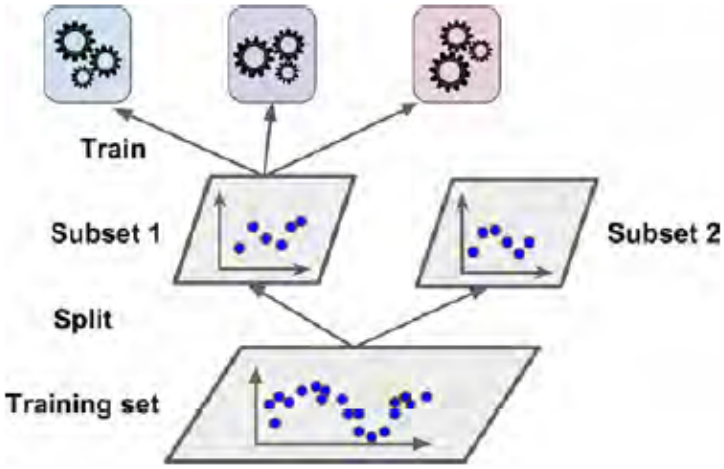
aggregate the predictions of all predictors in an ensemble, we train a model to perform this aggregation.

- Each of the predictors predicts a different value (3.1, 2.7, 2.9), and the final predcitor (blender or metal learner) takes these predictions and as inputs and makes the final prediction (3.0)



To train the blender, we use hold-out set.

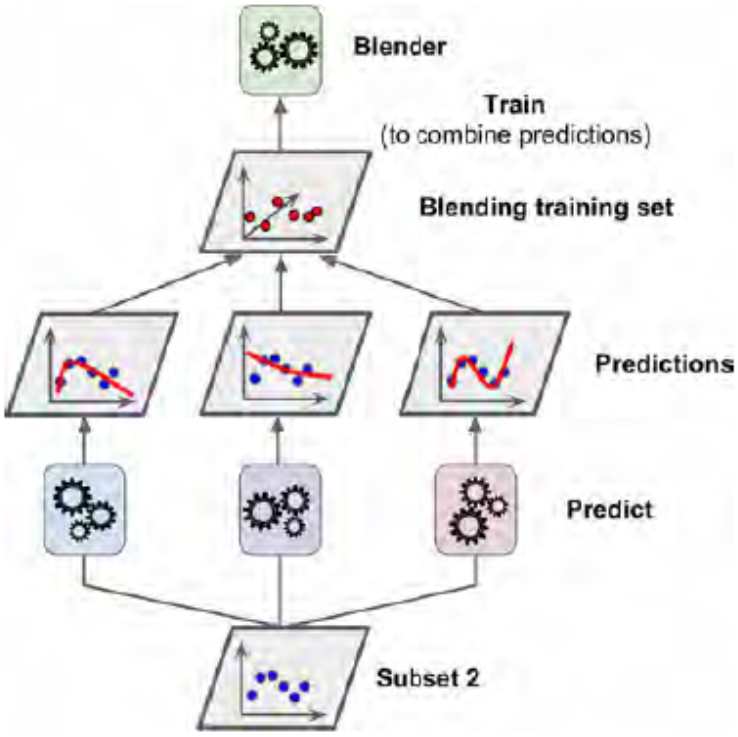
- First we split training data into two subsets. First subset is used to train the predictors in the first layer.



- We use second subset as the hold-out set where we evaluate on.
- We have three predicted values.
- We can create a new training set using these predicted values as input features and keeping the target values.
- The blender is trained on this new training set so it leans to predict the target value, given the first layer's predictions

7.1 Steps to create meta-learner (blender)

- Split the training data into two folds.
- Choose L weak learners and fit them to the data of the first fold
- For each of the L weak learners, make predictions for observations in the second fold
- Fit the meta-model on the second fold using the predictions made by the weak learners as inputs



7.2 Stacking V.S Boosting and Bagging

- Stacking often consideres heterogeneous weak learenrs (different learning algorithms combined) whereas bagging and boosting consider mainly homogenous weak learners.
- Stacking learns to combine the base models into a meta-model whereas bagging and boosting combine weak learners following deterministic algorithms

Takeaways

- Ensemble learning is a machine learning paradigm where multiple models (weak learners or base models) are trained to solve the same problem and combined to get better performances
- In bagging, several instance of the same base model are trained in parallel (independently from each other) on different bootstrap samples and then aggregated in some kind of "averaging" process. This allows us to obtain an ensemble model with a lower variance than its components.
- In boosting, several instance of the same base model are trained sequentially such that at each iteration, the way to train the current weak learner depends on the previou

weak learners. This allows us to obtain an ensemble model with a lower bias than its components.

- In stacking, different weak learners are fitted independently from each other and a metal-model is trained on top fo that to predict outputs based ont he outputs returned by the base models.

In [ ]: