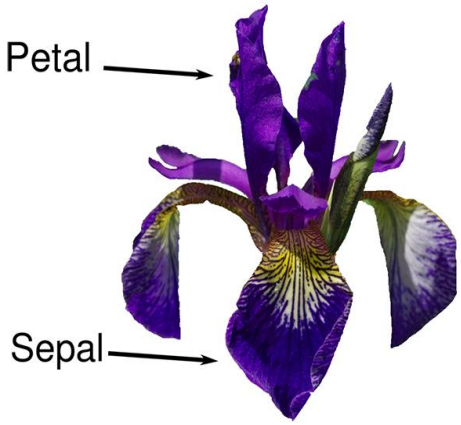
Classifying Iris Species

Problem Statement:

In this section, we will go through a simple machine learning application and create our first model. In the process, we will introduce some core concepts and terms. Let's assume that a hobby botanist is interested in distinguishing the species of some iris flowers that she has found. She has collected some measurements associated with each iris: the length and width of the petals and the length and width of the sepals, all measured in centimeters. She also has the measurements of some irises that have been previously identified by an expert botanist as belonging to the species setosa, versicolor, or virginica. For these measurements, she can be certain of which species each iris belongs to. Let's assume that these are the only species our hobby botanist will encounter in the wild. Our goal is to build a machine learning model that can learn from the measurements of these irises whose species is known, so that we can predict the species for a new iris.



Points to finalise the model:

- Because we have measurements for which we know the correct species of iris, this is a supervised learning problem.
- In this problem, we want to predict one of several options (the species of iris). This is an example of a classification problem.
- The possible outputs (different species of irises) are called classes. Every iris in the dataset belongs to one of three classes, so this problem is a three-class classification problem.
- The desired output for a single data point (an iris) is the species of this flower. For a particular data point, the species it belongs to is called its label.

Meet the Data

The data we will use for this example is the Iris dataset, a classical dataset in machine learning and statistics. It is included in scikit-learn in the datasets module. We can load it by calling the load_iris function.

Keys & Explanations:

- **DESCR** is a short description of the dataset.
- **target_names** containing the species of flower that we want to predict.
- **feature_names** giving the description of each feature.
- The data itself is contained in the **target and data** fields. data contains the numeric measurements of sepal length, sepal width, petal length, and petal width in a NumPy array.

Measuring Success:

Training and Testing Data We want to build a machine learning model from this data that can predict the species of iris for a new set of measurements. But before we can apply our model to new measurements, we need to know whether it actually works—that is, whether we should trust its predictions.

Unfortunately, we cannot use the data we used to build the model to evaluate it. This is because our model can always simply remember the whole training set, and will therefore always predict the correct label for any point in the training set. This "remembering" does not indicate to us whether our model will generalize well (in other words, whether it will also perform well on new data).

To assess the model's performance, we show it new data (data that it hasn't seen before) for which we have labels. This is usually done by splitting the labeled data we have collected (here, our 150 flower measurements) into two parts. One part of the data is used to build our machine learning model, and is called the training data or training set. The rest of the data will be used to assess how well the model works; this is called the test data, test set, or hold-out set.

scikit-learn contains a function that shuffles the dataset and splits it for you: the train_test_split function. This function extracts 75% of the rows in the data as the training set, together with the corresponding labels for this data. The remaining 25% of the data, together with the remaining labels, is declared as the test set. Deciding how much data you want to put into the training and the test set respectively is somewhat arbitrary, but using a test set containing 25% of the data is a good rule of thumb.

In scikit-learn, data is usually denoted with a capital X, while labels are denoted by a lowercase y. This is inspired by the standard formulation f(x)=y in mathematics, where x is the input to a function and y is the output. Following more conventions from mathematics, we use a capital X because the data is a two-dimensional array (a matrix) and a lowercase y because the target is a one-dimensional array (a vector).

First Things First: Look at Your Data

Before building a machine learning model it is often a good idea to inspect the data, to see if the task is easily solvable without machine learning, or if the desired information might not be contained in the data.

Additionally, inspecting your data is a good way to find abnormalities and peculiarities. Maybe some of your irises were measured using inches and not centimeters, for example. In the real world, inconsistencies in the data and unexpected measurements are very common.

One of the best ways to inspect data is to visualize it. One way to do this is by using a scatter plot. A scatter plot of the data puts one feature along the x-axis and another along the y-axis, and draws a dot for each data point. Unfortunately, computer screens have only two dimensions, which allows us to plot only two (or maybe three) features at a time. It is difficult to plot datasets with more than three features this way. One way around this problem is to do a pair plot, which looks at all possible pairs of features. If you have a small number of features, such as the four we have here, this is quite reasonable. You should keep in mind, however, that a pair plot does not show the interaction of all of features at once, so some interesting aspects of the data may not be revealed when visualizing it this way.

Building Model:

k-Nearest Neighbors Now we can start building the actual machine learning model. There are many classification algorithms in scikit-learn that we could use. Here we will use a k-nearest neighbors classifier, which is easy to understand. Building this model only consists of storing the training set. To make a prediction for a new data point, the algorithm finds the point in the training set that is closest to the new point. Then it assigns the label of this training point to the new data point.

The k in k-nearest neighbors signifies that instead of using only the closest neighbor to the new data point, we can consider any fixed number k of neighbors in the training (for example, the closest three or five neighbors). Then, we can make a prediction using the majority class among these neighbors.

All machine learning models in scikit-learn are implemented in their own classes, which are called Estimator classes. The k-nearest neighbors classification algorithm is implemented in the KNeighborsClassifier class in the neighbors module. Before we can use the model, we need to instantiate the class into an object. This is when we will set any parameters of the model.

The knn object encapsulates the algorithm that will be used to build the model from the training data, as well the algorithm to make predictions on new data points. It will also hold the information that the algorithm has extracted from the training data. In the case of KNeighborsClassifier, it will just store the training set.

To build the model on the training set, we call the fit method of the knn object, which takes as arguments the NumPy array X_train containing the training data and the NumPy array y_train of the corresponding training labels.

Making Predictions

We can now make predictions using this model on new data for which we might not know the correct labels. Imagine we found an iris in the wild with a sepal length of 5 cm, a sepal width of 2.9 cm, a petal length of 1 cm, and a petal width of 0.2 cm. What species of iris would this be? To know the answer, refer the Jupiter Notebook [Classiying_Iris_Species.ipynb].

Evaluating the Model

We can make a prediction for each iris in the test data and compare it against its label (the known species). We can measure how well the model works by computing the accuracy, which is the fraction of flowers for which the right species was predicted.

Conclusion:

For this model, the test set accuracy is about 0.97, which means we made the right prediction for 97% of the irises in the test set. Under some mathematical assumptions, this means that we can expect our model to be correct 97% of the time for new irises. For our hobby botanist application, this high level of accuracy means that our model may be trustworthy enough to use.