Chapter 9

Evaluate the Performance of Machine Learning Algorithms with Resampling

You need to know how well your algorithms perform on unseen data. The best way to evaluate the performance of an algorithm would be to make predictions for new data to which you already know the answers. The second best way is to use clever techniques from statistics called resampling methods that allow you to make accurate estimates for how well your algorithm will perform on new data. In this chapter you will discover how you can estimate the accuracy of your machine learning algorithms using resampling methods in Python and scikit-learn on the Pima Indians dataset. Let's get started.

9.1 Evaluate Machine Learning Algorithms

Why can't you train your machine learning algorithm on your dataset and use predictions from this same dataset to evaluate machine learning algorithms? The simple answer is overfitting.

Imagine an algorithm that remembers every observation it is shown during training. If you evaluated your machine learning algorithm on the same dataset used to train the algorithm, then an algorithm like this would have a perfect score on the training dataset. But the predictions it made on new data would be terrible. We must evaluate our machine learning algorithms on data that is not used to train the algorithm.

The evaluation is an estimate that we can use to talk about how well we think the algorithm may actually do in practice. It is not a guarantee of performance. Once we estimate the performance of our algorithm, we can then re-train the final algorithm on the entire training dataset and get it ready for operational use. Next up we are going to look at four different techniques that we can use to split up our training dataset and create useful estimates of performance for our machine learning algorithms:

- Train and Test Sets.
- k-fold Cross Validation.
- Leave One Out Cross Validation.
- Repeated Random Test-Train Splits.

9.2 Split into Train and Test Sets

The simplest method that we can use to evaluate the performance of a machine learning algorithm is to use different training and testing datasets. We can take our original dataset and split it into two parts. Train the algorithm on the first part, make predictions on the second part and evaluate the predictions against the expected results. The size of the split can depend on the size and specifics of your dataset, although it is common to use 67% of the data for training and the remaining 33% for testing.

This algorithm evaluation technique is very fast. It is ideal for large datasets (millions of records) where there is strong evidence that both splits of the data are representative of the underlying problem. Because of the speed, it is useful to use this approach when the algorithm you are investigating is slow to train. A downside of this technique is that it can have a high variance. This means that differences in the training and test dataset can result in meaningful differences in the estimate of accuracy. In the example below we split the Pima Indians dataset into 67%/33% splits for training and test and evaluate the accuracy of a Logistic Regression model.

```
# Evaluate using a train and a test set
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
result = model.score(X_test, Y_test)
print("Accuracy: %.3f%%") % (result*100.0)
```

Listing 9.1: Example of evaluating an algorithm with a train and test set.

We can see that the estimated accuracy for the model was approximately 75%. Note that in addition to specifying the size of the split, we also specify the random seed. Because the split of the data is random, we want to ensure that the results are reproducible. By specifying the random seed we ensure that we get the same random numbers each time we run the code and in turn the same split of data. This is important if we want to compare this result to the estimated accuracy of another machine learning algorithm or the same algorithm with a different configuration. To ensure the comparison was apples-for-apples, we must ensure that they are trained and tested on exactly the same data.

```
Accuracy: 75.591%
```

Listing 9.2: Output of evaluating an algorithm with a train and test set.

9.3 K-fold Cross Validation

Cross validation is an approach that you can use to estimate the performance of a machine learning algorithm with less variance than a single train-test set split. It works by splitting the dataset into k-parts (e.g. $k = 5$ or $k = 10$). Each split of the data is called a fold. The algorithm is trained on $k - 1$ folds with one held back and tested on the held back fold. This is repeated so that each fold of the dataset is given a chance to be the held back test set. After running cross validation you end up with k different performance scores that you can summarize using a mean and a standard deviation.

The result is a more reliable estimate of the performance of the algorithm on new data. It is more accurate because the algorithm is trained and evaluated multiple times on different data. The choice of k must allow the size of each test partition to be large enough to be a reasonable sample of the problem, whilst allowing enough repetitions of the train-test evaluation of the algorithm to provide a fair estimate of the algorithms performance on unseen data. For modest sized datasets in the thousands or tens of thousands of records, k values of 3, 5 and 10 are common. In the example below we use 10-fold cross validation.

```
# Evaluate using Cross Validation
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
seed = 7
kfold = KFold(n_splits=num_folds, random_state=seed)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0)
```

Listing 9.3: Example of evaluating an algorithm with k-fold Cross Validation.

You can see that we report both the mean and the standard deviation of the performance measure. When summarizing performance measures, it is a good practice to summarize the distribution of the measures, in this case assuming a Gaussian distribution of performance (a very reasonable assumption) and recording the mean and standard deviation.

```
Accuracy: 76.951% (4.841%)
```

Listing 9.4: Output of evaluating an algorithm with k-fold Cross Validation.

9.4 Leave One Out Cross Validation

You can configure cross validation so that the size of the fold is 1 (k is set to the number of observations in your dataset). This variation of cross validation is called leave-one-out cross validation. The result is a large number of performance measures that can be summarized in

an effort to give a more reasonable estimate of the accuracy of your model on unseen data. A downside is that it can be a computationally more expensive procedure than k-fold cross validation. In the example below we use leave-one-out cross validation.

```
# Evaluate using Leave One Out Cross Validation
from pandas import read_csv
from sklearn.model_selection import LeaveOneOut
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
num_folds = 10
loocv = LeaveOneOut()
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=loocv)
print("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0)
```

Listing 9.5: Example of evaluating an algorithm with Leave One Out Cross Validation.

You can see in the standard deviation that the score has more variance than the k-fold cross validation results described above.

```
Accuracy: 76.823% (42.196%)
```

Listing 9.6: Output of evaluating an algorithm with Leave One Out Cross Validation.

9.5 Repeated Random Test-Train Splits

Another variation on k-fold cross validation is to create a random split of the data like the train/test split described above, but repeat the process of splitting and evaluation of the algorithm multiple times, like cross validation. This has the speed of using a train/test split and the reduction in variance in the estimated performance of k-fold cross validation. You can also repeat the process many more times as needed to improve the accuracy. A down side is that repetitions may include much of the same data in the train or the test split from run to run, introducing redundancy into the evaluation. The example below splits the data into a 67%/33% train/test split and repeats the process 10 times.

```
# Evaluate using Shuffle Split Cross Validation
from pandas import read_csv
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
n_splits = 10
test_size = 0.33
```

```
seed = 7
kfold = ShuffleSplit(n_splits=n_splits, test_size=test_size, random_state=seed)
model = LogisticRegression()
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)") % (results.mean()*100.0, results.std()*100.0)
```

Listing 9.7: Example of evaluating an algorithm with Shuffle Split Cross Validation.

We can see that in this case the distribution of the performance measure is on par with k-fold cross validation above.

```
Accuracy: 76.496% (1.698%)
```

Listing 9.8: Output of evaluating an algorithm with Shuffle Split Cross Validation.

9.6 What Techniques to Use When

This section lists some tips to consider what resampling technique to use in different circumstances.

- Generally k-fold cross validation is the *gold standard* for evaluating the performance of a machine learning algorithm on unseen data with k set to 3, 5, or 10.
- Using a train/test split is good for speed when using a slow algorithm and produces performance estimates with lower bias when using large datasets.
- Techniques like leave-one-out cross validation and repeated random splits can be useful intermediates when trying to balance variance in the estimated performance, model training speed and dataset size.

The best advice is to experiment and find a technique for your problem that is fast and produces reasonable estimates of performance that you can use to make decisions. If in doubt, use 10-fold cross validation.

9.7 Summary

In this chapter you discovered statistical techniques that you can use to estimate the performance of your machine learning algorithms, called resampling. Specifically, you learned about:

- Train and Test Sets.
- Cross Validation.
- Leave One Out Cross Validation.
- Repeated Random Test-Train Splits.

9.7.1 Next

In the next section you will learn how you can evaluate the performance of classification and regression algorithms using a suite of different metrics and built in evaluation reports.

Chapter 10

Machine Learning Algorithm Performance Metrics

The metrics that you choose to evaluate your machine learning algorithms are very important. Choice of metrics influences how the performance of machine learning algorithms is measured and compared. They influence how you weight the importance of different characteristics in the results and your ultimate choice of which algorithm to choose. In this chapter you will discover how to select and use different machine learning performance metrics in Python with scikit-learn. Let's get started.

10.1 Algorithm Evaluation Metrics

In this lesson, various different algorithm evaluation metrics are demonstrated for both classification and regression type machine learning problems. In each recipe, the dataset is downloaded directly from the UCI Machine Learning repository.

- For classification metrics, the Pima Indians onset of diabetes dataset is used as demonstration. This is a binary classification problem where all of the input variables are numeric.
- For regression metrics, the Boston House Price dataset is used as demonstration. this is a regression problem where all of the input variables are also numeric.

All recipes evaluate the same algorithms, Logistic Regression for classification and Linear Regression for the regression problems. A 10-fold cross validation test harness is used to demonstrate each metric, because this is the most likely scenario you will use when employing different algorithm evaluation metrics.

A caveat in these recipes is the `cross_validation.cross_val_score` function[1] used to report the performance in each recipe. It does allow the use of different scoring metrics that will be discussed, but all scores are reported so that they can be sorted in ascending order (largest score is best). Some evaluation metrics (like mean squared error) are naturally descending scores (the smallest score is best) and as such are reported as negative by the

[1] http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.cross_val_score.html

`cross_validation.cross_val_score()` function. This is important to note, because some scores will be reported as negative that by definition can never be negative. I will remind you about this caveat as we work through the lesson.

You can learn more about machine learning algorithm performance metrics supported by scikit-learn on the page *Model evaluation: quantifying the quality of predictions*[2]. Let's get on with the evaluation metrics.

10.2 Classification Metrics

Classification problems are perhaps the most common type of machine learning problem and as such there are a myriad of metrics that can be used to evaluate predictions for these problems. In this section we will review how to use the following metrics:

- Classification Accuracy.
- Logarithmic Loss.
- Area Under ROC Curve.
- Confusion Matrix.
- Classification Report.

10.2.1 Classification Accuracy

Classification accuracy is the number of correct predictions made as a ratio of all predictions made. This is the most common evaluation metric for classification problems, it is also the most misused. It is really only suitable when there are an equal number of observations in each class (which is rarely the case) and that all predictions and prediction errors are equally important, which is often not the case. Below is an example of calculating classification accuracy.

```
# Cross Validation Classification Accuracy
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'accuracy'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Accuracy: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.1: Example of evaluating an algorithm by classification accuracy.

[2] http://scikit-learn.org/stable/modules/model_evaluation.html

You can see that the ratio is reported. This can be converted into a percentage by multiplying the value by 100, giving an accuracy score of approximately 77% accurate.

```
Accuracy: 0.770 (0.048)
```

Listing 10.2: Output of evaluating an algorithm by classification accuracy.

10.2.2 Logarithmic Loss

Logarithmic loss (or logloss) is a performance metric for evaluating the predictions of probabilities of membership to a given class. The scalar probability between 0 and 1 can be seen as a measure of confidence for a prediction by an algorithm. Predictions that are correct or incorrect are rewarded or punished proportionally to the confidence of the prediction. Below is an example of calculating logloss for Logistic regression predictions on the Pima Indians onset of diabetes dataset.

```
# Cross Validation Classification LogLoss
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'neg_log_loss'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("Logloss: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.3: Example of evaluating an algorithm by logloss.

Smaller logloss is better with 0 representing a perfect logloss. As mentioned above, the measure is inverted to be ascending when using the `cross_val_score()` function.

```
Logloss: -0.493 (0.047)
```

Listing 10.4: Output of evaluating an algorithm by logloss.

10.2.3 Area Under ROC Curve

Area under ROC Curve (or AUC for short) is a performance metric for binary classification problems. The AUC represents a model's ability to discriminate between positive and negative classes. An area of 1.0 represents a model that made all predictions perfectly. An area of 0.5 represents a model that is as good as random. ROC can be broken down into sensitivity and specificity. A binary classification problem is really a trade-off between sensitivity and specificity.

- Sensitivity is the true positive rate also called the recall. It is the number of instances from the positive (first) class that actually predicted correctly.

- Specificity is also called the true negative rate. Is the number of instances from the negative (second) class that were actually predicted correctly.

The example below provides a demonstration of calculating AUC.

```
# Cross Validation Classification ROC AUC
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
kfold = KFold(n_splits=10, random_state=7)
model = LogisticRegression()
scoring = 'roc_auc'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("AUC: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.5: Example of evaluating an algorithm by AUC.

You can see the AUC is relatively close to 1 and greater than 0.5, suggesting some skill in the predictions

```
AUC: 0.824 (0.041)
```

Listing 10.6: Output of evaluating an algorithm by AUC.

10.2.4 Confusion Matrix

The confusion matrix is a handy presentation of the accuracy of a model with two or more classes. The table presents predictions on the x-axis and accuracy outcomes on the y-axis. The cells of the table are the number of predictions made by a machine learning algorithm. For example, a machine learning algorithm can predict 0 or 1 and each prediction may actually have been a 0 or 1. Predictions for 0 that were actually 0 appear in the cell for $prediction = 0$ and $actual = 0$, whereas predictions for 0 that were actually 1 appear in the cell for $prediction = 0$ and $actual = 1$. And so on. Below is an example of calculating a confusion matrix for a set of predictions by a Logistic Regression on the Pima Indians onset of diabetes dataset.

```
# Cross Validation Classification Confusion Matrix
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
```

```
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
matrix = confusion_matrix(Y_test, predicted)
print(matrix)
```

Listing 10.7: Example of evaluating an algorithm by confusion matrix.

Although the array is printed without headings, you can see that the majority of the predictions fall on the diagonal line of the matrix (which are correct predictions).

```
[[141  21]
 [ 41  51]]
```

Listing 10.8: Output of evaluating an algorithm by confusion matrix.

10.2.5 Classification Report

The scikit-learn library provides a convenience report when working on classification problems to give you a quick idea of the accuracy of a model using a number of measures. The `classification_report()` function displays the precision, recall, F1-score and support for each class. The example below demonstrates the report on the binary classification problem.

```
# Cross Validation Classification Report
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
filename = 'pima-indians-diabetes.data.csv'
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(filename, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
test_size = 0.33
seed = 7
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_size,
    random_state=seed)
model = LogisticRegression()
model.fit(X_train, Y_train)
predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print(report)
```

Listing 10.9: Example of evaluating an algorithm by classification report.

You can see good prediction and recall for the algorithm.

```
             precision    recall  f1-score   support

        0.0       0.77      0.87      0.82       162
        1.0       0.71      0.55      0.62        92
```

```
avg / total      0.75      0.76      0.75       254
```

Listing 10.10: Output of evaluating an algorithm by classification report.

10.3 Regression Metrics

In this section will review 3 of the most common metrics for evaluating predictions on regression machine learning problems:

- Mean Absolute Error.
- Mean Squared Error.
- R^2.

10.3.1 Mean Absolute Error

The Mean Absolute Error (or MAE) is the sum of the absolute differences between predictions and actual values. It gives an idea of how wrong the predictions were. The measure gives an idea of the magnitude of the error, but no idea of the direction (e.g. over or under predicting). The example below demonstrates calculating mean absolute error on the Boston house price dataset.

```
# Cross Validation Regression MAE
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO',
    'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
kfold = KFold(n_splits=10, random_state=7)
model = LinearRegression()
scoring = 'neg_mean_absolute_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MAE: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.11: Example of evaluating an algorithm by Mean Absolute Error.

A value of 0 indicates no error or perfect predictions. Like logloss, this metric is inverted by the `cross_val_score()` function.

```
MAE: -4.005 (2.084)
```

Listing 10.12: Output of evaluating an algorithm by Mean Absolute Error.

10.3.2 Mean Squared Error

The Mean Squared Error (or MSE) is much like the mean absolute error in that it provides a gross idea of the magnitude of error. Taking the square root of the mean squared error converts the units back to the original units of the output variable and can be meaningful for description and presentation. This is called the Root Mean Squared Error (or RMSE). The example below provides a demonstration of calculating mean squared error.

```
# Cross Validation Regression MSE
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO',
    'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
num_folds = 10
kfold = KFold(n_splits=10, random_state=7)
model = LinearRegression()
scoring = 'neg_mean_squared_error'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("MSE: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.13: Example of evaluating an algorithm by Mean Squared Error.

This metric too is inverted so that the results are increasing. Remember to take the absolute value before taking the square root if you are interested in calculating the RMSE.

```
MSE: -34.705 (45.574)
```

Listing 10.14: Output of evaluating an algorithm by Mean Squared Error.

10.3.3 R^2 Metric

The R^2 (or R Squared) metric provides an indication of the goodness of fit of a set of predictions to the actual values. In statistical literature this measure is called the coefficient of determination. This is a value between 0 and 1 for no-fit and perfect fit respectively. The example below provides a demonstration of calculating the mean R^2 for a set of predictions.

```
# Cross Validation Regression R^2
from pandas import read_csv
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
filename = 'housing.csv'
names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX', 'PTRATIO',
    'B', 'LSTAT', 'MEDV']
dataframe = read_csv(filename, delim_whitespace=True, names=names)
array = dataframe.values
X = array[:,0:13]
Y = array[:,13]
```

```
kfold = KFold(n_splits=10, random_state=7)
model = LinearRegression()
scoring = 'r2'
results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)
print("R^2: %.3f (%.3f)") % (results.mean(), results.std())
```

Listing 10.15: Example of evaluating an algorithm by R Squared.

You can see the predictions have a poor fit to the actual values with a value closer to zero and less than 0.5.

```
R^2: 0.203 (0.595)
```

Listing 10.16: Output of evaluating an algorithm by R Squared.

10.4 Summary

In this chapter you discovered metrics that you can use to evaluate your machine learning algorithms.

You learned about three classification metrics: Accuracy, Logarithmic Loss and Area Under ROC Curve. You also learned about two convenience methods for classification prediction results: the Confusion Matrix and the Classification Report. Finally, you also learned about three metrics for regression problems: Mean Absolute Error, Mean Squared Error and R^2.

10.4.1 Next

You now know how to evaluate the performance of machine learning algorithms using a variety of different metrics and how to use those metrics to estimate the performance of algorithms on new unseen data using resampling. In the next lesson you will start looking at machine learning algorithms themselves, starting with classification techniques.