Hyper-parameter search

In addition to the "learned" parameters (e.g., Θ) there are a number of parameters to the learning process itself. Among the ones we've seen so far

- Strength of regularization penalty
- Number of folds for cross validation
- Degree of the polynomial features

How do we choose values for these *hyper parameters* ?

One way is by searching through a space of possible values.

`sklearn` makes this easy via the `GridSearchCV` method, which we briefly describe.

`sklearn` GridSearchCV documentation (https://scikit-learn.org/stable/auto_examples/model_selection/plot_grid_search_digits.html)

Set the parameters by cross-validation tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4], 'C': [1, 10, 100, 1000]}, {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}] scores = ['precision', 'recall'] clf = GridSearchCV(SVC(), tuned_parameters, cv=5, scoring='%s_macro' % score)

`GridSearchCV` will create an object (which turns out to be an estimator).

In creating this object, we specify

- an instance e of an estimator
- a dictionary
    - whose keys are names of parameters to e
    - whose values are a list of possible values for the parameters

```
clf.fit(X_train, y_train)
```

Fitting the `GridSearchCV` estimator does the following

- it creates every possible combination of parameter values in the dictionary
- for each combination, it performs Cross Validated fitting of the estimator e

So it fits estimator e many times, one for each possible parameter combination across the multi-dimensional space of hyper-parameters.

Hence the name: Grid Search.

One can then have Grid Search report summaries as well as the best combination of parameters.

Randomized search in `sklearn`

RandomizedSearchCV documentation (https://scikit-learn.org/stable/modules/grid_search.html#randomized-parameter-optimization)

`GridSearchCV` searches the entire multi-dimensional space of hyper-parameters, which can be very large and therefore time consuming.

`sklearn` implements a randomized version of the exploration of the multi-dimensional space.

Instead of searching the grid exhaustively, it randomly samples combinations of hyper-parameters to try.

The user specifies how many samples are taken.

Alternative to Randomized Search

The assumption underlying Randomized Search is that all points in the multi-dimensional space of hyper-parameters are equally likely candidates for being the best.

Like any other optimization problem, experience tells us this is not likely to be the case:

- Some parameters have less impact on the Performance Metric than others
- The values for a particular hyper-parameter that lead to high performance tend to be clustered rather than evenly distributed

Is there a way to improve Randomized Search ?

One idea is to partition the space of possible values of a hyper-parameter unevenly.

One can then try an equal number of values of the hyper-parameter in each partition

- the increment between values of a large partition are larger than for a small partition

For example suppose the range for hyper-parameter p is $[0, 1000]$

- we create partitions using a logarithmic scale:
    - $[0, .001]$
    - $[.001, .01]$
    - $\vdots$
    - $[100, 1000]$

This would be consistent with our belief that the best values for p are extremely small.

So we explore small values in smaller increments than the values between 100 and 1000.

In [2]:
```
print("Done !")
```

```
Done !
```