Machine Learning

The best small project to start with on a new tool is the classification of iris flowers. This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently.

Original link: <https://archive.ics.uci.edu/ml/datasets/Iris>

1. **Import Libraries**

Text, letter

Description automatically generated

1. **Load Data Set**

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

A picture containing text

Description automatically generated

1. **Check the Dataset**

print(dataset.shape) will show the number of instances (rows) and how many attributes (columns) the data contains with the shape property.



You should see 150 instances and 5 attributes.

We can use some of the code from previous modules to check the dataset – this is helpful for us to view the rows and columns.



1. **Summary of Data**

Do you remember how we check the count, mean, the min and max values?

Hint: we want to describe the data…..



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

****

**What does the answer from the class distribution tell us?**

We can see that each class has the same number of instances

1. **Data Visualization**

Given that the input variables are numeric, we can create box and whisker plots of each. (Univariate Plots) This gives us a much clearer idea of the distribution of the input attributes.

Logo

Description automatically generated

We can also create a histogram of each input variable to get an idea of the distribution. Have a go at creating some

Text

Description automatically generated

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.

1. **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.



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

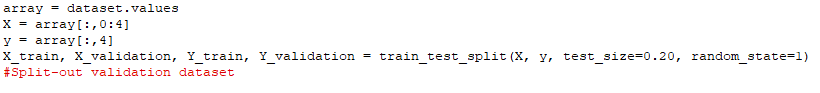
1. **Create a Validation Dataset**

Now it is time to create some models of the data and estimate their accuracy on unseen data. We need to know that the model we created is a good fit. 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 be.

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



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.

Notice that we used a python slice to select the columns in the NumPy array. This is a good tutorial explaining [how to index, slice and reshape NumPy arrays for machine learning.](https://machinelearningmastery.com/index-slice-reshape-numpy-arrays-machine-learning-python/)

1. **Test Harness**

We will use stratified 10-fold cross-validation to estimate model 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.

Stratified means that each fold or split of the dataset will aim to have the same distribution of example by class as exist in the whole training dataset. More information [on the k-fold cross-validation technique](https://machinelearningmastery.com/k-fold-cross-validation/).

We set the random seed via the *random\_state* argument to a fixed number to ensure that each algorithm is evaluated on the same splits of the training dataset.

The specific random seed does not matter. Further reading [Introduction to Random Number Generators for Machine Learning in Python](https://machinelearningmastery.com/introduction-to-random-number-generators-for-machine-learning/).

We are using the metric of ‘accuracy‘ to evaluate models.

This is a ratio of the number of correctly predicted instances 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.

1. **Build Models**

We don’t know which algorithms would be good for 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.

We will test 6 linear (LR and LDA) and nonlinear (KNN, CART, NB and SVM) 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).

build and evaluate our models:

**Graphical user interface, text, application

Description automatically generated**

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

Note: Your results may vary given the stochastic nature of the algorithm or evaluation procedure or differences in numerical precision. Consider running the example a few times and comparing the average outcome.

**What scores did you get?**

In my case, it looks like Support Vector Machines (SVM) has the largest estimated accuracy score at about 0.98 or 98%.

There is a population of accuracy measures for each algorithm because each algorithm was evaluated 10 times (via 10-fold-cross validation).

A useful way to compare the samples of results for each algorithm is to create a box and whisker plot for each distribution and compare the distributions.

Text

Description automatically generated

The model evaluation results compare the spread and the mean accuracy of each model. We can see that the box and whisker plots are squashed at the top of the range, with many evaluations achieving 100% accuracy, and some pushing down into the high 80% accuracies.

1. **Make Predictions**

We can fit the model on the entire training dataset and make predictions on the validation dataset.

Two additional reading areas are below on making predictions for single rows of data and how to save the model to file and load it later to make predictions on new data.

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

A picture containing text

Description automatically generated

1. **Evaluate Predictions**

We can evaluate the predictions by comparing them to the expected results in the validation set, then calculate classification accuracy, as well as a [confusion matrix\*](https://machinelearningmastery.com/confusion-matrix-machine-learning/)and a classification report.

Confusion Matrix

A confusion matrix is a way of presenting the results of a classifier in the context of what was really observed. Results are presented in a table showing the breakdown of the categorical outcome variable by value or level, comparing the frequency of observed values to the frequency of predicted values.

Often the observed frequencies are presented in columns and predicted frequencies are presented as rows, for example:

men women

men 3 1

women 2 4

Ideally, the predicted frequency would match the expected frequency and show number in the diagonal from the top left to the bottom right of the table and zeros everywhere else. A confusion matrix is useful as it allows you to quickly see the distribution of the types of errors made by a classifier on a classification predictive modelling problem.

Text

Description automatically generated

We can see that the accuracy is 0.966 or about 96% on the hold-out dataset.

The confusion matrix indicates the errors made.

Finally, the classification report provides a breakdown of each class by precision, recall, f1-score, and support showing excellent results (Although the validation dataset was small).

**This tutorial was taken from Jason Brownlee. He has further tutorials below and a published book. If you have time work through some of these other tutorials.**

Discover Python for machine learning

* + [A Gentle Introduction to Scikit-Learn: A Python Machine Learning Library](https://machinelearningmastery.com/a-gentle-introduction-to-scikit-learn-a-python-machine-learning-library/)

Discover the ecosystem for Python machine learning.

* + [Crash Course in Python for Machine Learning Developers](https://machinelearningmastery.com/crash-course-python-machine-learning-developers/)
  + [Python Ecosystem for Machine Learning](https://machinelearningmastery.com/python-ecosystem-machine-learning/)
  + [Python is the Growing Platform for Applied Machine Learning](https://machinelearningmastery.com/python-growing-platform-applied-machine-learning/)

Discover how to work through problems using machine learning in Python.

* + [Python Machine Learning Mini-Course](https://machinelearningmastery.com/python-machine-learning-mini-course/)