How to Tune Algorithm Parameters with Scikit-Learn

by **Jason Brownlee** on July 16, 2014 in **Python Machine Learning**

Last Updated on August 21, 2019

Machine learning models are parameterized so that their behavior can be tuned for a given problem.

Models can have many parameters and finding the best combination of parameters can be treated as a search problem.

In this post, you will discover how to tune the parameters of machine learning algorithms in Python using the scikit-learn library.

Discover how to prepare data with pandas, fit and evaluate models with scikit-learn, and more in my new book, with 16 step-by-step tutorials, 3 projects, and full python code.

Let's get started.

- **Update Jan/2017**: Updated to reflect changes to the scikit-learn API in version 0.18.

Tuning an algorithm like Tuning a Piano
Photo by Katie Fricker, some rights reserved

Machine Learning Algorithm Parameters

Algorithm tuning is a final step in the process of applied machine learning before presenting results.

It is sometimes called Hyperparameter optimization where the algorithm parameters are referred to as hyperparameters whereas the coefficients found by the machine learning algorithm itself are referred to as parameters. Optimization suggests the search-nature of the problem.

Phrased as a search problem, you can use different search strategies to find a good and robust parameter or set of parameters for an algorithm on a given problem.

Two simple and easy search strategies are grid search and random search. Scikit-learn provides these two methods for algorithm parameter tuning and examples of each are provided below.

Grid Search Parameter Tuning

Grid search is an approach to parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

The recipe below evaluates different alpha values for the Ridge Regression algorithm on the standard diabetes dataset. This is a one-dimensional grid search.

```
# Grid Search for Algorithm Tuning
```

```
import numpy as np
from sklearn import datasets
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV
# load the diabetes datasets
dataset = datasets.load_diabetes()
# prepare a range of alpha values to test
alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
# create and fit a ridge regression model, testing each alpha
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=dict(alpha=alphas))
grid.fit(dataset.data, dataset.target)
print(grid)
# summarize the results of the grid search
print(grid.best_score_)
print(grid.best_estimator_.alpha)
```

For more information see the API for GridSearchCV and Exhaustive Grid Search section in the user guide.

Random Search Parameter Tuning

Random search is an approach to parameter tuning that will sample algorithm parameters from a random distribution (i.e. uniform) for a fixed number of iterations. A model is constructed and evaluated for each combination of parameters chosen.

The recipe below evaluates different alpha random values between 0 and 1 for the Ridge Regression algorithm on the standard diabetes dataset.

```
# Randomized Search for Algorithm Tuning
import numpy as np
from scipy.stats import uniform as sp_rand
from sklearn import datasets
from sklearn.linear_model import Ridge
from sklearn.model_selection import RandomizedSearchCV
# load the diabetes datasets
dataset = datasets.load_diabetes()
# prepare a uniform distribution to sample for the alpha parameter
param_grid = {'alpha': sp_rand()}
# create and fit a ridge regression model, testing random alpha values
model = Ridge()
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100)
rsearch.fit(dataset.data, dataset.target)
print(rsearch)
# summarize the results of the random parameter search
print(rsearch.best_score_)
print(rsearch.best_estimator_.alpha)
```

For more information see the API for RandomizedSearchCV and the the Randomized Parameter Optimization section in the user guide.

Summary

Algorithm parameter tuning is an important step for improving algorithm performance right before presenting results or preparing a system for production.