< Implementing AdaBoost (10.04-Implementing-AdaBoost.ipynb) | Contents (../README.md) | Selecting the Right Model with Hyper-Parameter Tuning (11.00-Selecting-the-Right-Model-with-Hyper-Parameter-Tuning.ipynb) >

Combining Different Models Into a Voting Classifier

So far, we saw how to combine different instances of the same classifier or regressor into an ensemble. In this chapter, we are going to take this idea a step further and combine conceptually different classifiers into what is known as a **voting classifier**.

The idea behind voting classifiers is that the individual learners in the ensemble don't necessarily need to be of the same type. After all, no matter how the individual classifiers arrived at their prediction, in the end, we are going to apply a decision rule that integrates all the votes of the individual classifiers. This is also known as a **voting scheme**.

Two different voting schemes are common among voting classifiers:

- In **hard voting** (also known as **majority voting**), every individual classifier votes for a class, and the majority wins. In statistical terms, the predicted target label of the ensemble is the mode of the distribution of individually predicted labels.
- In **soft voting**, every individual classifier provides a probability value that a specific data point belongs to a particular target class. The predictions are weighted by the classifier's importance and summed up. Then the target label with the greatest sum of weighted probabilities wins the vote.

You can find an example of how these voting schemes work in practice in the book.

Implementing a Voting Classifier

Let's look at a simple example of a voting classifier that combines three different algorithms:

- A logistic regression classifier from Chapter 3 (03.00-First-Steps-in-Supervised-Learning.ipynb), *First Steps in Supervised Learning*
- A Gaussian naive Bayes classifier from Chapter 7 (07.00-Implementing-a-Spam-Filter-with-Bayesian-Learning.ipynb), *Implementing a Spam Filter with Bayesian Learning*
- A random forest classifier from this chapter

We can combine these three algorithms into a voting classifier and apply it to the breast cancer dataset with the following steps.

Load the dataset, and split it into training and test sets:

In [1]:
```
from sklearn.datasets import load_breast_cancer
iris = load_breast_cancer()
X = iris.data
y = iris.target
```

In [2]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=13
)
```

Instantiate the individual classifiers:

In [3]:
```
from sklearn.linear_model import LogisticRegression
model1 = LogisticRegression(random_state=13)
```

In [4]:
```
from sklearn.naive_bayes import GaussianNB
model2 = GaussianNB()
```

In [5]:
```
from sklearn.ensemble import RandomForestClassifier
model3 = RandomForestClassifier(random_state=13)
```

Assign the individual classifiers to the voting ensemble. Here, we need to pass a list of tuples (`estimators`), where every tuple consists of the name of the classifier (a string of our choosing) and the model object. The voting scheme can be either `voting='hard'` or `voting='soft'` :

In [6]:
```
from sklearn.ensemble import VotingClassifier
vote = VotingClassifier(estimators=[('lr', model1),
                                    ('gnb', model2),
                                    ('rfc', model3)],
                        voting='hard')
```

Fit the ensemble to the training data and score it on the test data:

In [7]:
```
vote.fit(X_train, y_train)
vote.score(X_test, y_test)
```

Out[7]:
```
0.95104895104895104
```

In order to convince us that 95.1% is a great accuracy score, we can compare the ensemble's performance to the theoretical performance of each individual classifier. We do this by fitting the individual classifiers to the data. Then we will see that the logistic regression model achieves 94.4% accuracy on its own:

In [8]:
```
model1.fit(X_train, y_train)
model1.score(X_test, y_test)
```

Out[8]:
```
0.94405594405594406
```

Similarly, the naive Bayes classifier achieves 93.0% accuracy:

In [9]:
```
model2.fit(X_train, y_train)
model2.score(X_test, y_test)
```

Out[9]:
```
0.93006993006993011
```

Last but not least, the random forest classifier also achieved 94.4% accuracy:

In [10]:
```
model3.fit(X_train, y_train)
model3.score(X_test, y_test)
```

Out[10]:
```
0.94405594405594406
```

All in all, we were just able to gain a good percent in performance by combining three unrelated classifiers into an ensemble. Each of these classifiers might have made different mistakes on the training set, but that's OK because on average, we need just two out of three classifiers to be correct.

< Implementing AdaBoost (10.04-Implementing-AdaBoost.ipynb) | Contents (../README.md) | Selecting the Right Model with Hyper-Parameter Tuning (11.00-Selecting-the-Right-Model-with-Hyper-Parameter-Tuning.ipynb) >