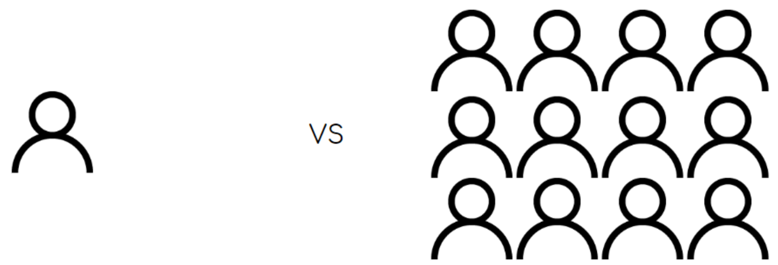
CSY2082 Introduction to Artificial Intelligence

Ensemble Learning and Random Forests

Ensemble Learning

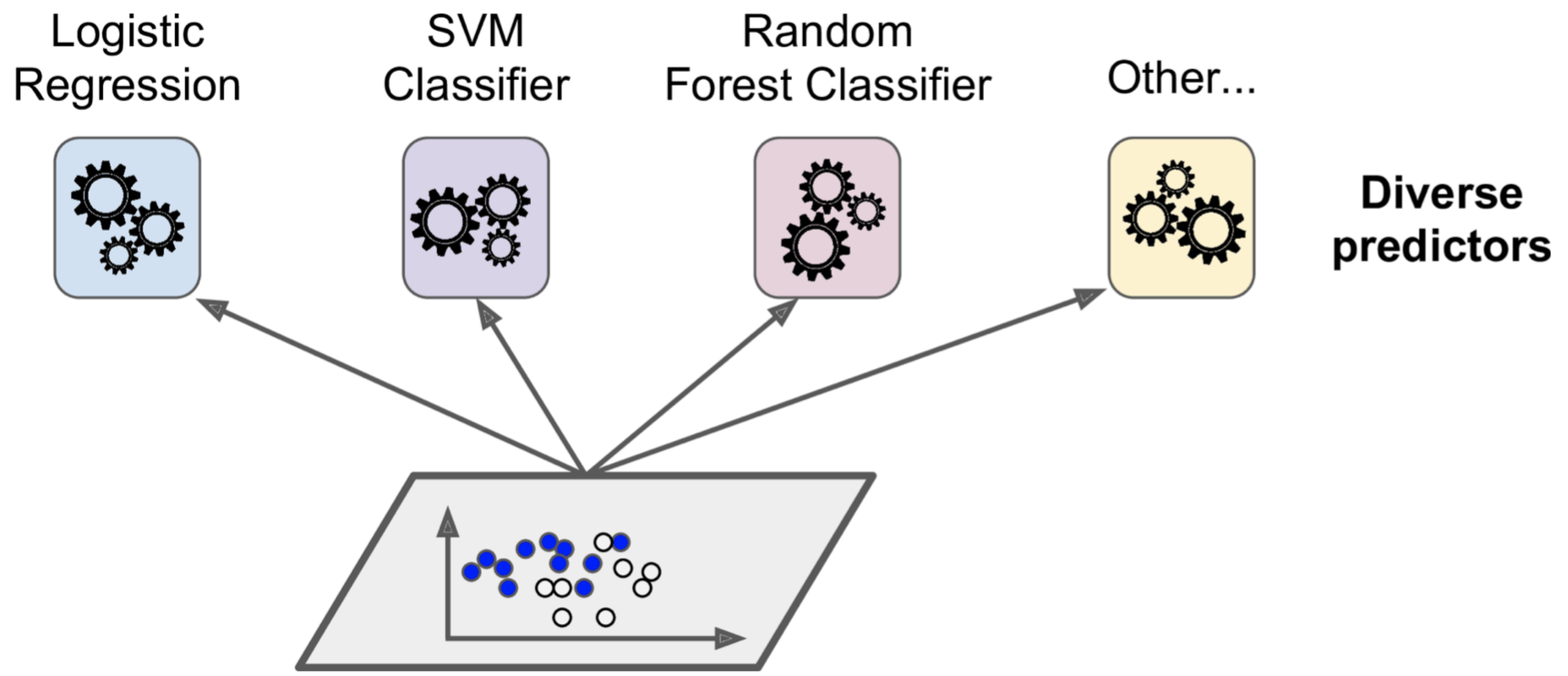
Ensemble Learning



- Suppose you ask a complex question to thousands of random people, then **aggregate** their answers. In many cases you will find that this aggregated answer is better than an expert’s answer. This is called the *wisdom of the crowd*.
- Similarly, if you aggregate the 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**
- You will often use Ensemble methods near the end of a project, once you have already built a few good predictors, to **combine** them into an even better predictor. In fact, the winning solutions in Machine Learning competitions often involve several Ensemble methods.

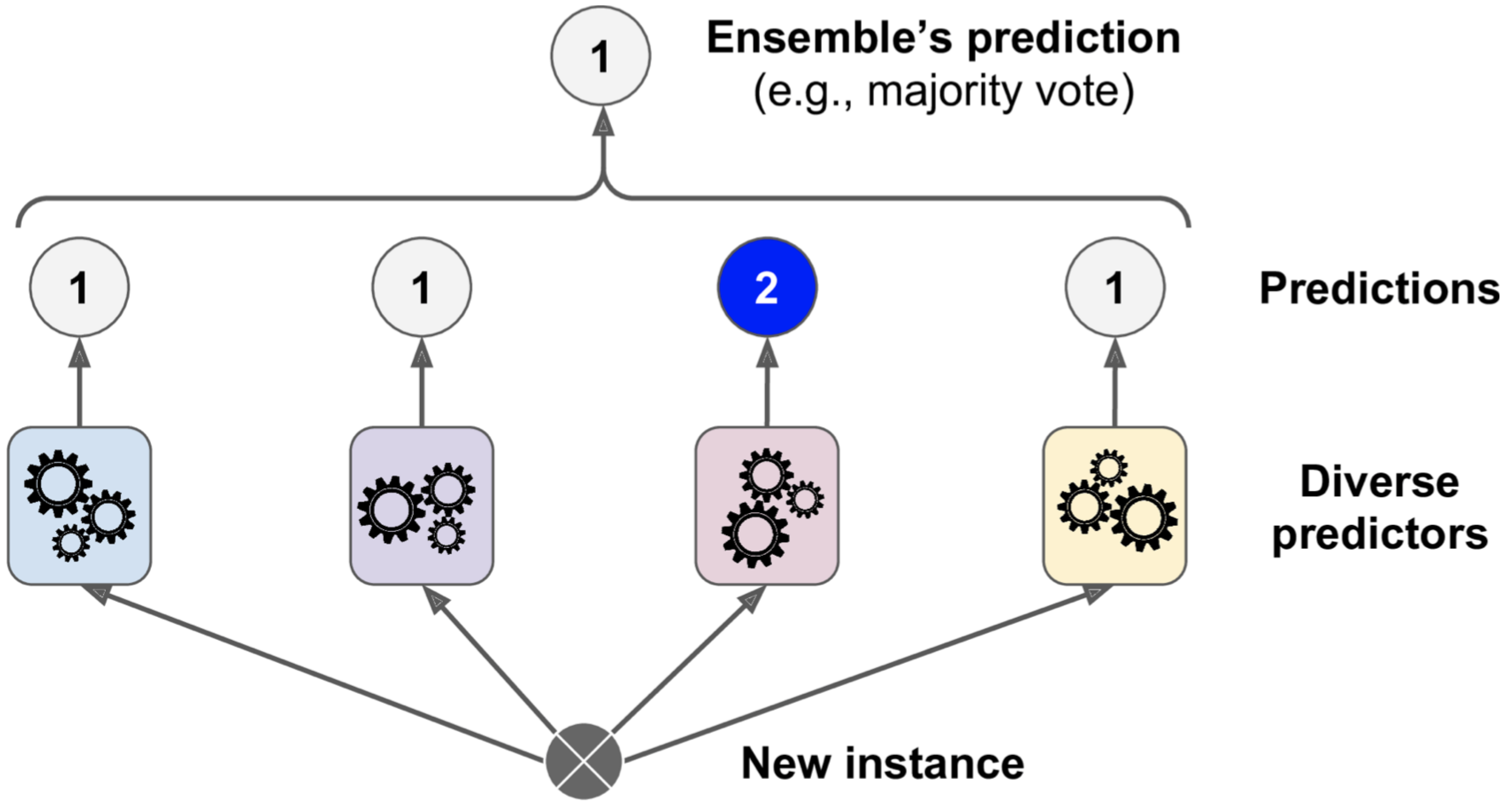
Voting Classifiers

- Suppose you have trained a few classifiers, each one achieving about 80% accuracy.



Voting Classifiers

- A very simple way to create an even better classifier is to aggregate the predictions of each classifier and predict the class that gets the most votes. This majority-vote classifier is called a **hard voting** classifier
- A few weak models -> strong model.



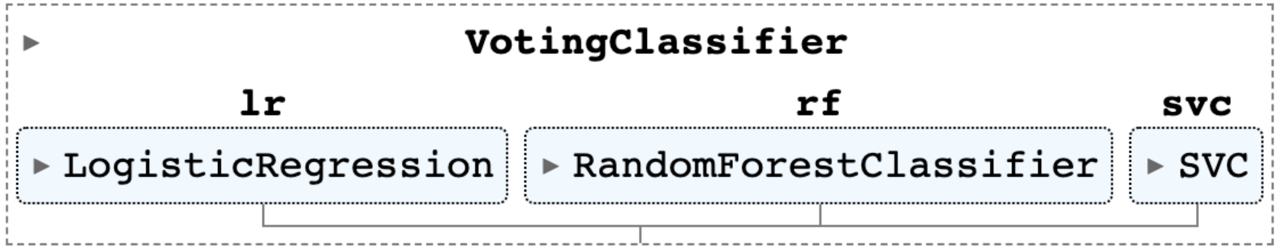
Voting Classifiers (hard) – make_moons data

```python
from sklearn.datasets import make_moons
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, 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)

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)
```

Get a diverse set of classifiers is to use very different training algorithms.
(delete "random_state" in your own code)



LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.904

Voting Classifiers (soft)

- If all classifiers are able to estimate class probabilities (i.e., they have a pre dict_proba() method), then you can tell Scikit-Learn to predict the class with the highest class probability, **averaged over all the individual classifiers**. This is called *soft voting*.

```
voting_clf.voting = "soft"
voting_clf.named_estimators["svc"].probability = True
voting_clf.fit(X_train, y_train)
voting_clf.score(X_test, y_test)
```

LogisticRegression 0.864
RandomForestClassifier 0.896
SVC 0.888
VotingClassifier 0.92

Random Forests

Random Forests

- A Random Forest is an ensemble of Decision Trees, generally trained via the bagging method (each tree picks a random subset for training). Each tree also picks a random subset of features for training (normally $\sqrt{n}$).
- The following code trains a Random Forest classifier with 500 trees (each limited to maximum 16 nodes).

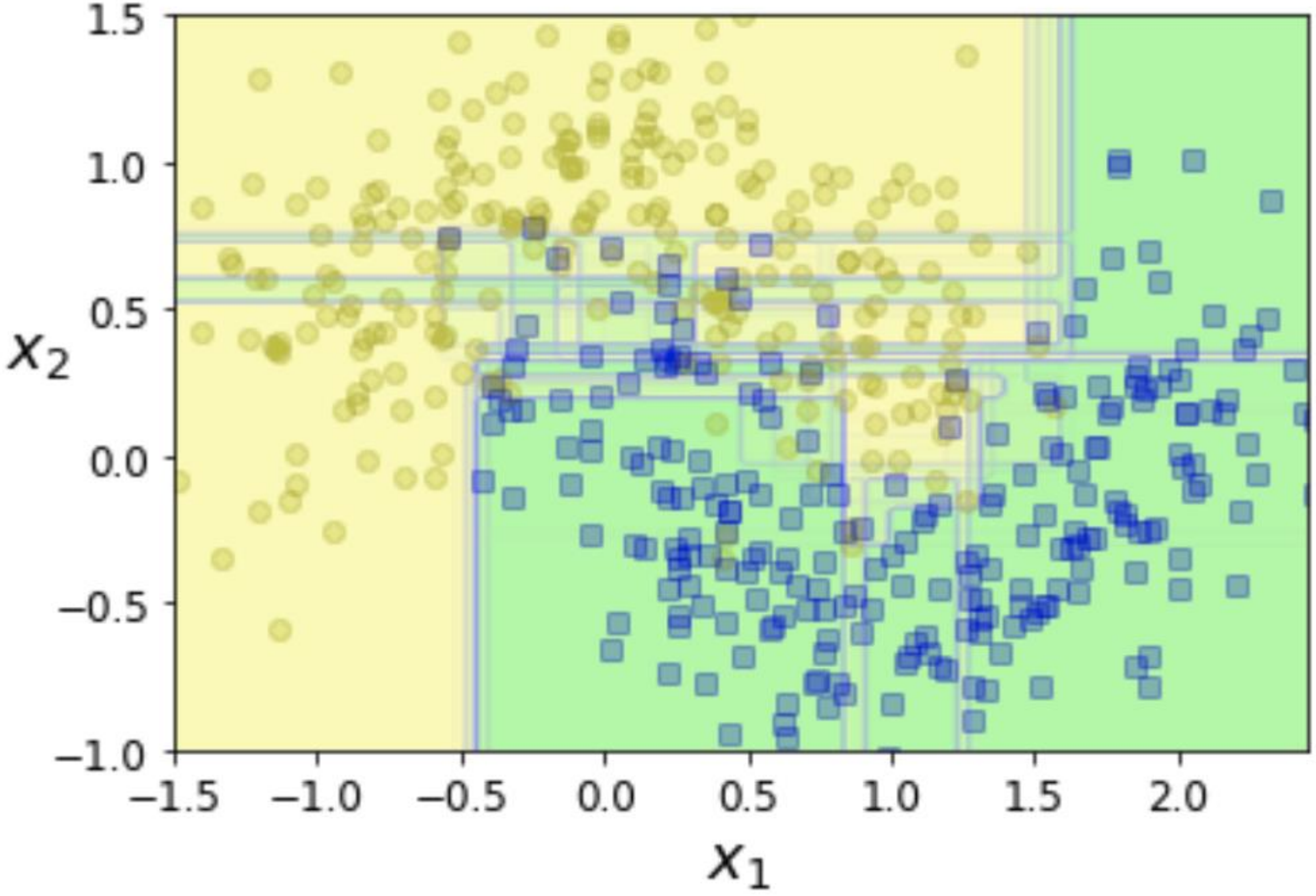
```
from sklearn.ensemble import RandomForestClassifier

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)
```

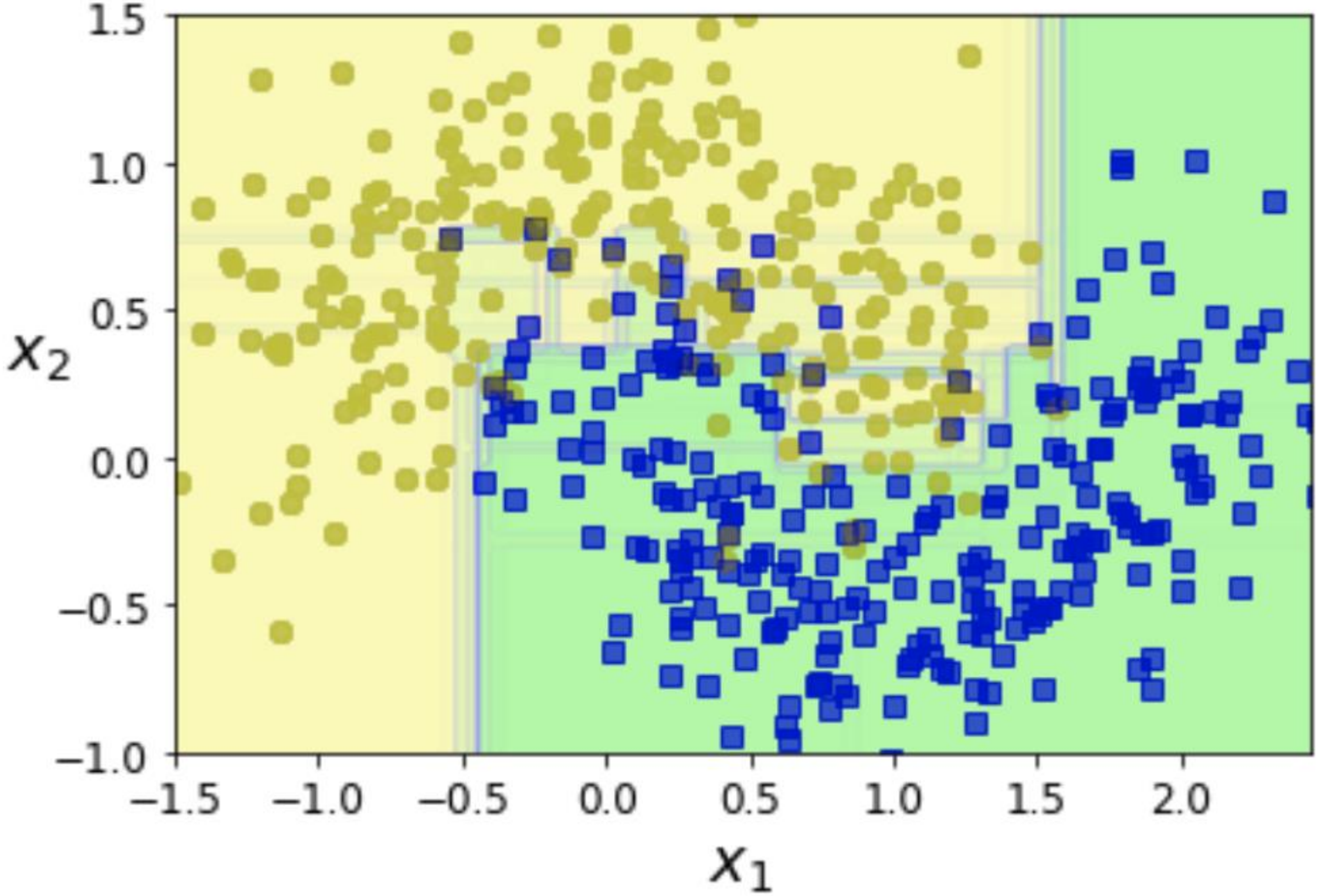
Random Forests

- 15 trees



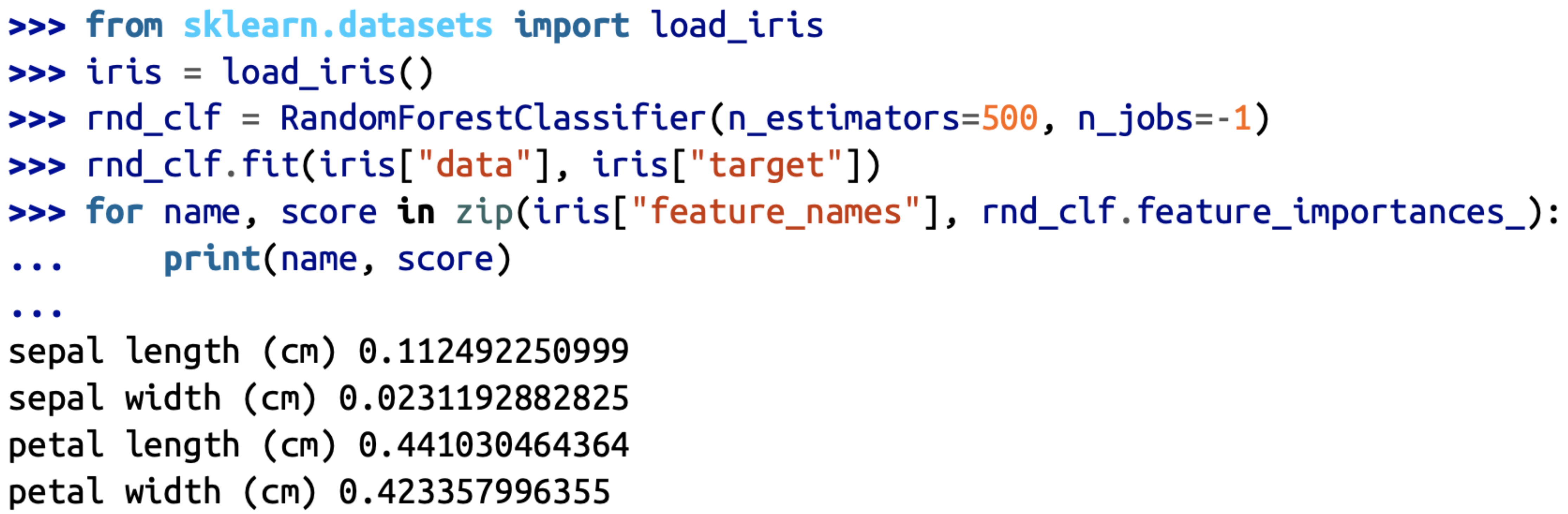
Random Forests

- 50 trees



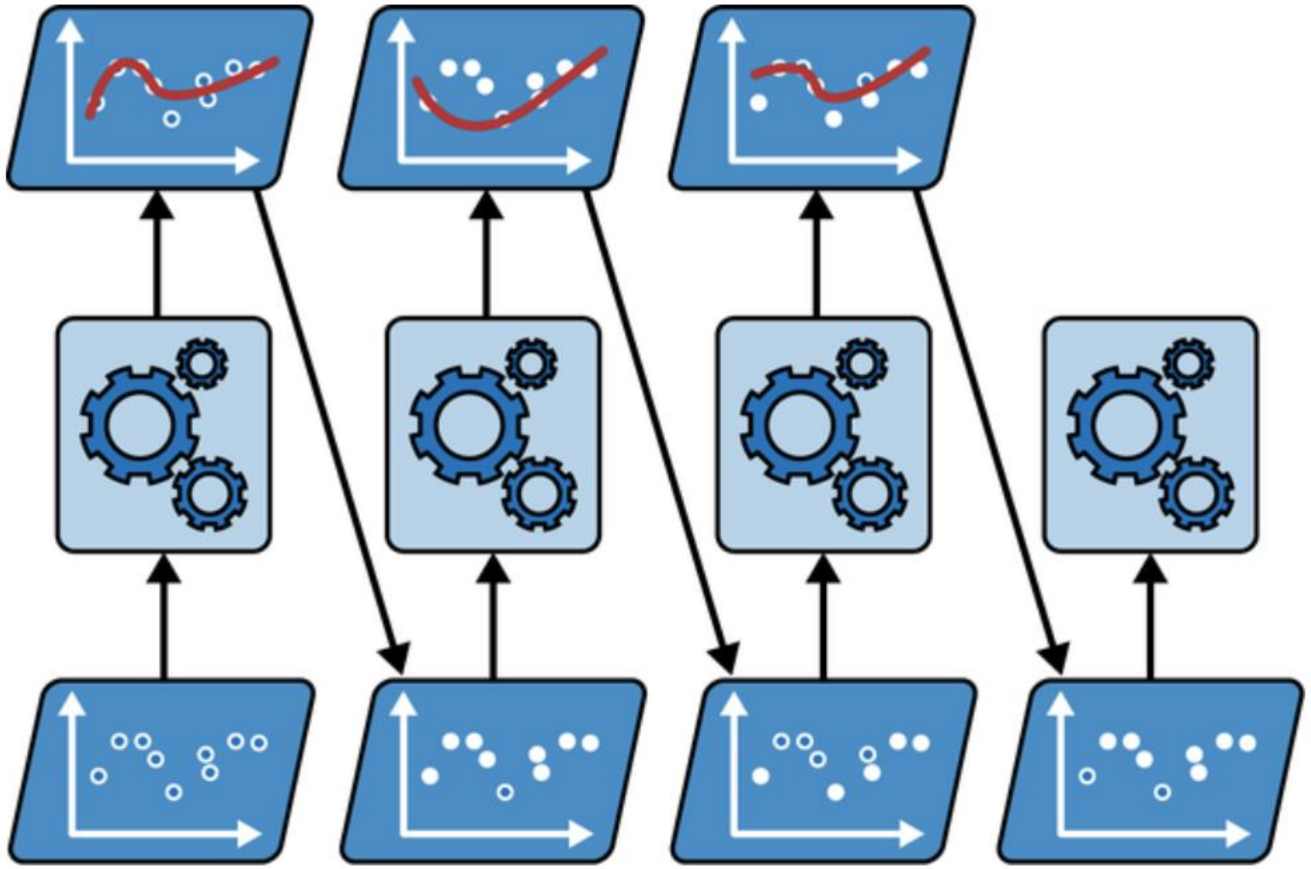
Feature Importance

- A great quality of Random Forests is that they make it easy to measure the relative importance of each feature. Scikit-Learn measures a feature’s importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest).
- Scikit-Learn computes this score automatically for each feature after training, then it scales the results so that the sum of all importances is equal to 1. You can access the result using the *feature_importances_* variable.

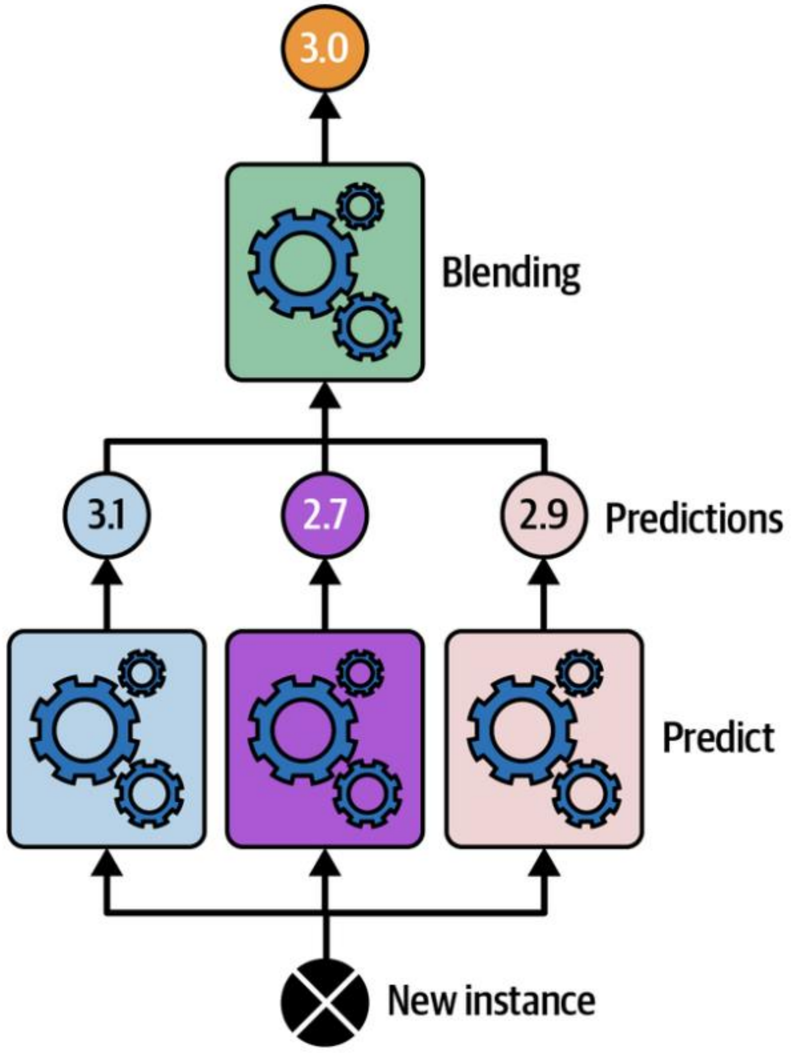


```
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> rnd_clf = RandomForestClassifier(n_estimators=500, n_jobs=-1)
>>> rnd_clf.fit(iris["data"], iris["target"])
>>> for name, score in zip(iris["feature_names"], rnd_clf.feature_importances_):
...     print(name, score)
...
sepal length (cm) 0.112492250999
sepal width (cm) 0.0231192882825
petal length (cm) 0.441030464364
petal width (cm) 0.423357996355
```

Boosting


AdaBoost sequential training with instance weight updates

Stacking



aggregating predictions using a blending predictor

Exercise

- Examine the classifiers listed on the following webpage and add some of these classifiers to your voting classifier to improve the overall performance.

https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html