**<https://machinelearningmastery.com/calculate-feature-importance-with-python/>**

**Calculate Feature Importance**

**Tutorial Overview**

This tutorial is divided into six parts; they are:

1. Feature Importance
2. Preparation
   1. Check Scikit-Learn Version
   2. Test Datasets
3. Coefficients as Feature Importance
   1. Linear Regression Feature Importance
   2. Logistic Regression Feature Importance
4. Decision Tree Feature Importance
   1. CART Feature Importance
   2. Random Forest Feature Importance
   3. XGBoost Feature Importance
5. Permutation Feature Importance
   1. Permutation Feature Importance for Regression
   2. Permutation Feature Importance for Classification
6. Feature Selection with Importance

**Classification Dataset**

We will use the [make\_classification() function](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html) to create a test binary classification dataset.

The dataset will have 1,000 examples, with 10 input features, five of which will be informative and the remaining five will be redundant. We will fix the random number seed to ensure we get the same examples each time the code is run.

An example of creating and summarizing the dataset is listed below.

|  |  |
| --- | --- |
|  | # test classification dataset  from sklearn.datasets import make\_classification  # define dataset  X, y = make\_classification(n\_samples=1000, n\_features=10, n\_informative=5, n\_redundant=5, random\_state=1)  # summarize the dataset  print(X.shape, y.shape) |

Running the example creates the dataset and confirms the expected number of samples and features.

|  |  |
| --- | --- |
|  | (1000, 10) (1000,) |

**Regression Dataset**

We will use the [make\_regression() function](https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_regression.html) to create a test regression dataset.

Like the classification dataset, the regression dataset will have 1,000 examples, with 10 input features, five of which will be informative and the remaining five that will be redundant.

|  |  |
| --- | --- |
|  | # test regression dataset  from sklearn.datasets import make\_regression  X, y = make\_regression(n\_samples=1000, n\_features=10, n\_informative=5, random\_state=1)  print(X.shape, y.shape) |

Running the example creates the dataset and confirms the expected number of samples and features.

|  |  |
| --- | --- |
| 1 | (1000, 10) (1000,) |

**Coefficients as Feature Importance**

Examples include linear regression, logistic regression, and extensions that add regularization, such as ridge regression and the elastic net.

**Note**: Your [results may vary](https://machinelearningmastery.com/different-results-each-time-in-machine-learning/) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Data

from matplotlib import pyplot

# define dataset

X, y = make\_regression(n\_samples=1000, n\_features=10, n\_informative=5, random\_state=1)

**Linear Regression Feature Importance**

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
| # linear regression feature importance  from sklearn.datasets import make\_regression  from sklearn.linear\_model import LinearRegression  model = LinearRegression()  model.fit(X, y)  importance = model.coef\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.00000  Feature: 1, Score: 12.44483  Feature: 2, Score: -0.00000  Feature: 3, Score: -0.00000  Feature: 4, Score: 93.32225  Feature: 5, Score: 86.50811  Feature: 6, Score: 26.74607  Feature: 7, Score: 3.28535  Feature: 8, Score: -0.00000  Feature: 9, Score: 0.00000 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of Linear Regression Coefficients as Feature Importance Scores |

This approach may also be used with [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) and [ElasticNet](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html) models.

**Logistic Regression Feature Importance**

|  |  |
| --- | --- |
| # logistic regression for feature importance  from sklearn.datasets import make\_classification  from sklearn.linear\_model import LogisticRegression  model = LogisticRegression()  model.fit(X, y)  importance = model.coef\_[0]  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.16320  Feature: 1, Score: -0.64301  Feature: 2, Score: 0.48497  Feature: 3, Score: -0.46190  Feature: 4, Score: 0.18432  Feature: 5, Score: -0.11978  Feature: 6, Score: -0.40602  Feature: 7, Score: 0.03772  Feature: 8, Score: -0.51785  Feature: 9, Score: 0.26540 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of Logistic Regression Coefficients as Feature Importance Scores |

Recall this is a classification problem with classes 0 and 1. Notice that the coefficients are both positive and negative. The positive scores indicate a feature that predicts class 1, whereas the negative scores indicate a feature that predicts class 0.

No clear pattern of important and unimportant features can be identified from these results,

**Decision Tree Feature Importance**

Decision tree algorithms like [classification and regression trees](https://machinelearningmastery.com/implement-decision-tree-algorithm-scratch-python/) (CART) offer importance scores based on the reduction in the criterion used to select split points, like Gini or entropy.

This same approach can be used for ensembles of decision trees, such as the random forest and stochastic gradient boosting algorithms.

Let’s take a look at a worked example of each.

**CART Feature Importance**

We can use the CART algorithm for feature importance implemented in scikit-learn as the *DecisionTreeRegressor* and *DecisionTreeClassifier* classes.

**CART Regression Feature Importance**

The complete example of fitting a [DecisionTreeRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html) and summarizing the calculated feature importance scores is listed below.

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_regression  from sklearn.tree import DecisionTreeRegressor  model = LogisticRegression()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.00294  Feature: 1, Score: 0.00502  Feature: 2, Score: 0.00318  Feature: 3, Score: 0.00151  Feature: 4, Score: 0.51648  Feature: 5, Score: 0.43814  Feature: 6, Score: 0.02723  Feature: 7, Score: 0.00200  Feature: 8, Score: 0.00244  Feature: 9, Score: 0.00106 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of DecisionTreeRegressor Feature Importance Scores |

**CART Classification Feature Importance**

The complete example of fitting a [DecisionTreeClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) and summarizing the calculated feature importance scores is listed below.

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_classification  from sklearn.tree import DecisionTreeClassifier  model = DecisionTreeClassifier()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.01486  Feature: 1, Score: 0.01029  Feature: 2, Score: 0.18347  Feature: 3, Score: 0.30295  Feature: 4, Score: 0.08124  Feature: 5, Score: 0.00600  Feature: 6, Score: 0.19646  Feature: 7, Score: 0.02908  Feature: 8, Score: 0.12820  Feature: 9, Score: 0.04745 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of DecisionTreeClassifier Feature Importance Scores |

**Random Forest Feature Importance**

We can use the [Random Forest](https://machinelearningmastery.com/implement-random-forest-scratch-python/) algorithm for feature importance implemented in scikit-learn as the *RandomForestRegressor* and *RandomForestClassifier* classes.

After being fit, the model provides a *feature\_importances\_* property that can be accessed to retrieve the relative importance scores for each input feature.

This approach can also be used with the bagging and extra trees algorithms.

**Random Forest Regression Feature Importance**

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_regression  from sklearn.ensemble import RandomForestRegressor  model = RandomForestRegressor()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.00280  Feature: 1, Score: 0.00545  Feature: 2, Score: 0.00294  Feature: 3, Score: 0.00289  Feature: 4, Score: 0.52992  Feature: 5, Score: 0.42046  Feature: 6, Score: 0.02663  Feature: 7, Score: 0.00304  Feature: 8, Score: 0.00304  Feature: 9, Score: 0.00283 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of RandomForestRegressor Feature Importance Scores |

**Random Forest Classification Feature Importance**

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_classification  from sklearn.ensemble import RandomForestClassifier  model = RandomForestClassifier()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.06523  Feature: 1, Score: 0.10737  Feature: 2, Score: 0.15779  Feature: 3, Score: 0.20422  Feature: 4, Score: 0.08709  Feature: 5, Score: 0.09948  Feature: 6, Score: 0.10009  Feature: 7, Score: 0.04551  Feature: 8, Score: 0.08830  Feature: 9, Score: 0.04493 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of RandomForestClassifier Feature Importance Scores |

**XGBoost Feature Importance**

XGBoost is a library that provides an efficient and effective implementation of the stochastic gradient boosting algorithm.

This algorithm can be used with scikit-learn via the *XGBRegressor* and *XGBClassifier* classes.

After being fit, the model provides a *feature\_importances\_* property that can be accessed to retrieve the relative importance scores for each input feature.

This algorithm is also provided via scikit-learn via the *GradientBoostingClassifier* and *GradientBoostingRegressor* classes and the same approach to feature selection can be used.

First, install the XGBoost library, such as with pip:

|  |  |
| --- | --- |
|  | sudo pip install xgboost |

Then confirm that the library was installed correctly and works by checking the version number.

|  |  |
| --- | --- |
|  | # check xgboost version  import xgboost  print(xgboost.\_\_version\_\_) |

Running the example, you should see the following version number or higher.

|  |  |
| --- | --- |
| 1 | 0.90 |

For more on the XGBoost library, start here:

* [XGBoost with Python](https://machinelearningmastery.com/start-here/#xgboost)

**XGBoost Regression Feature Importance**

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_regression  from xgboost import XGBRegressor  model = XGBRegressor()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.00060  Feature: 1, Score: 0.01917  Feature: 2, Score: 0.00091  Feature: 3, Score: 0.00118  Feature: 4, Score: 0.49380  Feature: 5, Score: 0.42342  Feature: 6, Score: 0.05057  Feature: 7, Score: 0.00419  Feature: 8, Score: 0.00124  Feature: 9, Score: 0.00491 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of XGBRegressor Feature Importance Scores |

**XGBoost Classification Feature Importance**

The complete example of fitting an [XGBClassifier](https://xgboost.readthedocs.io/en/latest/python/python_api.html" \l "xgboost.XGBRFClassifier) and summarizing the calculated feature importance scores is listed below.

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_classification  from xgboost import XGBClassifier  model = XGBClassifier()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.02464  Feature: 1, Score: 0.08153  Feature: 2, Score: 0.12516  Feature: 3, Score: 0.28400  Feature: 4, Score: 0.12694  Feature: 5, Score: 0.10752  Feature: 6, Score: 0.08624  Feature: 7, Score: 0.04820  Feature: 8, Score: 0.09357  Feature: 9, Score: 0.02220 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of XGBClassifier Feature Importance Scores |

**Permutation Feature Importance**

[Permutation feature importance](https://scikit-learn.org/stable/modules/permutation_importance.html) is a technique for calculating relative importance scores that is independent of the model used.

First, a model is fit on the dataset, such as a model that does not support native feature importance scores. Then the model is used to make predictions on a dataset, although the values of a feature (column) in the dataset are scrambled. This is repeated for each feature in the dataset. Then this whole process is repeated 3, 5, 10 or more times. The result is a mean importance score for each input feature (and distribution of scores given the repeats).

This approach can be used for regression or classification and requires that a performance metric be chosen as the basis of the importance score, such as the mean squared error for regression and accuracy for classification.

Permutation feature selection can be used via the [permutation\_importance() function](https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html) that takes a fit model, a dataset (train or test dataset is fine), and a scoring function.

Let’s take a look at this approach to feature selection with an algorithm that does not support feature selection natively, specifically [k-nearest neighbors](https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/).

**Permutation Feature Importance for Regression**

The complete example of fitting a [KNeighborsRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html) and summarizing the calculated permutation feature importance scores is listed below.

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_regression  from sklearn.neighbors import KNeighborsRegressor  from sklearn.inspection import permutation\_importance  model = XGBClassifier()  model.fit(X, y)  importance = model.feature\_importances\_  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 175.52007  Feature: 1, Score: 345.80170  Feature: 2, Score: 126.60578  Feature: 3, Score: 95.90081  Feature: 4, Score: 9666.16446  Feature: 5, Score: 8036.79033  Feature: 6, Score: 929.58517  Feature: 7, Score: 139.67416  Feature: 8, Score: 132.06246  Feature: 9, Score: 84.94768 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of KNeighborsRegressor With Permutation Feature Importance Scores |

**Permutation Feature Importance for Classification**

The complete example of fitting a [KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html) and summarizing the calculated permutation feature importance scores is listed below.

|  |  |
| --- | --- |
| # decision tree for feature importance on a regression problem  from sklearn.datasets import make\_classification  from sklearn.neighbors import KNeighborsClassifier  from sklearn.inspection import permutation\_importance  model = KNeighborsClassifier()  model.fit(X, y)  # perform permutation importance  results = permutation\_importance(model, X, y, scoring='accuracy')  # get importance  importance = results.importances\_mean  for i,v in enumerate(importance):  print('Feature: %0d, Score: %.5f' % (i,v)) | Feature: 0, Score: 0.04760  Feature: 1, Score: 0.06680  Feature: 2, Score: 0.05240  Feature: 3, Score: 0.09300  Feature: 4, Score: 0.05140  Feature: 5, Score: 0.05520  Feature: 6, Score: 0.07920  Feature: 7, Score: 0.05560  Feature: 8, Score: 0.05620  Feature: 9, Score: 0.03080 |
| # plot feature importance  pyplot.bar([x for x in range(len(importance))], importance)  pyplot.show() | Bar Chart of KNeighborsClassifier With Permutation Feature Importance Scores |

**Feature Selection with Importance**

Feature importance scores can be used to help interpret the data, but they can also be used directly to help rank and select features that are most useful to a predictive model.

Recall, our synthetic dataset has 1,000 examples each with 10 input variables, five of which are redundant and five of which are important to the outcome. We can use feature importance scores to help select the five variables that are relevant and only use them as inputs to a predictive model.

First, we can split the training dataset into train and test sets and train a model on the training dataset, make predictions on the test set and evaluate the result using classification accuracy. We will use a logistic regression model as the predictive model.

|  |  |
| --- | --- |
|  | # evaluation of a model using all features  from sklearn.datasets import make\_classification  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score  # define the dataset  X, y = make\_classification(n\_samples=1000, n\_features=10, n\_informative=5, n\_redundant=5, random\_state=1)  # split into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=1)  # fit the model  model = LogisticRegression(solver='liblinear')  model.fit(X\_train, y\_train)  # evaluate the model  yhat = model.predict(X\_test)  # evaluate predictions  accuracy = accuracy\_score(y\_test, yhat)  print('Accuracy: %.2f' % (accuracy\*100)) |

In this case we can see that the model achieved the classification accuracy of about 84.55 percent using all features in the dataset.

|  |  |
| --- | --- |
|  | Accuracy: 84.55 |

Given that we created the dataset, we would expect better or the same results with half the number of input variables.

We could use any of the feature importance scores explored above, but in this case we will use the feature importance scores provided by random forest.

We can use the [SelectFromModel](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html) class to define both the model we wish to calculate importance scores, [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) in this case, and the number of features to select, 5 in this case.

|  |  |
| --- | --- |
|  | ...  # configure to select a subset of features  fs = SelectFromModel(RandomForestClassifier(n\_estimators=200), max\_features=5) |

We can fit the feature selection method on the training dataset.

This will calculate the importance scores that can be used to rank all input features. We can then apply the method as a transform to select a subset of 5 most important features from the dataset. This transform will be applied to the training dataset and the test set.

|  |  |
| --- | --- |
|  | ...  # learn relationship from training data  fs.fit(X\_train, y\_train)  # transform train input data  X\_train\_fs = fs.transform(X\_train)  # transform test input data  X\_test\_fs = fs.transform(X\_test) |

Tying this all together, the complete example of using random forest feature importance for feature selection is listed below.

|  |  |
| --- | --- |
|  | # evaluation of a model using 5 features chosen with random forest importance  from sklearn.datasets import make\_classification  from sklearn.model\_selection import train\_test\_split  from sklearn.feature\_selection import SelectFromModel  from sklearn.ensemble import RandomForestClassifier  from sklearn.linear\_model import LogisticRegression  from sklearn.metrics import accuracy\_score    # feature selection  def select\_features(X\_train, y\_train, X\_test):  # configure to select a subset of features  fs = SelectFromModel(RandomForestClassifier(n\_estimators=1000), max\_features=5)  # learn relationship from training data  fs.fit(X\_train, y\_train)  # transform train input data  X\_train\_fs = fs.transform(X\_train)  # transform test input data  X\_test\_fs = fs.transform(X\_test)  return X\_train\_fs, X\_test\_fs, fs    # define the dataset  X, y = make\_classification(n\_samples=1000, n\_features=10, n\_informative=5, n\_redundant=5, random\_state=1)  # split into train and test sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, random\_state=1)  # feature selection  X\_train\_fs, X\_test\_fs, fs = select\_features(X\_train, y\_train, X\_test)  # fit the model  model = LogisticRegression(solver='liblinear')  model.fit(X\_train\_fs, y\_train)  # evaluate the model  yhat = model.predict(X\_test\_fs)  # evaluate predictions  accuracy = accuracy\_score(y\_test, yhat)  print('Accuracy: %.2f' % (accuracy\*100)) |

Running the example first performs feature selection on the dataset, then fits and evaluates the logistic regression model as before.

**Note**: Your [results may vary](https://machinelearningmastery.com/different-results-each-time-in-machine-learning/) given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

In this case, we can see that the model achieves the same performance on the dataset, although with half the number of input features. As expected, the feature importance scores calculated by random forest allowed us to accurately rank the input features and delete those that were not relevant to the target variable.

|  |  |
| --- | --- |
|  | Accuracy: 84.55 |