**How Do You Start Machine Learning in Python?**

The best way to learn machine learning is by designing and completing small projects.

**Python Can Be Intimidating When Getting Started**

Python is a popular and powerful interpreted language. Unlike R, Python is a complete language and platform that you can use for both research and development and developing production systems.

There are also a lot of modules and libraries to choose from, providing multiple ways to do each task. It can feel overwhelming.

The best way to get started using Python for machine learning is to complete a project.

* It will force you to install and start the Python interpreter (at the very least).
* It will given you a bird’s eye view of how to step through a small project.
* It will give you confidence, maybe to go on to your own small projects.

**Beginners Need A Small End-to-End Project**

Books and courses are frustrating. They give you lots of recipes and snippets, but you never get to see how they all fit together.

When you are applying machine learning to your own datasets, you are working on a project.

A machine learning project may not be linear, but it has a number of well known steps:

1. Define Problem.
2. Prepare Data.
3. Evaluate Algorithms.
4. Improve Results.
5. Present Results.

The best way to really come to terms with a new platform or tool is to work through a machine learning project end-to-end and cover the key steps. Namely, from loading data, summarizing data, evaluating algorithms and making some predictions.

If you can do that, you have a template that you can use on dataset after dataset. You can fill in the gaps such as further data preparation and improving result tasks later, once you have more confidence.

**Hello World of Machine Learning**

The best small project to start with on a new tool is the classification of iris flowers (e.g. [the iris dataset](https://archive.ics.uci.edu/ml/datasets/Iris)).

This is a good project because it is so well understood.

* Attributes are numeric so you have to figure out how to load and handle data.
* It is a classification problem, allowing you to practice with perhaps an easier type of supervised learning algorithm.
* It is a multi-class classification problem (multi-nominal) that may require some specialized handling.
* It only has 4 attributes and 150 rows, meaning it is small and easily fits into memory (and a screen or A4 page).
* All of the numeric attributes are in the same units and the same scale, not requiring any special scaling or transforms to get started.

Let’s get started with your hello world machine learning project in Python.

**Machine Learning in Python: Step-By-Step**

In this section, we are going to work through a small machine learning project end-to-end.

Here is an overview of what we are going to cover:

1. Installing the Python and SciPy platform.
2. Loading the dataset.
3. Summarizing the dataset.
4. Visualizing the dataset.
5. Evaluating some algorithms.
6. Making some predictions.

Take your time. Work through each step.

Try to type in the commands yourself or copy-and-paste the commands to speed things up.

If you have any questions at all, please leave a comment at the bottom of the post.

**1. Downloading, Installing and Starting Python SciPy**

Get the Python and SciPy platform installed on your system if it is not already.

I do not want to cover this in great detail, because others already have. This is already pretty straightforward, especially if you are a developer. If you do need help, ask a question in the comments.

**1.1 Install SciPy Libraries**

This tutorial assumes Python version 2.7 or 3.5.

There are 5 key libraries that you will need to install. Below is a list of the Python SciPy libraries required for this tutorial:

* scipy
* numpy
* matplotlib
* pandas
* sklearn

There are many ways to install these libraries. My best advice is to pick one method then be consistent in installing each library.

The scipy installation page provides excellent instructions for installing the above libraries on multiple different platforms, such as Linux, mac OS X and Windows. If you have any doubts or questions, refer to this guide, it has been followed by thousands of people.

* On Mac OS X, you can use macports to install Python 2.7 and these libraries.
* On Linux you can use your package manager, such as yum on Fedora to install RPMs.

If you are on Windows or you are not confident, I would recommend installing the free version of [Anaconda](https://www.continuum.io/downloads) that includes everything you need.

**1.2 Start Python and Check Versions**

It is a good idea to make sure your Python environment was installed successfully and is working as expected.

The script below will help you test out your environment. It imports each library required in this tutorial and prints the version.

|  |  |
| --- | --- |
| 1 | Python |

I recommend working directly in the interpreter or writing your scripts and running them on the command line rather than big editors and IDEs. Keep things simple and focus on the machine learning not the toolchain.

Type or copy and paste the following script:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | # Check the versions of libraries    # Python version  import sys  print('Python: {}'.format(sys.version))  # scipy  import scipy  print('scipy: {}'.format(scipy.\_\_version\_\_))  # numpy  import numpy  print('numpy: {}'.format(numpy.\_\_version\_\_))  # matplotlib  import matplotlib  print('matplotlib: {}'.format(matplotlib.\_\_version\_\_))  # pandas  import pandas  print('pandas: {}'.format(pandas.\_\_version\_\_))  # scikit-learn  import sklearn  print('sklearn: {}'.format(sklearn.\_\_version\_\_)) |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | Python: 2.7.11 (default, Mar  1 2016, 18:40:10)  [GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)]  scipy: 0.17.0  numpy: 1.10.4  matplotlib: 1.5.1  pandas: 0.17.1  sklearn: 0.18.1 |

Compare the above output to your versions.

Ideally, your versions should match or be more recent. The APIs do not change quickly, so do not be too concerned if you are a few versions behind, Everything in this tutorial will very likely still work for you.

If you get an error, stop. Now is the time to fix it.

If you cannot run the above script cleanly you will not be able to complete this tutorial.

My best advice is to Google search for your error message or post a question on [Stack Exchange](http://stackoverflow.com/questions/tagged/python).

**2. Load The Data**

We are going to use the iris flowers dataset. This dataset is famous because it is used as the “hello world” dataset in machine learning and statistics by pretty much everyone.

The dataset contains 150 observations of iris flowers. There are four columns of measurements of the flowers in centimeters. The fifth column is the species of the flower observed. All observed flowers belong to one of three species.

You can [learn more about this dataset on Wikipedia](https://en.wikipedia.org/wiki/Iris_flower_data_set).

In this step we are going to load the iris data from CSV file URL.

**2.1 Import libraries**

First, let’s import all of the modules, functions and objects we are going to use in this tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | # Load libraries  import pandas  from pandas.plotting import scatter\_matrix  import matplotlib.pyplot as plt  from sklearn import model\_selection  from sklearn.metrics import classification\_report  from sklearn.metrics import confusion\_matrix  from sklearn.metrics import accuracy\_score  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 |

Everything should load without error. If you have an error, stop. You need a working SciPy environment before continuing. See the advice above about setting up your environment.

**2.2 Load Dataset**

We can load the data directly from the UCI Machine Learning repository.

We are using pandas to load the data. We will also use pandas next to explore the data both with descriptive statistics and data visualization.

Note that we are specifying the names of each column when loading the data. This will help later when we explore the data.

|  |  |
| --- | --- |
| 1  2  3  4 | # Load dataset  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"  names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']  dataset = pandas.read\_csv(url, names=names) |

The dataset should load without incident.

If you do have network problems, you can download the [iris.data](https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data) file into your working directory and load it using the same method, changing URL to the local file name.

**3. Summarize the Dataset**

Now it is time to take a look at the data.

In this step we are going to take a look at the data a few different ways:

1. Dimensions of the dataset.
2. Peek at the data itself.
3. Statistical summary of all attributes.
4. Breakdown of the data by the class variable.

Don’t worry, each look at the data is one command. These are useful commands that you can use again and again on future projects.

**3.1 Dimensions of Dataset**

We can get a quick idea of how many instances (rows) and how many attributes (columns) the data contains with the shape property.

|  |  |
| --- | --- |
| 1  2 | # shape  print(dataset.shape) |

You should see 150 instances and 5 attributes:

|  |  |
| --- | --- |
| 1 | (150, 5) |

**3.2 Peek at the Data**

It is also always a good idea to actually eyeball your data.

|  |  |
| --- | --- |
| 1  2 | # head  print(dataset.head(20)) |

You should see the first 20 rows of the data:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | sepal-length  sepal-width  petal-length  petal-width        class  0            5.1          3.5           1.4          0.2  Iris-setosa  1            4.9          3.0           1.4          0.2  Iris-setosa  2            4.7          3.2           1.3          0.2  Iris-setosa  3            4.6          3.1           1.5          0.2  Iris-setosa  4            5.0          3.6           1.4          0.2  Iris-setosa  5            5.4          3.9           1.7          0.4  Iris-setosa  6            4.6          3.4           1.4          0.3  Iris-setosa  7            5.0          3.4           1.5          0.2  Iris-setosa  8            4.4          2.9           1.4          0.2  Iris-setosa  9            4.9          3.1           1.5          0.1  Iris-setosa  10           5.4          3.7           1.5          0.2  Iris-setosa  11           4.8          3.4           1.6          0.2  Iris-setosa  12           4.8          3.0           1.4          0.1  Iris-setosa  13           4.3          3.0           1.1          0.1  Iris-setosa  14           5.8          4.0           1.2          0.2  Iris-setosa  15           5.7          4.4           1.5          0.4  Iris-setosa  16           5.4          3.9           1.3          0.4  Iris-setosa  17           5.1          3.5           1.4          0.3  Iris-setosa  18           5.7          3.8           1.7          0.3  Iris-setosa  19           5.1          3.8           1.5          0.3  Iris-setosa |

**3.3 Statistical Summary**

Now we can take a look at a summary of each attribute.

This includes the count, mean, the min and max values as well as some percentiles.

|  |  |
| --- | --- |
| 1  2 | # descriptions  print(dataset.describe()) |

We can see that all of the numerical values have the same scale (centimeters) and similar ranges between 0 and 8 centimeters.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | sepal-length  sepal-width  petal-length  petal-width  count    150.000000   150.000000    150.000000   150.000000  mean       5.843333     3.054000      3.758667     1.198667  std        0.828066     0.433594      1.764420     0.763161  min        4.300000     2.000000      1.000000     0.100000  25%        5.100000     2.800000      1.600000     0.300000  50%        5.800000     3.000000      4.350000     1.300000  75%        6.400000     3.300000      5.100000     1.800000  max        7.900000     4.400000      6.900000     2.500000 |

**3.4 Class Distribution**

Let’s now take a look at the number of instances (rows) that belong to each class. We can view this as an absolute count.

|  |  |
| --- | --- |
| 1  2 | # class distribution  print(dataset.groupby('class').size()) |

We can see that each class has the same number of instances (50 or 33% of the dataset).

|  |  |
| --- | --- |
| 1  2  3  4 | Class  Iris-setosa        50  Iris-versicolor    50  Iris-virginica     50 |

**4. Data Visualization**

We now have a basic idea about the data. We need to extend that with some visualizations.

We are going to look at two types of plots:

1. Univariate plots to better understand each attribute.
2. Multivariate plots to better understand the relationships between attributes.

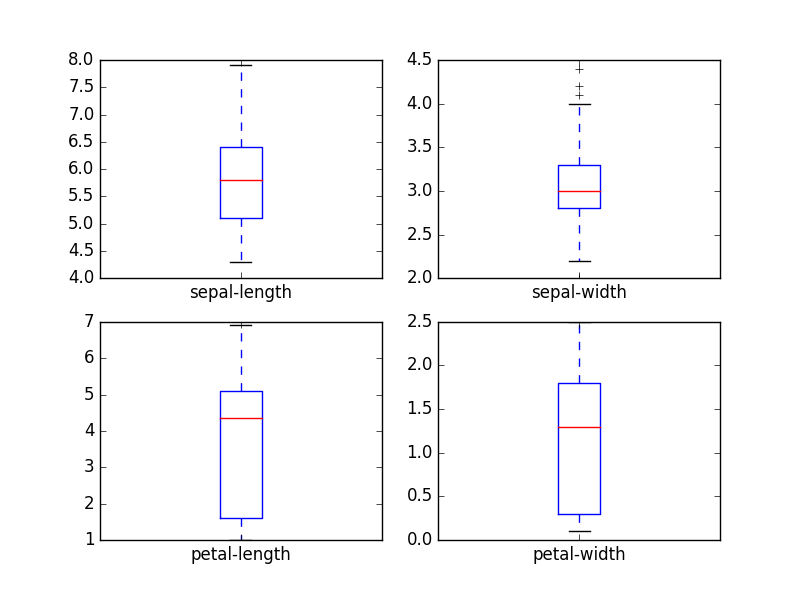
**4.1 Univariate Plots**

We start with some univariate plots, that is, plots of each individual variable.

Given that the input variables are numeric, we can create box and whisker plots of each.

|  |  |
| --- | --- |
| 1  2  3 | # box and whisker plots  dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)  plt.show() |

This gives us a much clearer idea of the distribution of the input attributes:

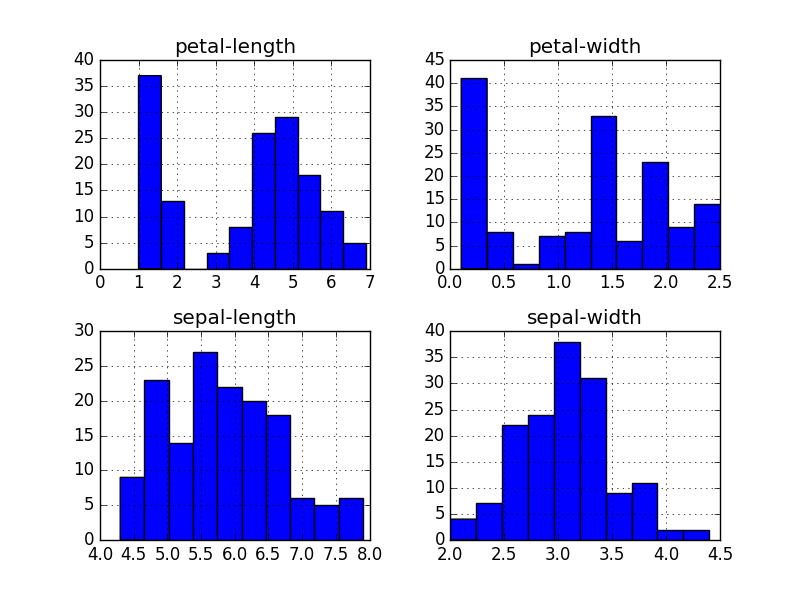


Box and Whisker Plots

We can also create a histogram of each input variable to get an idea of the distribution.

|  |  |
| --- | --- |
| 1  2  3 | # histograms  dataset.hist()  plt.show() |

It looks like perhaps two of the input variables have a Gaussian distribution. This is useful to note as we can use algorithms that can exploit this assumption.



Histogram Plots

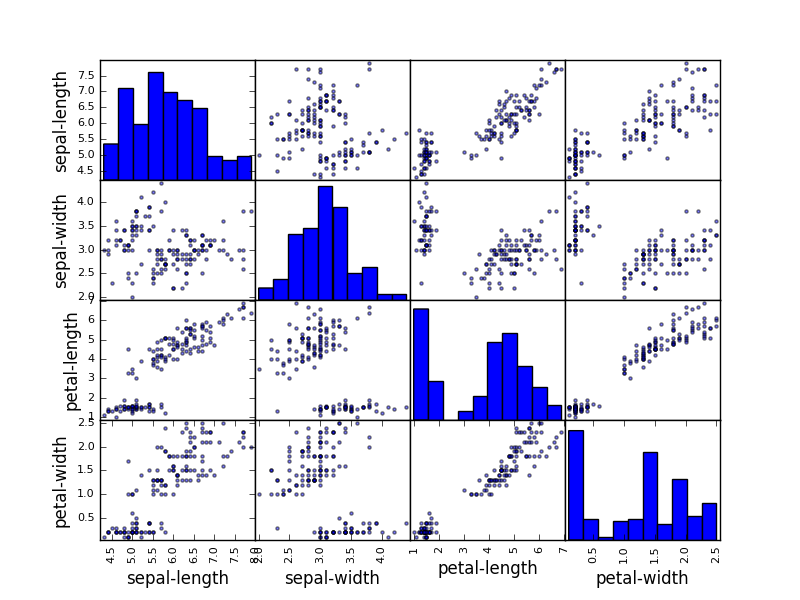
**4.2 Multivariate Plots**

Now we can look at the interactions between the variables.

First, let’s look at scatterplots of all pairs of attributes. This can be helpful to spot structured relationships between input variables.

|  |  |
| --- | --- |
| 1  2  3 | # scatter plot matrix  scatter\_matrix(dataset)  plt.show() |

Note the diagonal grouping of some pairs of attributes. This suggests a high correlation and a predictable relationship.



Scatterplot Matrix

**5. Evaluate Some Algorithms**

Now it is time to create some models of the data and estimate their accuracy on unseen data. Here is what we are going to cover in this step:

1. Separate out a validation dataset.
2. Set-up the test harness to use 10-fold cross validation.
3. Build 5 different models to predict species from flower measurements
4. Select the best model.

**5.1 Create a Validation Dataset**

We need to know that the model we created is any good.

Later, we will use statistical methods to estimate the accuracy of the models that we create on unseen data. We also want a more concrete estimate of the accuracy of the best model on unseen data by evaluating it on actual unseen data.

That is, we are going to hold back some data that the algorithms will not get to see and we will use this data to get a second and independent idea of how accurate the best model might actually be.

We will split the loaded dataset into two, 80% of which we will use to train our models and 20% that we will hold back as a validation dataset.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Split-out validation dataset  array = dataset.values  X = array[:,0:4]  Y = array[:,4]  validation\_size = 0.20  seed = 7  X\_train, X\_validation, Y\_train, Y\_validation = model\_selection.train\_test\_split(X, Y, test\_size=validation\_size, random\_state=seed) |

You now have training data in the *X\_train* and *Y\_train* for preparing models and a *X\_validation* and *Y\_validation* sets that we can use later.

**5.2 Test Harness**

We will use 10-fold cross validation to estimate accuracy.

This will split our dataset into 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

|  |  |
| --- | --- |
| 1  2  3 | # Test options and evaluation metric  seed = 7  scoring = 'accuracy' |

The specific random seed does not matter, learn more about pseudorandom number generators here:

* Introduction to Random Number Generators for Machine Learning in Python

We are using the metric of ‘*accuracy*‘ to evaluate models. This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the *scoring* variable when we run build and evaluate each model next.

**5.3 Build Models**

We don’t know which algorithms would be good on this problem or what configurations to use. We get an idea from the plots that some of the classes are partially linearly separable in some dimensions, so we are expecting generally good results.

Let’s evaluate 6 different algorithms:

* Logistic Regression (LR)
* Linear Discriminant Analysis (LDA)
* K-Nearest Neighbors (KNN).
* Classification and Regression Trees (CART).
* Gaussian Naive Bayes (NB).
* Support Vector Machines (SVM).

This is a good mixture of simple linear (LR and LDA), nonlinear (KNN, CART, NB and SVM) algorithms. We reset the random number seed before each run to ensure that the evaluation of each algorithm is performed using exactly the same data splits. It ensures the results are directly comparable.

Let’s build and evaluate our five models:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | # Spot Check Algorithms  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 = []  for name, model in models:  kfold = model\_selection.KFold(n\_splits=10, random\_state=seed)  cv\_results = model\_selection.cross\_val\_score(model, X\_train, Y\_train, cv=kfold, scoring=scoring)  results.append(cv\_results)  names.append(name)  msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())  print(msg) |

**5.4 Select Best Model**

We now have 6 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate.

Running the example above, we get the following raw results:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | LR: 0.966667 (0.040825)  LDA: 0.975000 (0.038188)  KNN: 0.983333 (0.033333)  CART: 0.975000 (0.038188)  NB: 0.975000 (0.053359)  SVM: 0.981667 (0.025000) |

Note, you’re results may differ. For more on this see the post:

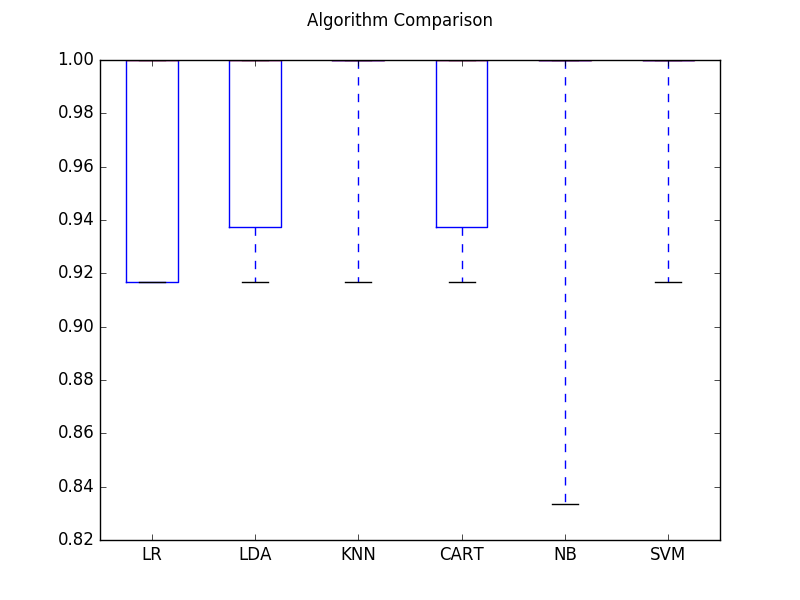
* [Embrace Randomness in Machine Learning](https://machinelearningmastery.com/randomness-in-machine-learning/)

We can see that it looks like KNN has the largest estimated accuracy score.

We can also create a plot of the model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (10 fold cross validation).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Compare Algorithms  fig = plt.figure()  fig.suptitle('Algorithm Comparison')  ax = fig.add\_subplot(111)  plt.boxplot(results)  ax.set\_xticklabels(names)  plt.show() |

You can see that the box and whisker plots are squashed at the top of the range, with many samples achieving 100% accuracy.



Compare Algorithm Accuracy

**6. Make Predictions**

The KNN algorithm was the most accurate model that we tested. Now we want to get an idea of the accuracy of the model on our validation set.

This will give us an independent final check on the accuracy of the best model. It is valuable to keep a validation set just in case you made a slip during training, such as overfitting to the training set or a data leak. Both will result in an overly optimistic result.

We can run the KNN model directly on the validation set and summarize the results as a final accuracy score, a [confusion matrix](http://machinelearningmastery.com/confusion-matrix-machine-learning/) and a classification report.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | # Make predictions on validation dataset  knn = KNeighborsClassifier()  knn.fit(X\_train, Y\_train)  predictions = knn.predict(X\_validation)  print(accuracy\_score(Y\_validation, predictions))  print(confusion\_matrix(Y\_validation, predictions))  print(classification\_report(Y\_validation, predictions)) |

We can see that the accuracy is 0.9 or 90%. The confusion matrix provides an indication of the three errors made. Finally, the classification report provides a breakdown of each class by precision, recall, f1-score and support showing excellent results (granted the validation dataset was small).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | 0.9    [[ 7  0  0]  [ 0 11  1]  [ 0  2  9]]                 precision    recall  f1-score   support    Iris-setosa       1.00      1.00      1.00         7  Iris-versicolor   0.85      0.92      0.88        12  Iris-virginica    0.90      0.82      0.86        11    avg / total       0.90      0.90      0.90        30 |