Hyperparameter tuning (HPT)

- Hyper-parameters are parameters that are not directly learnt within estimators.
- In `sklearn`, they are passed as arguments to the constructor of the estimator classes. e.g. regularization rate `alpha` for Lasso.

> It is possible and recommended to search the hyper-parameter space for the best cross validation score.

Any parameter provided when constructing an estimator may be optimized in this manner.

> Specifically, to find the names and current values for all parameters for a given estimator, use:
>
> ```
> estimator.get_param()
> ```

Note that it is common that a small subset of those parameters can have a large impact on the predictive or computation performance of the model while others can be left to their default values.

> It is recommended to read the docstring of the estimator class to get a finer understanding of their expected behavior, possibly by reading the enclosed reference to the literature.

Components of hyperparameter search

Hyper parameter search consists of

- an estimator (regressor or classifier);
- a parameter space;
- a method for searching or sampling candidates;
- a cross-validation scheme; and
- a score function.

HPT approaches

Two generic HPT approaches are:

- `GridSearchCV` exhaustively considers all parameter combinations for given values.
- `RandomizedSearchCV` can sample a given number of candidates from a parameter space with a specified distribution.

There are successive halving counterparts `HalvingGridSearchCV` and `HalvingRandomSearchCV`, that can be much faster at finding a good parameter combination.

A few models allow for specialized, efficient search strategy for HPT and we would use those alternatives whenever available.

Exhaustive grid search

The grid search provided by `GridSearchCV` exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter. For instance, the following param_grid:

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
```

specifies that two grids should be explored:

- One with a linear kernel and `C` values in [1, 10, 100, 1000], and
- The second one with an RBF kernel, and the cross-product of `C` values ranging in [1, 10, 100, 1000] and `gamma` values in [0.001, 0.0001].

The GridSearchCV instance implements the usual estimator API:

> It evaluates all possible combination of parameter values when “fitting” on a dataset and retains the best combination.

Randomized parameter optimization

- Implements a randomized search over parameters, where each setting is sampled from a distribution over possible parameter values.
- This has two main benefits over an exhaustive search:
    - A budget can be chosen independent of the number of parameters and possible values.
    - Adding parameters that do not influence the performance does not decrease efficiency.
- A dictionary is used to specify how parameters should be sampled.
- Additionally, a computation budget in terms of the number of sampled candidates or sampling iterations, is specified using the `n_iter` parameter.

For each parameter, either a distribution over possible values or a list of discrete choices (which will be sampled uniformly) can be specified:

```
{'C': scipy.stats.expon(scale=100), 'gamma': scipy.stats.expon(scale=.1),
  'kernel': ['rbf'], 'class_weight':['balanced', None]}
```

Tips for parameter search

- When evaluating the resulting model, it is important to do it on held-out samples that were not seen during the grid search process: it is recommended to split the data into a *development set* (to be fed to the `GridSearchCV` instance) and an *evaluation set* to compute performance metrics.

  > This can be done by using the `train_test_split` utility function.

- Specify an objective metric: single or multiple through `scoring` parameter.
- `GridSearchCV` and `RandomizedSearchCV` allow searching over parameters of composite or nested estimators such as `Pipeline`.
- [Parallelism] The parameter search tools evaluate each parameter combination on each data fold independently.

  > Computations can be run in parallel by using the keyword n_jobs=-1

- [Robustness for failure]
  - Some parameter settings may result in a failure to fit one or more folds of the data.
  - By default, this will cause the entire search to fail, even if some parameter settings could be fully evaluated.
  - Setting `error_score=0` (or `=np.NaN`) will make the procedure robust to such failure, issuing a warning and setting the score for that fold to `0` (or `NaN`), but completing the search.

Note to slidemaker: Include an example for Pipeline and gridsearch for polynomial regression.

Alternatives to brute force parameter search

Model-specific CV

- Some models can fit data for a range of values of some parameter almost as efficiently as fitting the estimator for a single value of the parameter.

- This feature can be leveraged to perform a more efficient cross-validation used for model selection of this parameter.

> The most common parameter amenable to this strategy is the parameter encoding the strength of the regularizer. Here we compute the regularization path of the estimator.

List of models for regression:

- `linear_model.LassoCV`
- `linear_model.LassoLarsCV`
- `linear_model.RidgeCV`
- `linear_model.ElasticNetCV`