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:

In[109]:

```
# make the boolean True/False into 0 and 1
greater_zero = (gbrt.decision_function(X_test) > 0).astype(int)
# use 0 and 1 as indices into classes_
pred = gbrt.classes_[greater_zero]
# pred is the same as the output of gbrt.predict
print("pred is equal to predictions: {}".format(
    np.all(pred == gbrt.predict(X_test))))
```

Out[109]:

```
pred is equal to predictions: True
```

The range of `decision_function` can be arbitrary, and depends on the data and the model parameters:

In[110]:

```
decision_function = gbrt.decision_function(X_test)
print("Decision function minimum: {:.2f} maximum: {:.2f}".format(
    np.min(decision_function), np.max(decision_function)))
```

Out[110]:

```
Decision function minimum: -7.69 maximum: 4.29
```

This arbitrary scaling makes the output of `decision_function` often hard to interpret.

In the following example we plot the `decision_function` for all points in the 2D plane using a color coding, next to a visualization of the decision boundary, as we saw earlier. We show training points as circles and test data as triangles (Figure 2-55):

In[111]:

```
fig, axes = plt.subplots(1, 2, figsize=(13, 5))
mglearn.tools.plot_2d_separator(gbrt, X, ax=axes[0], alpha=.4,
                                fill=True, cm=mglearn.cm2)
scores_image = mglearn.tools.plot_2d_scores(gbrt, X, ax=axes[1],
                                            alpha=.4, cm=mglearn.ReBl)

for ax in axes:
    # plot training and test points
    mglearn.discrete_scatter(X_test[:, 0], X_test[:, 1], y_test,
                             markers='^', ax=ax)
    mglearn.discrete_scatter(X_train[:, 0], X_train[:, 1], y_train,
                             markers='o', ax=ax)
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
cbar = plt.colorbar(scores_image, ax=axes.tolist())
axes[0].legend(["Test class 0", "Test class 1", "Train class 0",
                "Train class 1"], ncol=4, loc=(.1, 1.1))
```

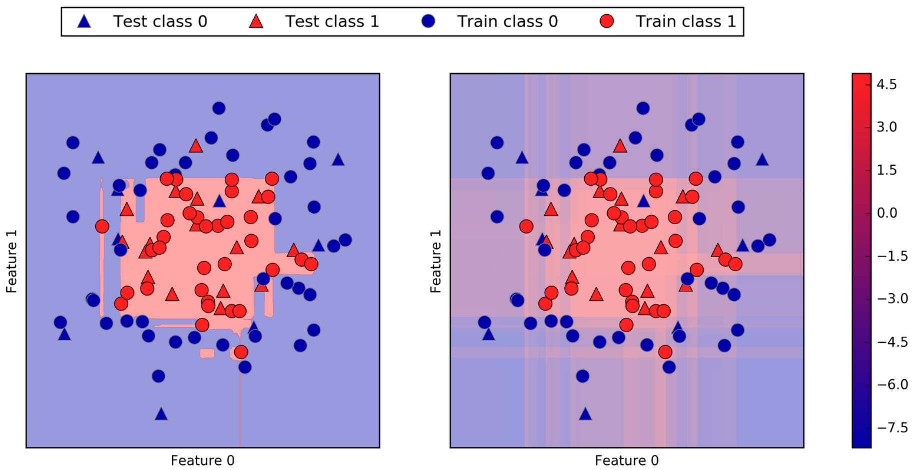

Figure 2-55. Decision boundary (left) and decision function (right) for a gradient boosting model on a two-dimensional toy dataset

Encoding not only the predicted outcome but also how certain the classifier is provides additional information. However, in this visualization, it is hard to make out the boundary between the two classes.

Predicting Probabilities

The output of `predict_proba` is a probability for each class, and is often more easily understood than the output of `decision_function`. It is always of shape `(n_samples, 2)` for binary classification:

In[112]:

```
print("Shape of probabilities: {}".format(gbrt.predict_proba(X_test).shape))
```

Out[112]:

```
Shape of probabilities: (25, 2)
```

The first entry in each row is the estimated probability of the first class, and the second entry is the estimated probability of the second class. Because it is a probability, the output of `predict_proba` is always between 0 and 1, and the sum of the entries for both classes is always 1:

In[113]:

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