Feature Importance

20 October 2023 22:24

Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction.
Feature importance scores can be calculated for problems that involve predicting a numerical value, called regression, and those problems that involve predicting a class label, called classification.
The scores are useful and can be used in a range of situations in a predictive modeling problem, such as:

- Better understanding the data.
- Better understanding a model.
- Reducing the number of input features.

Feature importance scores can provide insight into the dataset. The relative scores can highlight which features may be most relevant to the target, and the converse, which features are the least relevant. This may be interpreted by a domain expert and could be used as the basis for gathering more or different data.

Feature importance scores can provide insight into the model. Most importance scores are calculated by a predictive model that has been fit on the dataset. Inspecting the importance score provides insight into that specific model and which features are the most important and least important to the model when making a prediction. This is a type of model interpretation that can be performed for those models that support it.

Feature importance can be used to improve a predictive model. This can be achieved by using the importance scores to select those features to delete (lowest scores) or those features to keep (highest scores). This is a type of feature selection and can simplify the problem that is being modeled, speed up the modeling process (deleting features is called dimensionality reduction), and in some cases, improve the performance of the model.

Test Datasets

Next, let's define some test datasets that we can use as the basis for demonstrating and exploring feature importance scores.
Each test problem has five important and five unimportant features, and it may be interesting to see which methods are consistent at finding or differentiating the features based on their importance.

Classification Dataset

We will use the make_classification() function 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 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
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=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,)
```

Next, let's take a closer look at coefficients as importance scores.

Coefficients as Feature Importance

Linear machine learning algorithms fit a model where the prediction is the weighted sum of the input values.
Examples include linear regression, logistic regression, and extensions that add regularization, such as ridge regression and the elastic net.
All of these algorithms find a set of coefficients to use in the weighted sum in order to make a prediction. These coefficients can be used directly as a crude type of feature importance score.
Let's take a closer look at using coefficients as feature importance for classification and regression. We will fit a model on the dataset to find the coefficients, then summarize the importance scores for each input feature and finally create a bar chart to get an idea of the relative importance of the features.

Linear Regression Feature Importance

We can fit a LinearRegression model on the regression dataset and retrieve the *coeff_* property that contains the coefficients found for each input variable.
These coefficients can provide the basis for a crude feature importance score. This assumes that the input variables have the same scale or have been scaled prior to fitting a model.
The complete example of linear regression coefficients for feature importance is listed below.

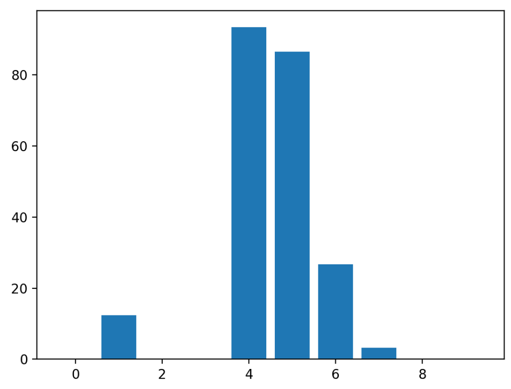
```
# linear regression feature importance
from sklearn.datasets import make_regression
from sklearn.linear_model import LinearRegression
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
```

```
model = LinearRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.
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 compare the average outcome.
The scores suggest that the model found the five important features and marked all other features with a zero coefficient, essentially removing them from the model.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of Linear Regression Coefficients as Feature Importance Scores
This approach may also be used with Ridge and ElasticNet models.

Logistic Regression Feature Importance

We can fit a LogisticRegression model on the regression dataset and retrieve the *coeff_* property that contains the coefficients found for each input variable.
These coefficients can provide the basis for a crude feature importance score. This assumes that the input variables have the same scale or have been scaled prior to fitting a model.
The complete example of logistic regression coefficients for feature importance is listed below.

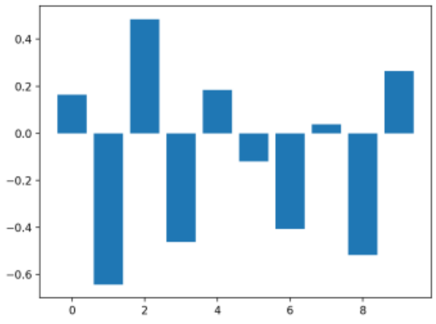
```
# logistic regression for feature importance
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)
# define the model
model = LogisticRegression()
# fit the model
model.fit(X, y)
# get importance
importance = model.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.
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 compare the average outcome.
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, at least from what I can tell.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of Logistic Regression Coefficients as Feature Importance Scores

Now that we have seen the use of coefficients as importance scores, let's look at the more common example of decision-tree-based importance scores.

Decision Tree Feature Importance

Decision tree algorithms like classification and regression trees (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.

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

Let's take a look at an example of this for regression and classification.

CART Regression Feature Importance

The complete example of fitting a DecisionTreeRegressor 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
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = DecisionTreeRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

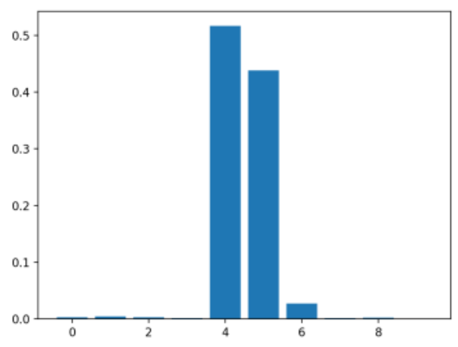
Running the example fits the model, then reports the coefficient value for each feature.

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 compare the average outcome.

The results suggest perhaps three of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of DecisionTreeRegressor Feature Importance Scores

CART Classification Feature Importance

The complete example of fitting a DecisionTreeClassifier and summarizing the calculated feature importance scores is listed below.

```
# decision tree for feature importance on a classification problem
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)
# define the model
model = DecisionTreeClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

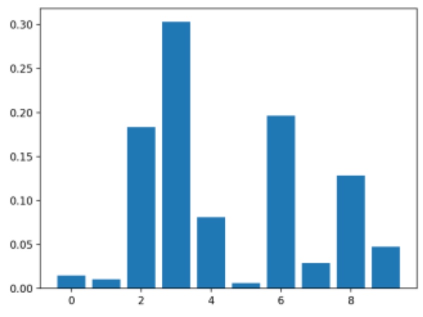
Running the example fits the model, then reports the coefficient value for each feature.

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 compare the average outcome.

The results suggest perhaps four of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of DecisionTreeClassifier Feature Importance Scores

Random Forest Feature Importance

We can use the Random Forest 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.

Let's take a look at an example of this for regression and classification.

Random Forest Regression Feature Importance

The complete example of fitting a RandomForestRegressor and summarizing the calculated feature importance scores is listed below.

```
# random forest for feature importance on a regression problem
from sklearn.datasets import make_regression
from sklearn.ensemble import RandomForestRegressor
from matplotlib import pyplot
```

```
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = RandomForestRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

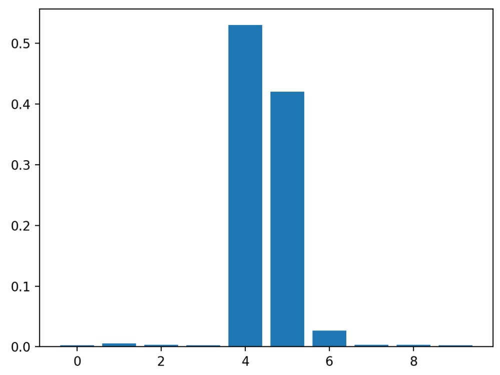
Running the example fits the model, then reports the coefficient value for each feature.

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 compare the average outcome.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of RandomForestRegressor Feature Importance Scores

Random Forest Classification Feature Importance

The complete example of fitting a RandomForestClassifier and summarizing the calculated feature importance scores is listed below.

```
# random forest for feature importance on a classification problem
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)
# define the model
model = RandomForestClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

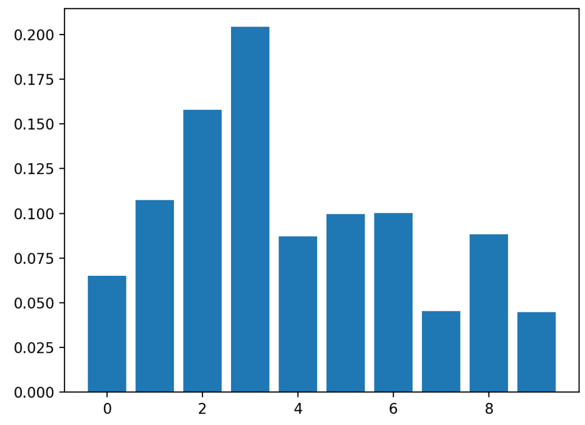
Running the example fits the model, then reports the coefficient value for each feature.

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 compare the average outcome.

The results suggest perhaps two or three of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



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.

```
0.90
```

For more on the XGBoost library, start here:

- XGBoost with Python

Let's take a look at an example of XGBoost for feature importance on regression and classification problems.

XGBoost Regression Feature Importance

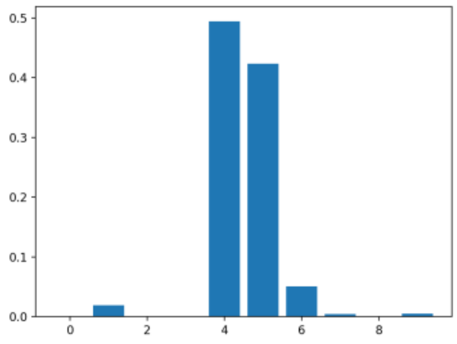
The complete example of fitting a XGBRegressor and summarizing the calculated feature importance scores is listed below.

```
# xgboost for feature importance on a regression problem
from sklearn.datasets import make_regression
from xgboost import XGBRegressor
from matplotlib import pyplot
# define dataset
X, y = make_regression(n_samples=1000, n_features=10, n_informative=5, random_state=1)
# define the model
model = XGBRegressor()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

Running the example fits the model, then reports the coefficient value for each feature.
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 compare the average outcome.
The results suggest perhaps two or three of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



Bar Chart of XGBRegressor Feature Importance Scores

XGBoost Classification Feature Importance

The complete example of fitting an XGBClassifier and summarizing the calculated feature importance scores is listed below.

```
# xgboost for feature importance on a classification problem
from sklearn.datasets import make_classification
from xgboost import XGBClassifier
from matplotlib import pyplot
# define dataset
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_redundant=5, random_state=1)
# define the model
model = XGBClassifier()
# fit the model
model.fit(X, y)
# get importance
importance = model.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
 print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

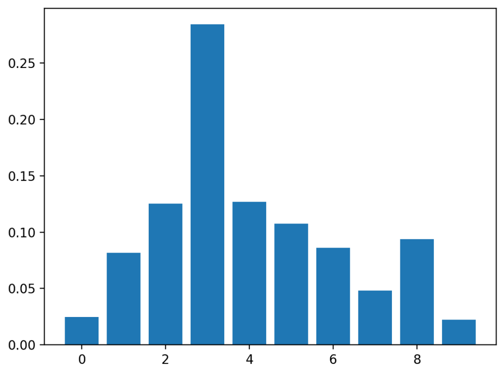
Running the example fits the model then reports the coefficient value for each feature.

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 compare the average outcome.

The results suggest perhaps seven of the 10 features as being important to prediction.

```
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
```

A bar chart is then created for the feature importance scores.



➢