eters that need to be tuned for best results. You can find all of these parameters and their definitions in the user guide. When starting to work with MLPs, we recommend sticking to 'adam' and 'l-bfgs'.

> **fit Resets a Model**
>
> An important property of scikit-learn models is that calling fit will always reset everything a model previously learned. So if you build a model on one dataset, and then call fit again on a different dataset, the model will "forget" everything it learned from the first dataset. You can call fit as often as you like on a model, and the outcome will be the same as calling fit on a "new" model.

Uncertainty Estimates from Classifiers

Another useful part of the scikit-learn interface that we haven't talked about yet is the ability of classifiers to provide uncertainty estimates of predictions. Often, you are not only interested in which class a classifier predicts for a certain test point, but also how certain it is that this is the right class. In practice, different kinds of mistakes lead to very different outcomes in real-world applications. Imagine a medical application testing for cancer. Making a false positive prediction might lead to a patient undergoing additional tests, while a false negative prediction might lead to a serious disease not being treated. We will go into this topic in more detail in Chapter 6.

There are two different functions in scikit-learn that can be used to obtain uncertainty estimates from classifiers: decision_function and predict_proba. Most (but not all) classifiers have at least one of them, and many classifiers have both. Let's look at what these two functions do on a synthetic two-dimensional dataset, when building a GradientBoostingClassifier classifier, which has both a decision_function and a predict_proba method:

In[105]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.datasets import make_blobs, make_circles
X, y = make_circles(noise=0.25, factor=0.5, random_state=1)

# we rename the classes "blue" and "red" for illustration purposes
y_named = np.array(["blue", "red"])[y]

# we can call train_test_split with arbitrarily many arrays;
# all will be split in a consistent manner
X_train, X_test, y_train_named, y_test_named, y_train, y_test = \
    train_test_split(X, y_named, y, random_state=0)

# build the gradient boosting model
gbrt = GradientBoostingClassifier(random_state=0)
gbrt.fit(X_train, y_train_named)
```

The Decision Function

In the binary classification case, the return value of decision_function is of shape (n_samples,), and it returns one floating-point number for each sample:

In[106]:

```
print("X_test.shape: {}".format(X_test.shape))
print("Decision function shape: {}".format(
    gbrt.decision_function(X_test).shape))
```

Out[106]:

```
X_test.shape: (25, 2)
Decision function shape: (25,)
```

This value encodes how strongly the model believes a data point to belong to the "positive" class, in this case class 1. Positive values indicate a preference for the positive class, and negative values indicate a preference for the "negative" (other) class:

In[107]:

```
# show the first few entries of decision_function
print("Decision function:\n{}".format(gbrt.decision_function(X_test)[:6]))
```

Out[107]:

```
Decision function:
[ 4.136 -1.683 -3.951 -3.626  4.29   3.662]
```

We can recover the prediction by looking only at the sign of the decision function:

In[108]:

```
print("Thresholded decision function:\n{}".format(
    gbrt.decision_function(X_test) > 0))
print("Predictions:\n{}".format(gbrt.predict(X_test)))
```

Out[108]:

```
Thresholded decision function:
[ True False False False  True  True False  True  True  True False  True
  True False  True False False False  True  True  True  True  True False
  False]
Predictions:
['red' 'blue' 'blue' 'blue' 'red' 'red' 'blue' 'red' 'red' 'red' 'blue'
 'red' 'red' 'blue' 'red' 'blue' 'blue' 'blue' 'red' 'red' 'red' 'red'
 'red' 'blue' 'blue']
```

For binary classification, the "negative" class is always the first entry of the classes_ attribute, and the "positive" class is the second entry of classes_. So if you want to fully recover the output of predict, you need to make use of the classes_ attribute: