Test datasets are small contrived datasets that let you test a machine learning algorithm or test harness.

The data from test datasets have well-defined properties, such as linearly or non-linearity, that allow you to explore specific algorithm behavior. The scikit-learn Python library provides a suite of functions for generating samples from configurable test problems for regression and classification.

In this tutorial, you will discover test problems and how to use them in Python with scikit-learn.

After completing this tutorial, you will know:

- How to generate multi-class classification prediction test problems.
- How to generate binary classification prediction test problems.
- How to generate linear regression prediction test problems.

Let's get started.

Tutorial Overview

This tutorial is divided into 3 parts; they are:

1. Test Datasets
2. Classification Test Problems
3. Regression Test Problems

Test Datasets

A problem when developing and implementing machine learning algorithms is how do you know whether you have implemented them correctly. They seem to work even with bugs.

Test datasets are small contrived problems that allow you to test and debug your algorithms and test harness. They are also useful for better understanding the behavior of algorithms in response to changes in hyperparameters.

Below are some desirable properties of test datasets:

- They can be generated quickly and easily.
- They contain "known" or "understood" outcomes for comparison with predictions.
- They are stochastic, allowing random variations on the same problem each time they are generated.
- They are small and easily visualized in two dimensions.

- They can be scaled up trivially.

I recommend using test datasets when getting started with a new machine learning algorithm or when developing a new test harness.

scikit-learn is a Python library for machine learning that provides functions for generating a suite of test problems.

In this tutorial, we will look at some examples of generating test problems for classification and regression algorithms.

Classification Test Problems

Classification is the problem of assigning labels to observations.

In this section, we will look at three classification problems: blobs, moons and circles.

Blobs Classification Problem

The make_blobs() function can be used to generate blobs of points with a Gaussian distribution.

You can control how many blobs to generate and the number of samples to generate, as well as a host of other properties.

The problem is suitable for linear classification problems given the linearly separable nature of the blobs.

The example below generates a 2D dataset of samples with three blobs as a multi-class classification prediction problem. Each observation has two inputs and 0, 1, or 2 class values.

```
# generate 2d classification dat
X, y = make_blobs(n_samples=
```

```
1  # generate 2d classification dataset
2  X, y = make_blobs(n_samples=100, centers=3, n_features=2)
```

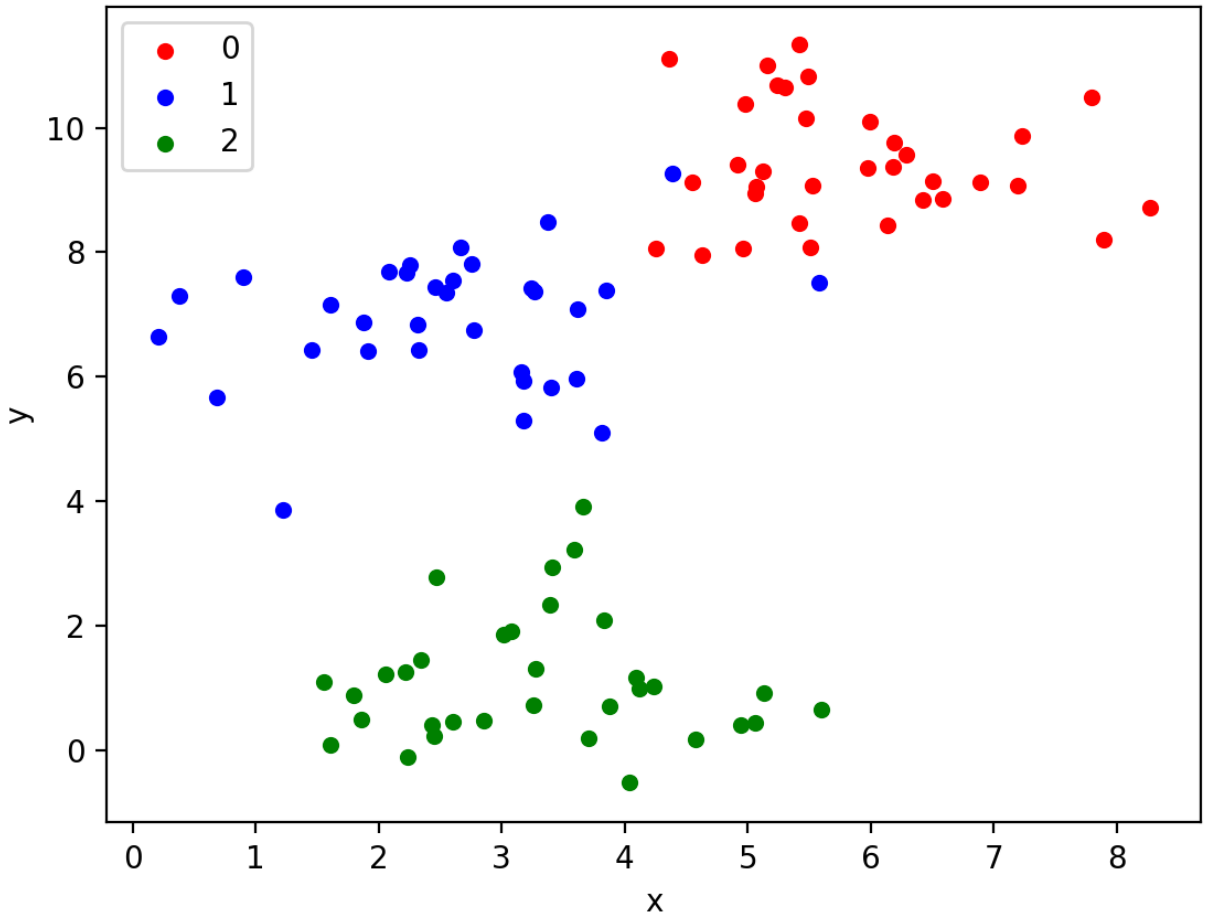
The complete example is listed below.

```
from sklearn.datasets.samples_
from matplotlib import pyplot
from pandas import DataFrame
# generate 2d classification dat
X, y = make_blobs(n_samples=
```

```
from sklearn.datasets.samples_generator import make_blobs
from matplotlib import pyplot
from pandas import DataFrame
# generate 2d classification dataset
X, y = make_blobs(n_samples=100, centers=3, n_features=2)
# scatter plot, dots colored by class value
df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
colors = {0:'red', 1:'blue', 2:'green'}
fig, ax = pyplot.subplots()
grouped = df.groupby('label')
for key, group in grouped:
    group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
pyplot.show()
```

Running the example generates the inputs and outputs for the problem and then creates a handy 2D plot showing points for the different classes using different colors.

Note, your specific dataset and resulting plot will vary given the stochastic nature of the problem generator. This is a feature, not a bug.



Scatter Plot of Blobs Test Classification Problem

We will use this same example structure for the following examples.

Moons Classification Problem

The make_moons() function is for binary classification and will generate a swirl pattern, or two moons.

You can control how noisy the moon shapes are and the number of samples to generate.

This test problem is suitable for algorithms that are capable of learning nonlinear class boundaries.

The example below generates a moon dataset with moderate noise.

```
# generate 2d classification dat
X, y = make_moons(n_samples=
```

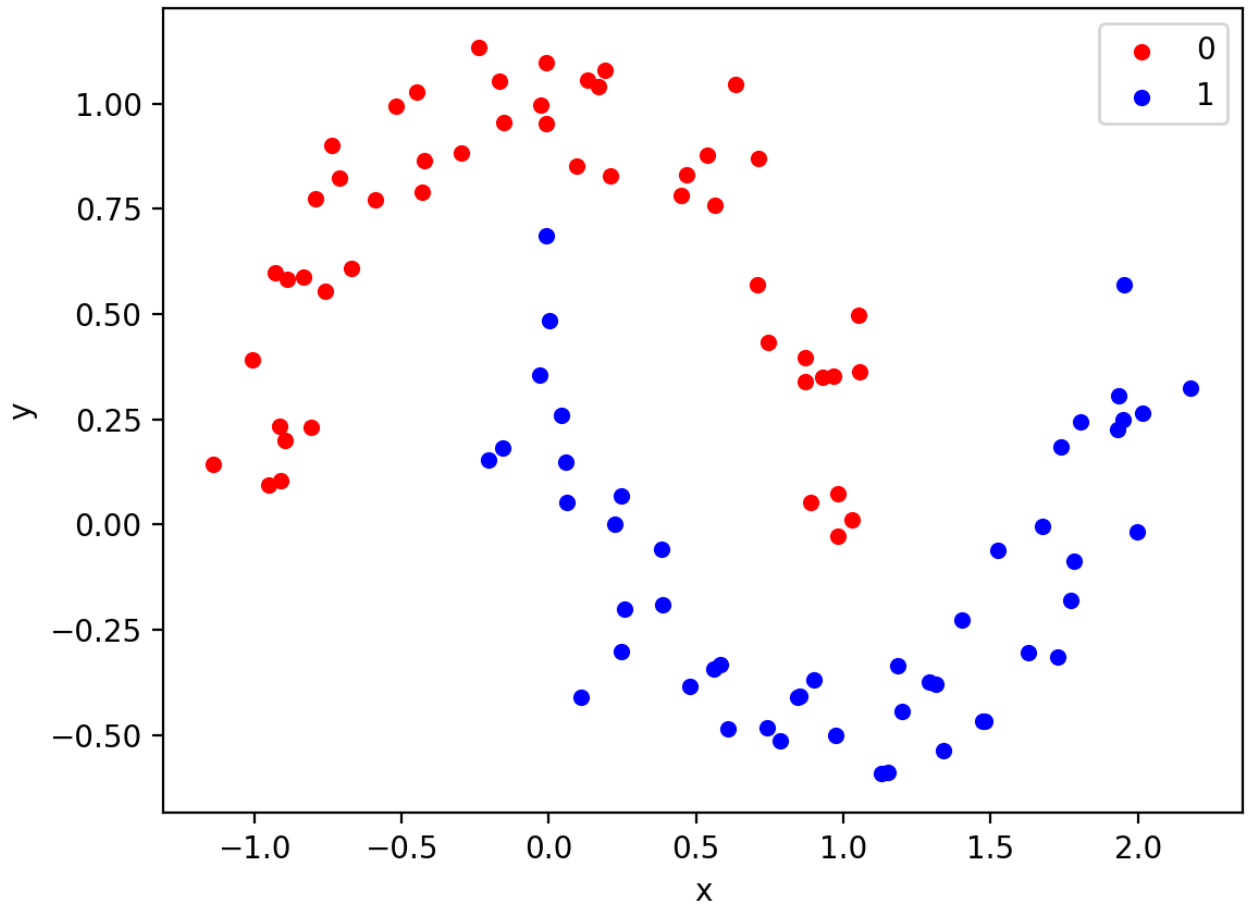
```
1  # generate 2d classification dataset
2  X, y = make_moons(n_samples=100, noise=0.1)
```

The complete example is listed below.

```
from sklearn.datasets import ma
from matplotlib import pyplot
from pandas import DataFrame
# generate 2d classification dat
X, y = make_moons(n_samples=
```

```
1   from sklearn.datasets import make_moons
2   from matplotlib import pyplot
3   from pandas import DataFrame
4   # generate 2d classification dataset
5   X, y = make_moons(n_samples=100, noise=0.1)
6   # scatter plot, dots colored by class value
7   df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
8   colors = {0:'red', 1:'blue'}
9   fig, ax = pyplot.subplots()
10  grouped = df.groupby('label')
11  for key, group in grouped:
12      group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
13  pyplot.show()
```

Running the example generates and plots the dataset for review, again coloring samples by their assigned class.



Scatter plot of Moons Test Classification Problem

Circles Classification Problem

The make_circles() function generates a binary classification problem with datasets that fall into concentric circles.

Again, as with the moons test problem, you can control the amount of noise in the shapes.

This test problem is suitable for algorithms that can learn complex non-linear manifolds.

The example below generates a circles dataset with some noise.

```
# generate 2d classification dat
X, y = make_circles(n_samples=
```

```
1  # generate 2d classification dataset
```

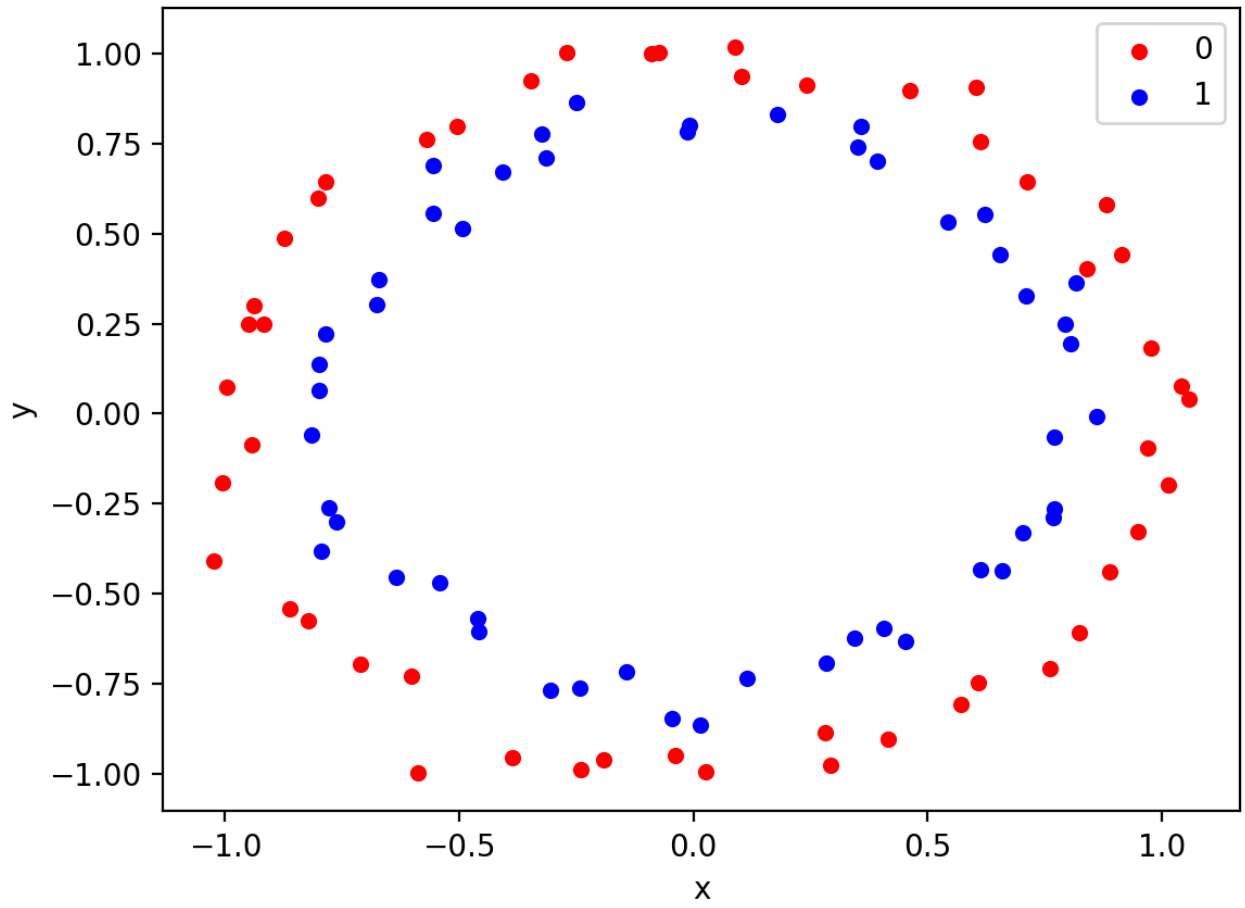
```
2  X, y = make_circles(n_samples=100, noise=0.05)
```

The complete example is listed below.

```
from sklearn.datasets import ma
from matplotlib import pyplot
from pandas import DataFrame
# generate 2d classification dat
X, y = make_circles(n_samples=
```

```
1  from sklearn.datasets import make_circles
2  from matplotlib import pyplot
3  from pandas import DataFrame
4  # generate 2d classification dataset
5  X, y = make_circles(n_samples=100, noise=0.05)
6  # scatter plot, dots colored by class value
7  df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))
8  colors = {0:'red', 1:'blue'}
9  fig, ax = pyplot.subplots()
10 grouped = df.groupby('label')
11 for key, group in grouped:
12     group.plot(ax=ax, kind='scatter', x='x', y='y', label=key, color=colors[key])
13 pyplot.show()
```

Running the example generates and plots the dataset for review.



Scatter Plot of Circles Test Classification Problem

Regression Test Problems

Regression is the problem of predicting a quantity given an observation.

The make_regression() function will create a dataset with a linear relationship between inputs and the outputs.

You can configure the number of samples, number of input features, level of noise, and much more.

This dataset is suitable for algorithms that can learn a linear regression function.

The example below will generate 100 examples with one input feature and one output feature with modest noise.

```
# generate regression dataset
X, y = make_regression(n_sam
```

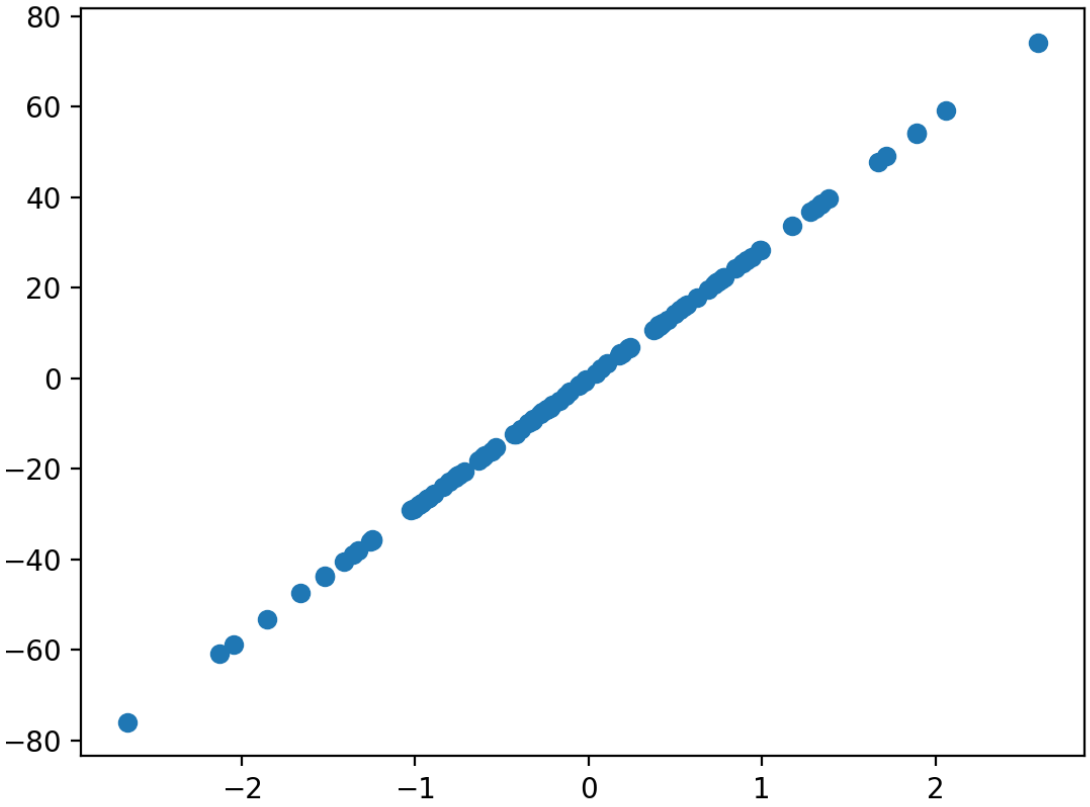
```
1  # generate regression dataset
2  X, y = make_regression(n_samples=100, n_features=1, noise=0.1)
```

The complete example is listed below.

```
from sklearn.datasets import ma
from matplotlib import pyplot
# generate regression dataset
X, y = make_regression(n_sam
# plot regression dataset
```

```
1  from sklearn.datasets import make_regression
2  from matplotlib import pyplot
3  # generate regression dataset
4  X, y = make_regression(n_samples=100, n_features=1, noise=0.1)
5  # plot regression dataset
6  pyplot.scatter(X,y)
7  pyplot.show()
```

Running the example will generate the data and plot the X and y relationship, which, given that it is linear, is quite boring.



Scatter Plot of Regression Test Problem

Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Compare Algorithms**. Select a test problem and compare a suite of algorithms on the problem and report the performance.
- **Scale Up Problem**. Select a test problem and explore scaling it up, use progression methods to visualize the results, and perhaps explore model skill vs problem scale for a given algorithm.
- **Additional Problems**. The library provides a suite of additional test problems; write a code example for each to demonstrate how they work.

If you explore any of these extensions, I'd love to know.