Power-Tuning

Recall the lab assignment where you used a set of nested for-loops to search through different combinations of your model's parameters with the hopes of finding the one that produced the highest accuracy score. SciKit-Learn offers you a systematic way of doing that automatically called `GridSearchCV`. When you were using the nested for-loop approach, we mentioned that was a very naive method. `GridSearchCV` takes care of your parameter tuning and also tacks on end-to-end cross validation. This results in more precisely tuned parameter than depending on simple model accuracy scores, and is why the algorithm is name Grid-Search-CV.

In its simplest form, `GridSearchCV` works by taking in an estimator, a grid of parameters you want optimized, and your cv split value. This is the example from SciKit-Learn's API page:

```
>>> from sklearn import svm, grid_search, datasets

>>> iris = datasets.load_iris()
>>> parameters = {'kernel':('linear', 'rbf'), 'C':[1, 5, 10]}
>>> model = svm.SVC()

>>> classifier = grid_search.GridSearchCV(model, parameters)
>>> classifier.fit(iris.data, iris.target)
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'kernel': ('linear', 'rbf'), 'C': [1, 5, 10]},
       pre_dispatch='2*n_jobs', refit=True, scoring=None, verbose=0)
```

In this example, `GridSearchCV` is being used to optimize a support vector classifier model. Since the exact parameters have been specified, GridSearchCV will build a table of every combination (but not permutation) of the available parameters and cross-validate each one separately:

CV Trials	**kernel = Linear**	**kernel = RBF**
C = 1.0	Linear, 1.0	RBF, 1.0
C = 5	Linear, 5	RBF, 5
C = 10	Linear, 10	RBF, 10

In addition to explicitly defining the parameters you want tested, you can also use randomized parameter optimization with SciKit-Learn's `RandomizedSearchCV` class. The semantics are a bit different here. First, instead of passing a list of grid objects (with GridSearchCV, you can actually perform multiple grid optimizations, consecutively), this time you pass in a your parameters as a single dictionary that holds either possible, discrete parameter values or distribution over them. SciPy's Statistics module have many such functions you can use to create continuous, discrete, and multivariate type distributions, such as `expon`, `gamma`, `uniform`, `randin` and many more:

```
>>> parameter_dist = {
  'C': scipy.stats.expon(scale=100),
  'kernel': ['linear'],
  'gamma': scipy.stats.expon(scale=.1),
}

>>> classifier = grid_search.RandomizedSearchCV(model, parameter_dist)
>>> classifier.fit(iris.data, iris.target)

RandomizedSearchCV(cv=None, error_score='raise',
          estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=
  decision_function_shape=None, degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
          fit_params={}, iid=True, n_iter=10, n_jobs=1,
          param_distributions={'kernel': ['linear'], 'C': <scipy.stats._
 'gamma': <scipy.stats._distn_infrastructure.rv_frozen object at 0x11034
          pre_dispatch='2*n_jobs', random_state=None, refit=True,
          scoring=None, verbose=0)
```

`RandomizedSearchCV` also takes in an optional `n_iter` parameter you can use to control the number of parameter settings that are sampled. Regardless of the cross validation search tool you end up using, after all of the methods exposed by the class are ran using the estimator that maximized the score of the out-of-bag data. So in the examples above, the `.fit()` method along with any subsequent methods, such as `.predict(), .score(), .transform(), .predict()` are all executed and return values as-if they were called on the best found estimator directly. SciKit-Learn has a very nice example that compares the execution times as well as scoring results of randomized search versus grid search, while trying to optimize various random forest parameters.