

In[120]:

```
logreg = LogisticRegression()

# represent each target by its class name in the iris dataset
named_target = iris.target_names[y_train]
logreg.fit(X_train, named_target)
print("unique classes in training data: {}".format(logreg.classes_))
print("predictions: {}".format(logreg.predict(X_test)[:10]))
argmax_dec_func = np.argmax(logreg.decision_function(X_test), axis=1)
print("argmax of decision function: {}".format(argmax_dec_func[:10]))
print("argmax combined with classes: {}".format(
    logreg.classes_[argmax_dec_func[:10]]))
```

Out[120]:

```
unique classes in training data: ['setosa' 'versicolor' 'virginica']
predictions: ['versicolor' 'setosa' 'virginica' 'versicolor' 'versicolor'
 'setosa' 'versicolor' 'virginica' 'versicolor' 'versicolor']
argmax of decision function: [1 0 2 1 1 0 1 2 1 1]
argmax combined with classes_: ['versicolor' 'setosa' 'virginica' 'versicolor'
 'versicolor' 'setosa' 'versicolor' 'virginica' 'versicolor' 'versicolor']
```

Summary and Outlook

We started this chapter with a discussion of model complexity, then discussed *generalization*, or learning a model that is able to perform well on new, previously unseen data. This led us to the concepts of underfitting, which describes a model that cannot capture the variations present in the training data, and overfitting, which describes a model that focuses too much on the training data and is not able to generalize to new data very well.

We then discussed a wide array of machine learning models for classification and regression, what their advantages and disadvantages are, and how to control model complexity for each of them. We saw that for many of the algorithms, setting the right parameters is important for good performance. Some of the algorithms are also sensitive to how we represent the input data, and in particular to how the features are scaled. Therefore, blindly applying an algorithm to a dataset without understanding the assumptions the model makes and the meanings of the parameter settings will rarely lead to an accurate model.

This chapter contains a lot of information about the algorithms, and it is not necessary for you to remember all of these details for the following chapters. However, some knowledge of the models described here—and which to use in a specific situation—is important for successfully applying machine learning in practice. Here is a quick summary of when to use each model:

Nearest neighbors

For small datasets, good as a baseline, easy to explain.

Linear models

Go-to as a first algorithm to try, good for very large datasets, good for very high-dimensional data.

Naive Bayes

Only for classification. Even faster than linear models, good for very large datasets and high-dimensional data. Often less accurate than linear models.

Decision trees

Very fast, don't need scaling of the data, can be visualized and easily explained.

Random forests

Nearly always perform better than a single decision tree, very robust and powerful. Don't need scaling of data. Not good for very high-dimensional sparse data.

Gradient boosted decision trees

Often slightly more accurate than random forests. Slower to train but faster to predict than random forests, and smaller in memory. Need more parameter tuning than random forests.

Support vector machines

Powerful for medium-sized datasets of features with similar meaning. Require scaling of data, sensitive to parameters.

Neural networks

Can build very complex models, particularly for large datasets. Sensitive to scaling of the data and to the choice of parameters. Large models need a long time to train.

When working with a new dataset, it is in general a good idea to start with a simple model, such as a linear model or a naive Bayes or nearest neighbors classifier, and see how far you can get. After understanding more about the data, you can consider moving to an algorithm that can build more complex models, such as random forests, gradient boosted decision trees, SVMs, or neural networks.

You should now be in a position where you have some idea of how to apply, tune, and analyze the models we discussed here. In this chapter, we focused on the binary classification case, as this is usually easiest to understand. Most of the algorithms presented have classification and regression variants, however, and all of the classification algorithms support both binary and multiclass classification. Try applying any of these algorithms to the built-in datasets in `scikit-learn`, like the `boston_housing` or `diabetes` datasets for regression, or the `digits` dataset for multiclass classification. Playing around with the algorithms on different datasets will give you a better feel for

how long they need to train, how easy it is to analyze the models, and how sensitive they are to the representation of the data.

While we analyzed the consequences of different parameter settings for the algorithms we investigated, building a model that actually generalizes well to new data in production is a bit trickier than that. We will see how to properly adjust parameters and how to find good parameters automatically in [Chapter 6](#).

First, though, we will dive in more detail into unsupervised learning and preprocessing in the next chapter.

Unsupervised Learning and Preprocessing

The second family of machine learning algorithms that we will discuss is unsupervised learning algorithms. Unsupervised learning subsumes all kinds of machine learning where there is no known output, no teacher to instruct the learning algorithm. In unsupervised learning, the learning algorithm is just shown the input data and asked to extract knowledge from this data.

Types of Unsupervised Learning

We will look into two kinds of unsupervised learning in this chapter: transformations of the dataset and clustering.

Unsupervised transformations of a dataset are algorithms that create a new representation of the data which might be easier for humans or other machine learning algorithms to understand compared to the original representation of the data. A common application of unsupervised transformations is dimensionality reduction, which takes a high-dimensional representation of the data, consisting of many features, and finds a new way to represent this data that summarizes the essential characteristics with fewer features. A common application for dimensionality reduction is reduction to two dimensions for visualization purposes.

Another application for unsupervised transformations is finding the parts or components that “make up” the data. An example of this is topic extraction on collections of text documents. Here, the task is to find the unknown topics that are talked about in each document, and to learn what topics appear in each document. This can be useful for tracking the discussion of themes like elections, gun control, or pop stars on social media.

Clustering algorithms, on the other hand, partition data into distinct groups of similar items. Consider the example of uploading photos to a social media site. To allow you