# **Machine Learning Algorithm Recipes in scikit-learn**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on June 20, 2014 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on August 21, 2019

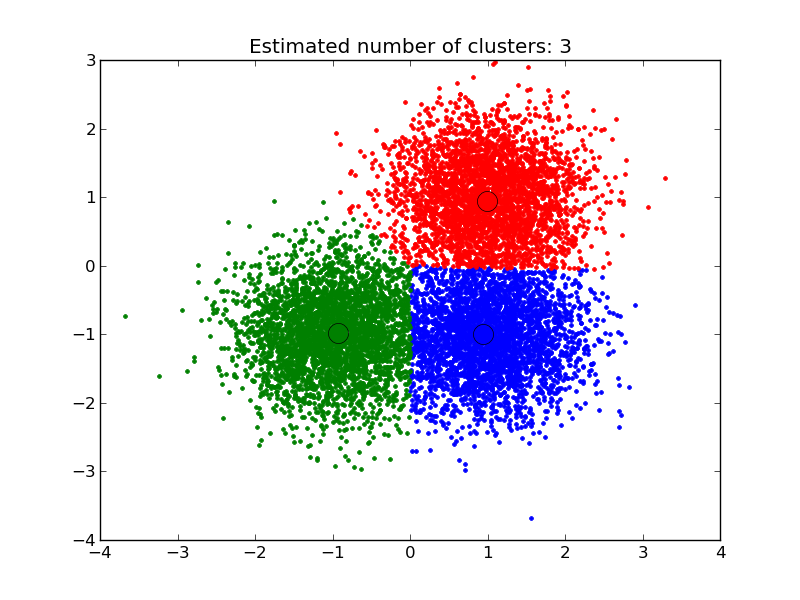
You have to get your hands dirty.

You can read all of the blog posts and watch all the videos in the world, but you’re not actually going to start really get machine learning until you start practicing.

The [scikit-learn Python library](http://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/) is very easy to get up and running. Nevertheless I see a lot of hesitation from beginners looking get started. In this blog post I want to give a few very simple examples of using scikit-learn for some supervised classification algorithms.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

Let’s get started.



## **Scikit-Learn Recipes**

You don’t need to know about and use all of the algorithms in scikit-learn, at least initially, pick one or two (or a handful) and practice with only those.

In this post you will see 5 recipes of supervised classification algorithms applied to small standard datasets that are provided with the scikit-learn library.

The recipes are principled. Each example is:

* **Standalone**: Each code example is a self-contained, complete and executable recipe.
* **Just Code**: The focus of each recipe is on the code with minimal exposition on machine learning theory.
* **Simple**: Recipes present the common use case, which is probably what you are looking to do.
* **Consistent**: All code example are presented consistently and follow the same code pattern and style conventions.

The recipes do not explore the parameters of a given algorithm. They provide a skeleton that you can copy and paste into your file, project or python REPL and start to play with immediately.

These recipes show you that you can get started practicing with scikit-learn right now. Stop putting it off.

## **Logistic Regression**

Logistic regression fits a logistic model to data and makes predictions about the probability of an event (between 0 and 1).

This recipe shows the fitting of a logistic regression model to the iris dataset. Because this is a mutli-class classification problem and logistic regression makes predictions between 0 and 1, a one-vs-all scheme is used (one model per class).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Logistic Regression  from sklearn import datasets  from sklearn import metrics  from sklearn.linear\_model import LogisticRegression  # load the iris datasets  dataset = datasets.load\_iris()  # fit a logistic regression model to the data  model = LogisticRegression()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for Logistic Regression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression) for details on configuring the algorithm parameters. Also see the [Logistic Regression section of the user guide](http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression).

## **Naive Bayes**

Naive Bayes uses Bayes Theorem to model the conditional relationship of each attribute to the class variable.

This recipe shows the fitting of an Naive Bayes model to the iris dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Gaussian Naive Bayes  from sklearn import datasets  from sklearn import metrics  from sklearn.naive\_bayes import GaussianNB  # load the iris datasets  dataset = datasets.load\_iris()  # fit a Naive Bayes model to the data  model = GaussianNB()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for the Gaussian Naive Bayes](http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB) for details on configuring the algorithm parameters. Also see the [Naive Bayes section of the user guide](http://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes).

## **k-Nearest Neighbor**

The k-Nearest Neighbor (kNN) method makes predictions by locating similar cases to a given data instance (using a similarity function) and returning the average or majority of the most similar data instances. The kNN algorithm can be used for classification or regression.

This recipe shows use of the kNN model to make predictions for the iris dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # k-Nearest Neighbor  from sklearn import datasets  from sklearn import metrics  from sklearn.neighbors import KNeighborsClassifier  # load iris the datasets  dataset = datasets.load\_iris()  # fit a k-nearest neighbor model to the data  model = KNeighborsClassifier()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for the k-Nearest Neighbor](http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier) for details on configuring the algorithm parameters. Also see the [k-Nearest Neighbor section of the user guide](http://scikit-learn.org/stable/modules/neighbors.html#neighbors).

## **Classification and Regression Trees**

Classification and Regression Trees (CART) are constructed from a dataset by making splits that best separate the data for the classes or predictions being made. The CART algorithm can be used for classification or regression.

This recipe shows use of the CART model to make predictions for the iris dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Decision Tree Classifier  from sklearn import datasets  from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier  # load the iris datasets  dataset = datasets.load\_iris()  # fit a CART model to the data  model = DecisionTreeClassifier()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for CART](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier) for details on configuring the algorithm parameters. Also see the [Decision Tree section of the user guide](http://scikit-learn.org/stable/modules/tree.html#tree).

## **Support Vector Machines**

[Support Vector Machines (SVM)](https://machinelearningmastery.com/support-vector-machines-for-machine-learning/) are a method that uses points in a transformed problem space that best separate classes into two groups. Classification for multiple classes is supported by a one-vs-all method. SVM also supports regression by modeling the function with a minimum amount of allowable error.

This recipe shows use of the SVM model to make predictions for the iris dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Support Vector Machine  from sklearn import datasets  from sklearn import metrics  from sklearn.svm import SVC  # load the iris datasets  dataset = datasets.load\_iris()  # fit a SVM model to the data  model = SVC()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

For more information see the [API reference for SVM](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC) for details on configuring the algorithm parameters. Also see the [SVM section of the user guide](http://scikit-learn.org/stable/modules/svm.html#svm).

## **Summary**

In this post you have seen 5 self-contained recipes demonstrating some of the most popular and powerful supervised classification problems.

Each example is less than 20 lines that you can copy and paste and start using scikit-learn, right now. Stop reading and start practicing. Pick one recipe and run it, then start to play with the parameters and see what effect that has on the results.

# **A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on April 16, 2014 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on August 21, 2019

If you are a Python programmer or you are looking for a robust library you can use to bring machine learning into a production system then a library that you will want to seriously consider is scikit-learn.

In this post you will get an overview of the scikit-learn library and useful references of where you can learn more.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

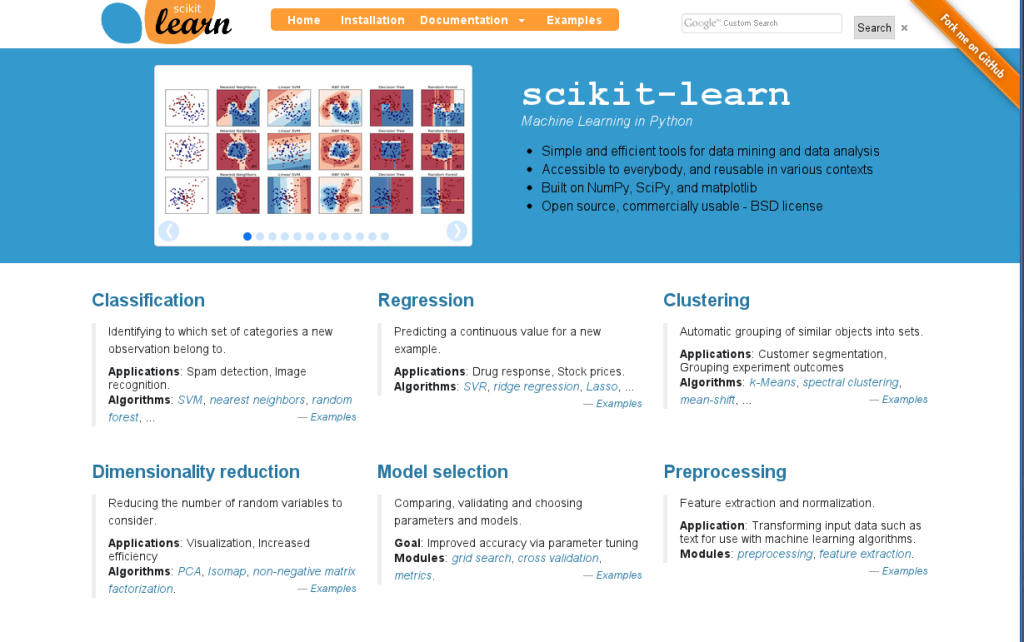
Let’s get started.

## **Where did it come from?**

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.

Later Matthieu Brucher joined the project and started to use it as apart of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.

The project now has more than 30 active contributors and has had paid sponsorship from [INRIA](http://www.inria.fr/en/), Google, [Tinyclues](http://www.tinyclues.com/) and the [Python Software Foundation](https://www.python.org/psf/).



[Scikit-learn Homepage](http://scikit-learn.org/stable/index.html)

## **What is scikit-learn?**

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

* **NumPy**: Base n-dimensional array package
* **SciPy**: Fundamental library for scientific computing
* **Matplotlib**: Comprehensive 2D/3D plotting
* **IPython**: Enhanced interactive console
* **Sympy**: Symbolic mathematics
* **Pandas**: Data structures and analysis

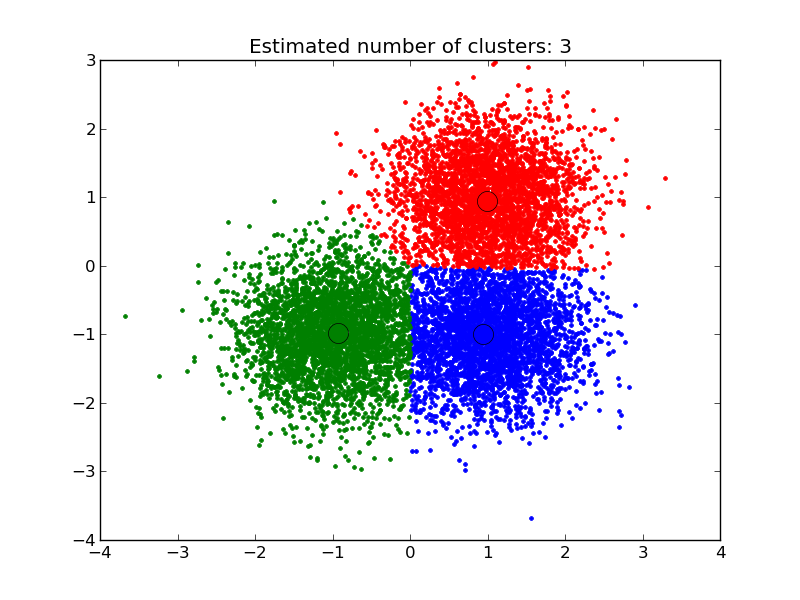
Extensions or modules for SciPy care conventionally named [SciKits](http://scikits.appspot.com/scikits). As such, the module provides learning algorithms and is named scikit-learn.

The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

Although the interface is Python, c-libraries are leverage for performance such as numpy for arrays and matrix operations, [LAPACK](http://www.netlib.org/lapack/), [LibSVM](http://www.csie.ntu.edu.tw/~cjlin/libsvm/) and the careful use of cython.

## **What are the features?**

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.



Screenshot taken from [a demo of the mean-shift clustering algorithm](http://scikit-learn.org/stable/auto_examples/cluster/plot_mean_shift.html)

Some popular groups of models provided by scikit-learn include:

* **Clustering**: for grouping unlabeled data such as KMeans.
* **Cross Validation**: for estimating the performance of supervised models on unseen data.
* **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior.
* **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
* **Ensemble methods**: for combining the predictions of multiple supervised models.
* **Feature extraction**: for defining attributes in image and text data.
* **Feature selection**: for identifying meaningful attributes from which to create supervised models.
* **Parameter Tuning**: for getting the most out of supervised models.
* **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
* **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

## **Example: Classification and Regression Trees**

I want to give you an example to show you how easy it is to use the library.

In this example, we use the Classification and Regression Trees (CART) decision tree algorithm to model the Iris flower dataset.

This dataset is provided as an example dataset with the library and is loaded. The classifier is fit on the data and then predictions are made on the training data.

Finally, the classification accuracy and a [confusion matrix](https://machinelearningmastery.com/confusion-matrix-machine-learning/) is printed.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | # Sample Decision Tree Classifier  from sklearn import datasets  from sklearn import metrics  from sklearn.tree import DecisionTreeClassifier  # load the iris datasets  dataset = datasets.load\_iris()  # fit a CART model to the data  model = DecisionTreeClassifier()  model.fit(dataset.data, dataset.target)  print(model)  # make predictions  expected = dataset.target  predicted = model.predict(dataset.data)  # summarize the fit of the model  print(metrics.classification\_report(expected, predicted))  print(metrics.confusion\_matrix(expected, predicted)) |

Running this example produces the following output, showing you the details of the trained model, the skill of the model according to some common metrics and a confusion matrix.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | DecisionTreeClassifier(class\_weight=None, criterion='gini', max\_depth=None,  max\_features=None, max\_leaf\_nodes=None, min\_samples\_leaf=1,  min\_samples\_split=2, min\_weight\_fraction\_leaf=0.0,  presort=False, random\_state=None, splitter='best')  precision recall f1-score support    0 1.00 1.00 1.00 50  1 1.00 1.00 1.00 50  2 1.00 1.00 1.00 50    avg / total 1.00 1.00 1.00 150    [[50 0 0]  [ 0 50 0]  [ 0 0 50]] |

## **Who is using it?**

The [scikit-learn testimonials page](http://scikit-learn.org/stable/testimonials/testimonials.html) lists Inria, Mendeley, wise.io , Evernote, Telecom ParisTech and AWeber as users of the library.

If this is a small indication of companies that have presented on their use, then there are very likely tens to hundreds of larger organizations using the library.

It has good test coverage and managed releases and is suitable for prototype and production projects alike.

## **Resources**

If you are interested in learning more, checkout the [Scikit-Learn homepage](http://scikit-learn.org/) that includes documentation and related resources.

You can get the code from the [github repository](https://github.com/scikit-learn), and releases are historically available on the [Sourceforge project](http://sourceforge.net/projects/scikit-learn/).

### **Documentation**

I recommend starting out with the quick-start tutorial and flicking through the user guide and example gallery for algorithms that interest you.

Ultimately, scikit-learn is a library and the API reference will be the best documentation for getting things done.

* Quick Start Tutorial <http://scikit-learn.org/stable/tutorial/basic/tutorial.html>
* User Guide <http://scikit-learn.org/stable/user_guide.html>
* API Reference <http://scikit-learn.org/stable/modules/classes.html>
* Example Gallery <http://scikit-learn.org/stable/auto_examples/index.html>

### **Papers**

If you interested in more information about how the project started and it’s vision, there are some papers you may want to check-out.

* [Scikit-learn: Machine Learning in Python](http://jmlr.org/papers/v12/pedregosa11a.html) (2011)
* [API design for machine learning software: experiences from the scikit-learn project](http://arxiv.org/abs/1309.0238) (2013)

### **Books**

If you are looking for a good book, I recommend “Building Machine Learning Systems with Python”. It’s well written and the examples are interesting.

* [Learning scikit-learn: Machine Learning in Python](http://www.amazon.com/dp/1783281936?tag=inspiredalgor-20) (2013)
* [Building Machine Learning Systems with Python](http://www.amazon.com/dp/1782161406?tag=inspiredalgor-20) (2013)
* [Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data](http://www.amazon.com/dp/0691151687?tag=inspiredalgor-20) (2014)

# **How To Compare Machine Learning Algorithms in Python with scikit-learn**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on June 1, 2016 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on December 13, 2019

It is important to compare the performance of multiple different machine learning algorithms consistently.

In this post you will discover how you can create a test harness to compare multiple different machine learning algorithms in Python with scikit-learn.

You can use this test harness as a template on your own machine learning problems and add more and different algorithms to compare.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

Let’s get started.

* **Update Mar/2018**: Added alternate link to download the dataset as the original appears to have been taken down.



How To Compare Machine Learning Algorithms in Python with scikit-learn

Photo by [Michael Knight](https://www.flickr.com/photos/mknightphoto/2295688304/), some rights reserved.

## **Choose The Best Machine Learning Model**

How do you choose the best model for your problem?

When you work on a machine learning project, you often end up with multiple good models to choose from. Each model will have different performance characteristics.

Using resampling methods like cross validation, you can get an estimate for how accurate each model may be on unseen data. You need to be able to use these estimates to choose one or two best models from the suite of models that you have created.

### **Compare Machine Learning Models Carefully**

When you have a new dataset, it is a good idea to visualize the data using different techniques in order to look at the data from different perspectives.

The same idea applies to model selection. You should use a number of different ways of looking at the estimated accuracy of your machine learning algorithms in order to choose the one or two to finalize.

A way to do this is to use different visualization methods to show the average accuracy, variance and other properties of the distribution of model accuracies.

In the next section you will discover exactly how you can do that in Python with scikit-learn.

### **Need help with Machine Learning in Python?**

Take my free 2-week email course and discover data prep, algorithms and more (with code).

Click to sign-up now and also get a free PDF Ebook version of the course.

[**Start Your FREE Mini-Course Now!**](https://machinelearningmastery.leadpages.co/leadbox/146d399f3f72a2%3A164f8be4f346dc/5655869022797824/)

## **Compare Machine Learning Algorithms Consistently**

The key to a fair comparison of machine learning algorithms is ensuring that each algorithm is evaluated in the same way on the same data.

You can achieve this by forcing each algorithm to be evaluated on a consistent test harness.

In the example below 6 different algorithms are compared:

1. Logistic Regression
2. Linear Discriminant Analysis
3. K-Nearest Neighbors
4. Classification and Regression Trees
5. Naive Bayes
6. Support Vector Machines

The problem is a standard binary classification dataset called the Pima Indians onset of diabetes problem. The problem has two classes and eight numeric input variables of varying scales.

You can learn more about the dataset here:

* [Dataset File](https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv).
* [Dataset Details](https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names)

The 10-fold cross validation procedure is used to evaluate each algorithm, importantly configured with the same random seed to ensure that the same splits to the training data are performed and that each algorithms is evaluated in precisely the same way.

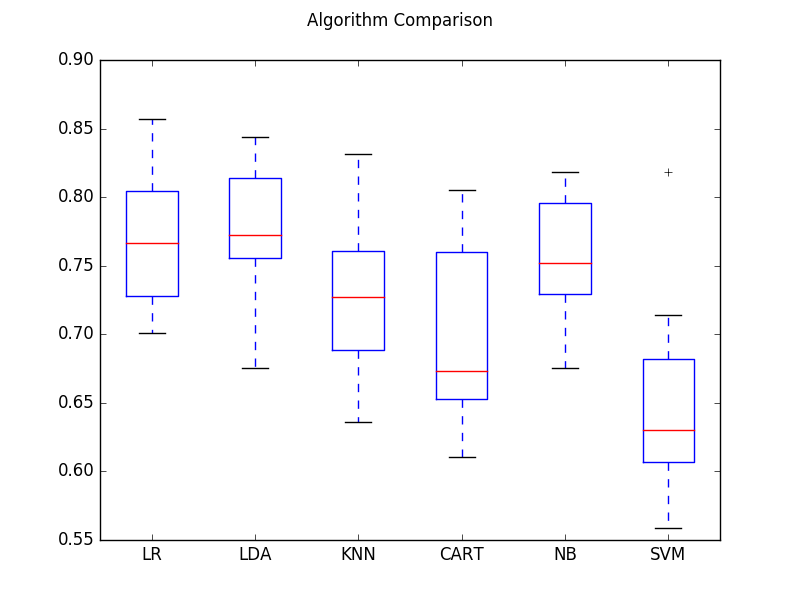
Each algorithm is given a short name, useful for summarizing results afterward.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45 | # Compare Algorithms  import pandas  import matplotlib.pyplot as plt  from sklearn import model\_selection  from sklearn.linear\_model import LogisticRegression  from sklearn.tree import DecisionTreeClassifier  from sklearn.neighbors import KNeighborsClassifier  from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis  from sklearn.naive\_bayes import GaussianNB  from sklearn.svm import SVC  # load dataset  url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"  names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']  dataframe = pandas.read\_csv(url, names=names)  array = dataframe.values  X = array[:,0:8]  Y = array[:,8]  # prepare configuration for cross validation test harness  seed = 7  # prepare models  models = []  models.append(('LR', LogisticRegression()))  models.append(('LDA', LinearDiscriminantAnalysis()))  models.append(('KNN', KNeighborsClassifier()))  models.append(('CART', DecisionTreeClassifier()))  models.append(('NB', GaussianNB()))  models.append(('SVM', SVC()))  # evaluate each model in turn  results = []  names = []  scoring = 'accuracy'  for name, model in models:  kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)  cv\_results = model\_selection.cross\_val\_score(model, X, Y, cv=kfold, scoring=scoring)  results.append(cv\_results)  names.append(name)  msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())  print(msg)  # boxplot algorithm comparison  fig = plt.figure()  fig.suptitle('Algorithm Comparison')  ax = fig.add\_subplot(111)  plt.boxplot(results)  ax.set\_xticklabels(names)  plt.show() |

Running the example provides a list of each algorithm short name, the mean accuracy and the standard deviation accuracy.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | LR: 0.769515 (0.048411)  LDA: 0.773462 (0.051592)  KNN: 0.726555 (0.061821)  CART: 0.695232 (0.062517)  NB: 0.755178 (0.042766)  SVM: 0.651025 (0.072141) |

The example also provides a box and whisker plot showing the spread of the accuracy scores across each cross validation fold for each algorithm.



Compare Machine Learning Algorithms

From these results, it would suggest that both logistic regression and linear discriminate analysis are perhaps worthy of further study on this problem.

## **Summary**

In this post you discovered how to evaluate multiple different machine learning algorithms on a dataset in Python with scikit-learn.

You learned how to both use the same test harness to evaluate the algorithms and how to summarize the results both numerically and using a box and whisker plot.

You can use this recipe as a template for evaluating multiple algorithms on your own problems.

# **How to Generate Test Datasets in Python with scikit-learn**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on January 15, 2018 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on January 10, 2020

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.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

Let’s get started.

* **Updated Jan/2020**: Updated for changes in scikit-learn v0.22 API.

## **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()](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html) 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.

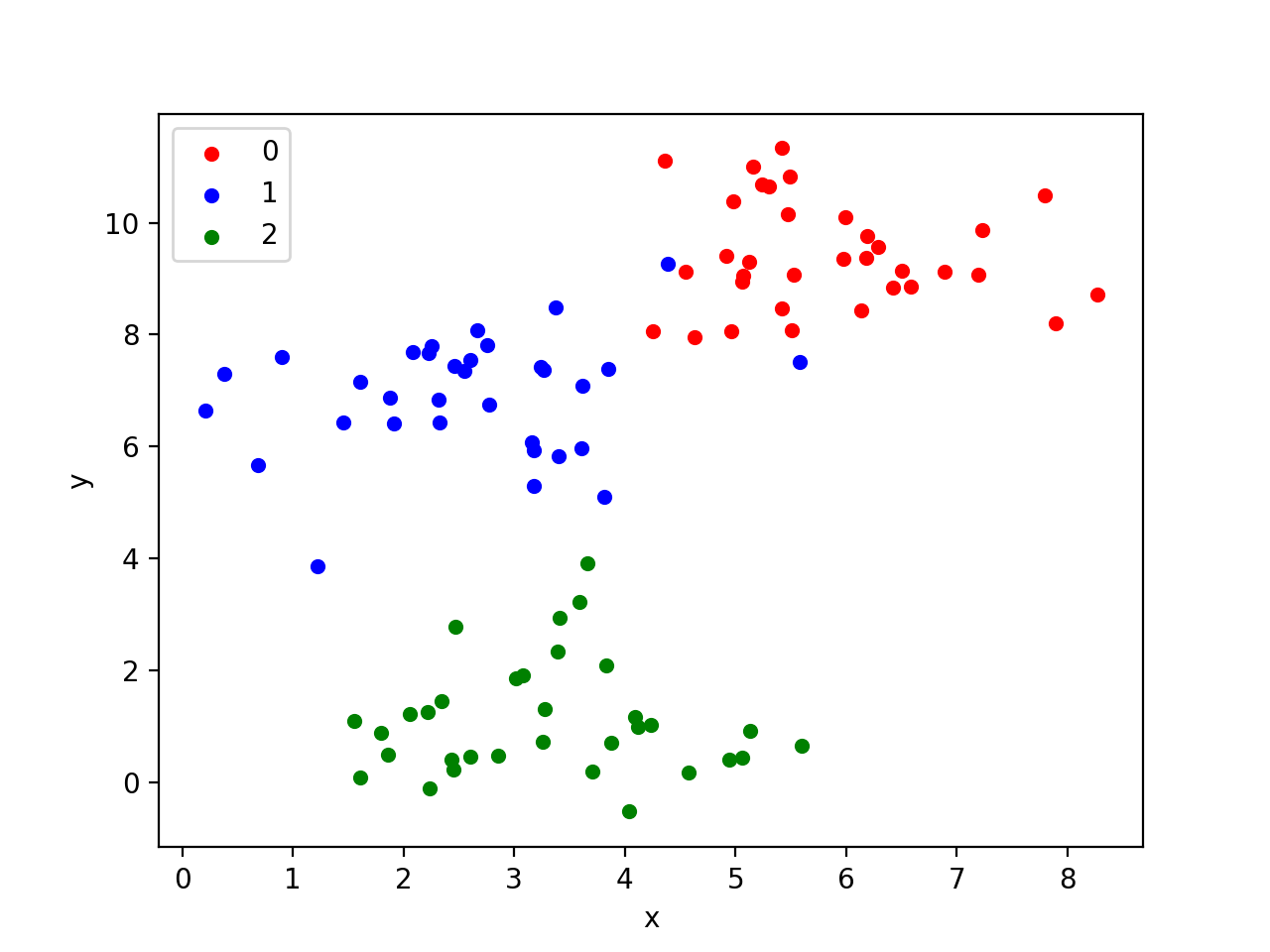
|  |  |
| --- | --- |
| 1  2 | # generate 2d classification dataset  X, y = make\_blobs(n\_samples=100, centers=3, n\_features=2) |

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from sklearn.datasets 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](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html) 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.

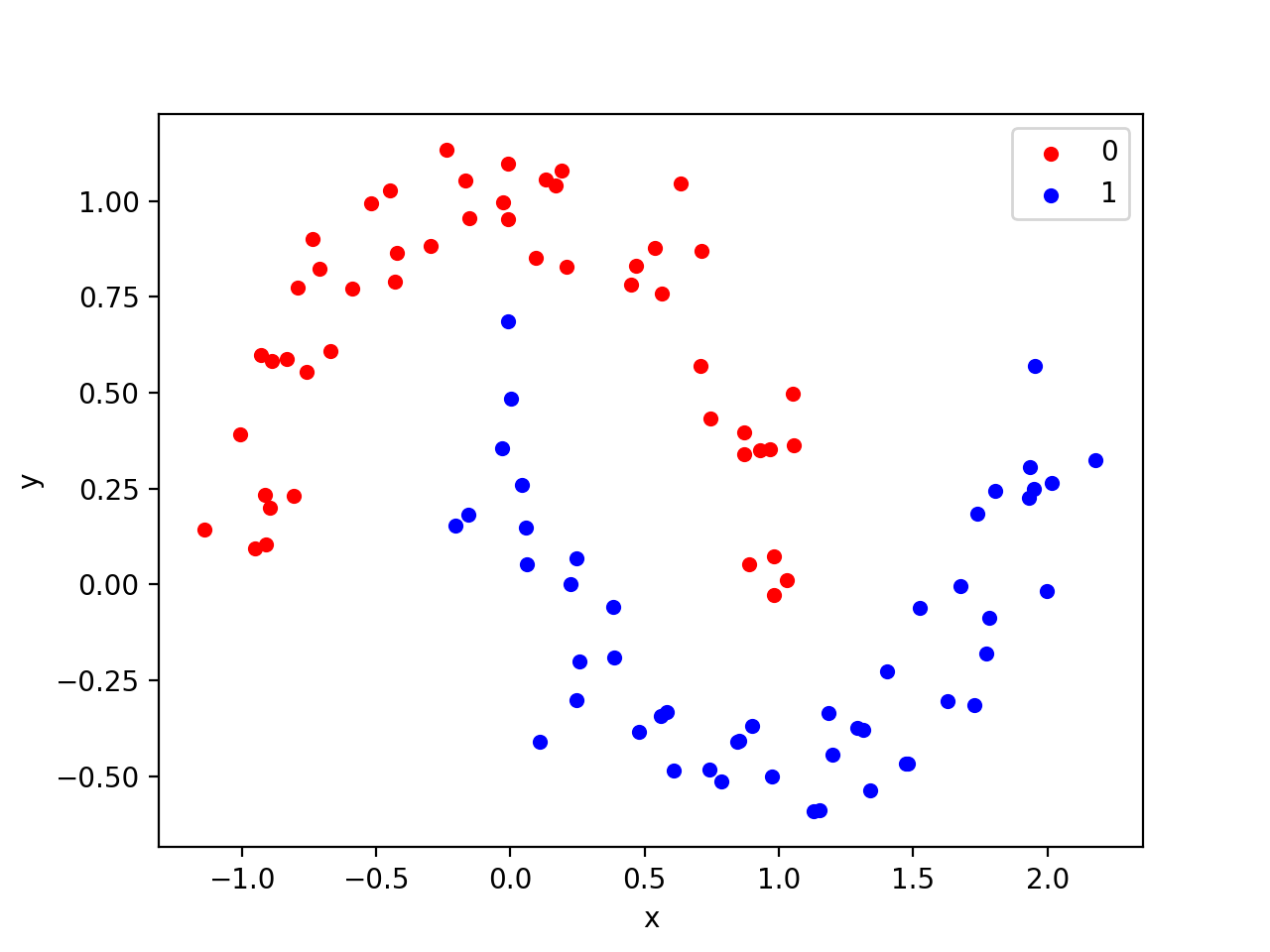
The example below generates a moon dataset with moderate noise.

|  |  |
| --- | --- |
| 1  2 | # generate 2d classification dataset  X, y = make\_moons(n\_samples=100, noise=0.1) |

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from sklearn.datasets import make\_moons  from matplotlib import pyplot  from pandas import DataFrame  # generate 2d classification dataset  X, y = make\_moons(n\_samples=100, noise=0.1)  # scatter plot, dots colored by class value  df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))  colors = {0:'red', 1:'blue'}  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 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](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_circles.html) 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.

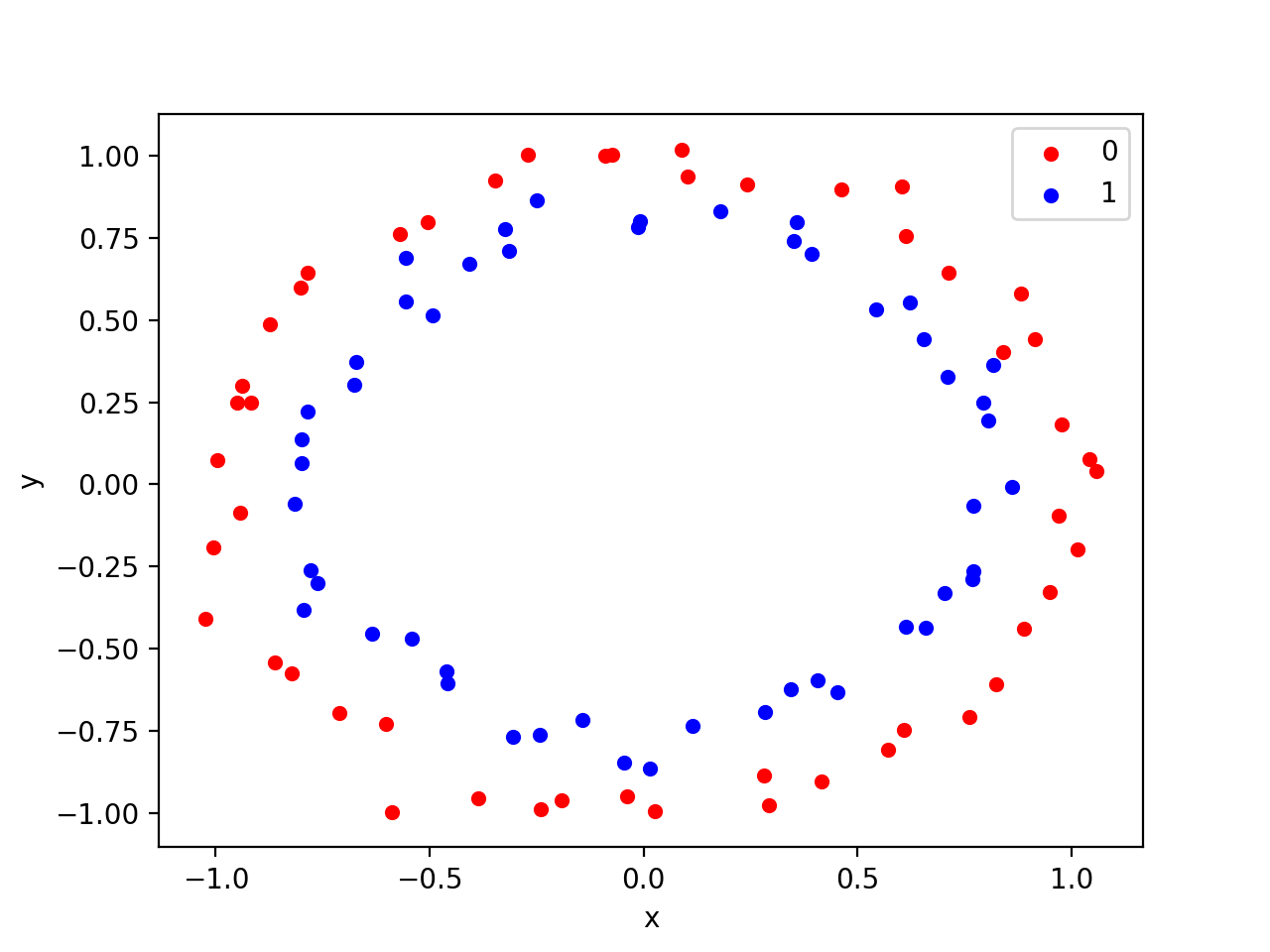
The example below generates a circles dataset with some noise.

|  |  |
| --- | --- |
| 1  2 | # generate 2d classification dataset  X, y = make\_circles(n\_samples=100, noise=0.05) |

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | from sklearn.datasets import make\_circles  from matplotlib import pyplot  from pandas import DataFrame  # generate 2d classification dataset  X, y = make\_circles(n\_samples=100, noise=0.05)  # scatter plot, dots colored by class value  df = DataFrame(dict(x=X[:,0], y=X[:,1], label=y))  colors = {0:'red', 1:'blue'}  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 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](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html) 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.

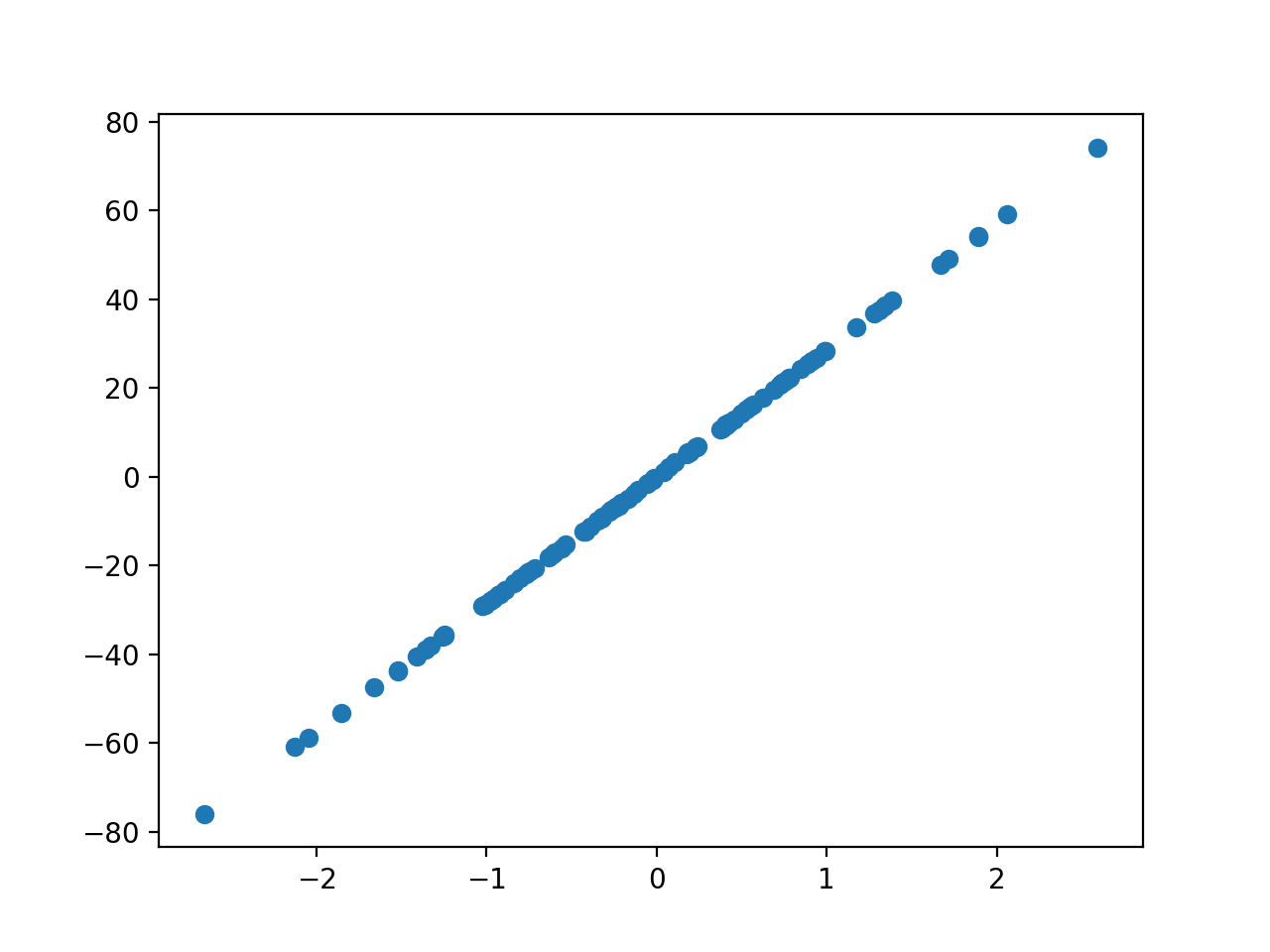
The example below will generate 100 examples with one input feature and one output feature with modest noise.

|  |  |
| --- | --- |
| 1  2 | # generate regression dataset  X, y = make\_regression(n\_samples=100, n\_features=1, noise=0.1) |

The complete example is listed below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | from sklearn.datasets import make\_regression  from matplotlib import pyplot  # generate regression dataset  X, y = make\_regression(n\_samples=100, n\_features=1, noise=0.1)  # plot regression dataset  pyplot.scatter(X,y)  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.

## **Further Reading**

This section provides more resources on the topic if you are looking to go deeper.

* [scikit-learn User Guide: Dataset loading utilities](http://scikit-learn.org/stable/datasets/index.html)
* [scikit-learn API: sklearn.datasets: Datasets](http://scikit-learn.org/stable/modules/classes.html#module-sklearn.datasets)

## **Summary**

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

Specifically, you learned:

* 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.

# **How to Load Data in Python with Scikit-Learn**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on July 9, 2014 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on December 13, 2019

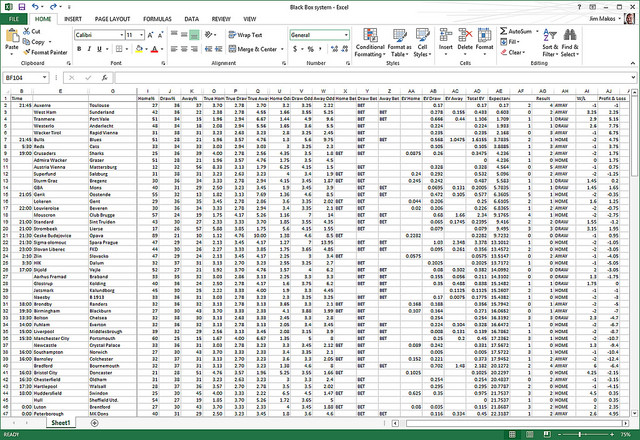
Before you can build machine learning models, you need to load your data into memory.

In this post you will discover how to load data for machine learning in Python using [scikit-learn](http://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/).

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

Let’s get started.

* **Update March/2018**: Added alternate link to download the dataset as the original appears to have been taken down.



Load CSV Data

Photo by [Jim Makos](https://www.flickr.com/photos/jim-makos/13775073055), some rights reserved

## **Packaged Datasets**

The scikit-learn library is [packaged with datasets](http://scikit-learn.org/stable/datasets/). These datasets are useful for getting a handle on a given machine learning algorithm or library feature before using it in your own work.

This recipe demonstrates how to load the famous [Iris flowers dataset](http://en.wikipedia.org/wiki/Iris_flower_data_set).

|  |  |
| --- | --- |
| 1  2  3  4 | # Load the packaged iris flowers dataset  # Iris flower dataset (4x150, reals, multi-label classification)  iris = load\_iris()  print(iris) |

## **Load from CSV**

It is very common for you to have a dataset as a CSV file on your local workstation or on a remote server.

This recipe show you how to load a CSV file from a URL, in this case the Pima Indians diabetes classification dataset.

You can learn more about the dataset here:

* [Dataset File](https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.csv).
* [Dataset Details](https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.names)

From the prepared X and y variables, you can train a machine learning model.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # Load the Pima Indians diabetes dataset from CSV URL  import numpy as np  import urllib  # URL for the Pima Indians Diabetes dataset (UCI Machine Learning Repository)  url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"  # download the file  raw\_data = urllib.urlopen(url)  # load the CSV file as a numpy matrix  dataset = np.loadtxt(raw\_data, delimiter=",")  print(dataset.shape)  # separate the data from the target attributes  X = dataset[:,0:7]  y = dataset[:,8] |

## **Summary**

In this post you discovered that the scikit-learn method comes with packaged data sets including the iris flowers dataset. These datasets can be loaded easily and used for explore and experiment with different machine learning models.

You also saw how you can load CSV data with scikit-learn. You learned a way of opening CSV files from the web using the [urllib library](https://docs.python.org/2/library/urllib.html) and how you can read that data as a NumPy matrix for use in scikit-learn.

# **How to Make Predictions with scikit-learn**

by [**Jason Brownlee**](https://machinelearningmastery.com/author/jasonb/) on April 6, 2018 in [**Python Machine Learning**](https://machinelearningmastery.com/category/python-machine-learning/)

Tweet **Share**

**Share**

Last Updated on January 10, 2020

#### **How to predict classification or regression outcomes**

#### **with scikit-learn models in Python.**

Once you choose and fit a final machine learning model in scikit-learn, you can use it to make predictions on new data instances.

There is some confusion amongst beginners about how exactly to do this. I often see questions such as:

How do I make predictions with my model in scikit-learn?

In this tutorial, you will discover exactly how you can make classification and regression predictions with a finalized machine learning model in the scikit-learn Python library.

After completing this tutorial, you will know:

* How to finalize a model in order to make it ready for making predictions.
* How to make class and probability predictions in scikit-learn.
* How to make regression predictions in scikit-learn.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more [in my new book](https://machinelearningmastery.com/machine-learning-with-python/), with 16 step-by-step tutorials, 3 projects, and full python code.

Let’s get started.

* **Updated Jan/2020**: Updated for changes in scikit-learn v0.22 API.



Gentle Introduction to Vector Norms in Machine Learning

Photo by Cosimo, some rights reserved.

## **Tutorial Overview**

This tutorial is divided into 3 parts; they are:

1. First Finalize Your Model
2. How to Predict With Classification Models
3. How to Predict With Regression Models

## **1. First Finalize Your Model**

Before you can make predictions, you must train a final model.

You may have trained models using k-fold cross validation or train/test splits of your data. This was done in order to give you an estimate of the skill of the model on out-of-sample data, e.g. new data.

These models have served their purpose and can now be discarded.

You now must train a final model on all of your available data.

You can learn more about how to train a final model here:

* [How to Train a Final Machine Learning Model](https://machinelearningmastery.com/train-final-machine-learning-model/)

## **2. How to Predict With Classification Models**

Classification problems are those where the model learns a mapping between input features and an output feature that is a label, such as “*spam*” and “*not spam*.”

Below is sample code of a finalized [LogisticRegression](http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) model for a simple binary classification problem.

Although we are using *LogisticRegression* in this tutorial, the same functions are available on practically all classification algorithms in scikit-learn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # example of training a final classification model  from sklearn.linear\_model import LogisticRegression  from sklearn.datasets import make\_blobs  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=100, centers=2, n\_features=2, random\_state=1)  # fit final model  model = LogisticRegression()  model.fit(X, y) |

After finalizing your model, you may want to save the model to file, e.g. via pickle. Once saved, you can load the model any time and use it to make predictions. For an example of this, see the post:

* [Save and Load Machine Learning Models in Python with scikit-learn](https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/)

For simplicity, we will skip this step for the examples in this tutorial.

There are two types of classification predictions we may wish to make with our finalized model; they are class predictions and probability predictions.

### **Class Predictions**

A class prediction is: given the finalized model and one or more data instances, predict the class for the data instances.

We do not know the outcome classes for the new data. That is why we need the model in the first place.

We can predict the class for new data instances using our finalized classification model in scikit-learn using the *predict()* function.

For example, we have one or more data instances in an array called *Xnew*. This can be passed to the *predict()* function on our model in order to predict the class values for each instance in the array.

|  |  |
| --- | --- |
| 1  2 | Xnew = [[...], [...]]  ynew = model.predict(Xnew) |

### **Multiple Class Predictions**

Let’s make this concrete with an example of predicting multiple data instances at once.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # example of training a final classification model  from sklearn.linear\_model import LogisticRegression  from sklearn.datasets import make\_blobs  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=100, centers=2, n\_features=2, random\_state=1)  # fit final model  model = LogisticRegression()  model.fit(X, y)  # new instances where we do not know the answer  Xnew, \_ = make\_blobs(n\_samples=3, centers=2, n\_features=2, random\_state=1)  # make a prediction  ynew = model.predict(Xnew)  # show the inputs and predicted outputs  for i in range(len(Xnew)):  print("X=%s, Predicted=%s" % (Xnew[i], ynew[i])) |

Running the example predicts the class for the three new data instances, then prints the data and the predictions together.

|  |  |
| --- | --- |
| 1  2  3 | X=[-0.79415228 2.10495117], Predicted=0  X=[-8.25290074 -4.71455545], Predicted=1  X=[-2.18773166 3.33352125], Predicted=0 |

### **Single Class Prediction**

If you had just one new data instance, you can provide this as instance wrapped in an array to the *predict()* function; for example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | # example of making a single class prediction  from sklearn.linear\_model import LogisticRegression  from sklearn.datasets import make\_blobs  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=100, centers=2, n\_features=2, random\_state=1)  # fit final model  model = LogisticRegression()  model.fit(X, y)  # define one new instance  Xnew = [[-0.79415228, 2.10495117]]  # make a prediction  ynew = model.predict(Xnew)  print("X=%s, Predicted=%s" % (Xnew[0], ynew[0])) |

Running the example prints the single instance and the predicted class.

|  |  |
| --- | --- |
| 1 | X=[-0.79415228, 2.10495117], Predicted=0 |

### **A Note on Class Labels**

When you prepared your data, you will have mapped the class values from your domain (such as strings) to integer values. You may have used a [LabelEncoder](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html#sklearn.preprocessing.LabelEncoder).

This *LabelEncoder* can be used to convert the integers back into string values via the *inverse\_transform()* function.

For this reason, you may want to save (pickle) the *LabelEncoder* used to encode your y values when fitting your final model.

### **Probability Predictions**

Another type of prediction you may wish to make is the probability of the data instance belonging to each class.

This is called a probability prediction where given a new instance, the model returns the probability for each outcome class as a value between 0 and 1.

You can make these types of predictions in scikit-learn by calling the *predict\_proba()* function, for example:

|  |  |
| --- | --- |
| 1  2 | Xnew = [[...], [...]]  ynew = model.predict\_proba(Xnew) |

This function is only available on those classification models capable of making a probability prediction, which is most, but not all, models.

The example below makes a probability prediction for each example in the *Xnew* array of data instance.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # example of making multiple probability predictions  from sklearn.linear\_model import LogisticRegression  from sklearn.datasets import make\_blobs  # generate 2d classification dataset  X, y = make\_blobs(n\_samples=100, centers=2, n\_features=2, random\_state=1)  # fit final model  model = LogisticRegression()  model.fit(X, y)  # new instances where we do not know the answer  Xnew, \_ = make\_blobs(n\_samples=3, centers=2, n\_features=2, random\_state=1)  # make a prediction  ynew = model.predict\_proba(Xnew)  # show the inputs and predicted probabilities  for i in range(len(Xnew)):  print("X=%s, Predicted=%s" % (Xnew[i], ynew[i])) |

Running the instance makes the probability predictions and then prints the input data instance and the probability of each instance belonging to the first and second classes (0 and 1).

|  |  |
| --- | --- |
| 1  2  3 | X=[-0.79415228 2.10495117], Predicted=[0.94556472 0.05443528]  X=[-8.25290074 -4.71455545], Predicted=[3.60980873e-04 9.99639019e-01]  X=[-2.18773166 3.33352125], Predicted=[0.98437415 0.01562585] |

This can be helpful in your application if you want to present the probabilities to the user for expert interpretation.

## **3. How to Predict With Regression Models**

Regression is a supervised learning problem where, given input examples, the model learns a mapping to suitable output quantities, such as “0.1” and “0.2”, etc.

Below is an example of a finalized *LinearRegression* model. Again, the functions demonstrated for making regression predictions apply to all of the regression models available in scikit-learn.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | # example of training a final regression model  from sklearn.linear\_model import LinearRegression  from sklearn.datasets import make\_regression  # generate regression dataset  X, y = make\_regression(n\_samples=100, n\_features=2, noise=0.1, random\_state=1)  # fit final model  model = LinearRegression()  model.fit(X, y) |

We can predict quantities with the finalized regression model by calling the *predict()* function on the finalized model.

As with classification, the predict() function takes a list or array of one or more data instances.

### **Multiple Regression Predictions**

The example below demonstrates how to make regression predictions on multiple data instances with an unknown expected outcome.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | # example of training a final regression model  from sklearn.linear\_model import LinearRegression  from sklearn.datasets import make\_regression  # generate regression dataset  X, y = make\_regression(n\_samples=100, n\_features=2, noise=0.1)  # fit final model  model = LinearRegression()  model.fit(X, y)  # new instances where we do not know the answer  Xnew, \_ = make\_regression(n\_samples=3, n\_features=2, noise=0.1, random\_state=1)  # make a prediction  ynew = model.predict(Xnew)  # show the inputs and predicted outputs  for i in range(len(Xnew)):  print("X=%s, Predicted=%s" % (Xnew[i], ynew[i])) |

Running the example makes multiple predictions, then prints the inputs and predictions side-by-side for review.

|  |  |
| --- | --- |
| 1  2  3 | X=[-1.07296862 -0.52817175], Predicted=-61.32459258381131  X=[-0.61175641 1.62434536], Predicted=-30.922508147981667  X=[-2.3015387 0.86540763], Predicted=-127.34448527071137 |

### **Single Regression Prediction**

The same function can be used to make a prediction for a single data instance as long as it is suitably wrapped in a surrounding list or array.

For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # example of training a final regression model  from sklearn.linear\_model import LinearRegression  from sklearn.datasets import make\_regression  # generate regression dataset  X, y = make\_regression(n\_samples=100, n\_features=2, noise=0.1)  # fit final model  model = LinearRegression()  model.fit(X, y)  # define one new data instance  Xnew = [[-1.07296862, -0.52817175]]  # make a prediction  ynew = model.predict(Xnew)  # show the inputs and predicted outputs  print("X=%s, Predicted=%s" % (Xnew[0], ynew[0])) |

Running the example makes a single prediction and prints the data instance and prediction for review.

|  |  |
| --- | --- |
| 1 | X=[-1.07296862, -0.52817175], Predicted=-77.17947088762787 |

## **Further Reading**

This section provides more resources on the topic if you are looking to go deeper.

* [How to Train a Final Machine Learning Model](https://machinelearningmastery.com/train-final-machine-learning-model/)
* [Save and Load Machine Learning Models in Python with scikit-learn](https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/)
* [scikit-learn API Reference](http://scikit-learn.org/stable/modules/classes.html)

### **Summary**

In this tutorial, you discovered how you can make classification and regression predictions with a finalized machine learning model in the scikit-learn Python library.

Specifically, you learned:

* How to finalize a model in order to make it ready for making predictions.
* How to make class and probability predictions in scikit-learn.
* How to make regression predictions in scikit-learn.