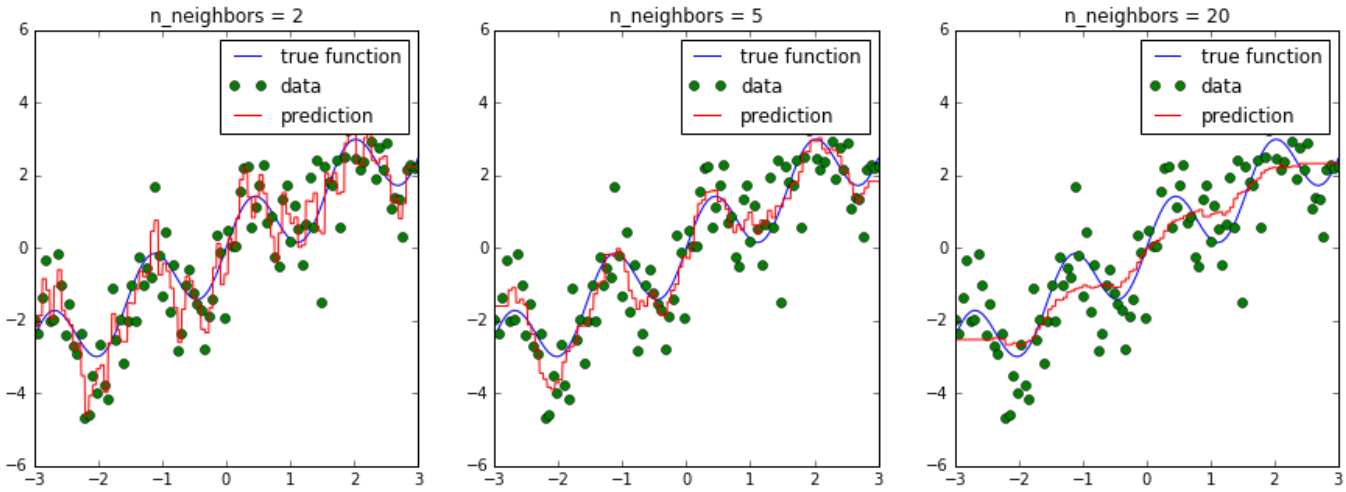
In [1]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Parameter selection, Validation, and Testing

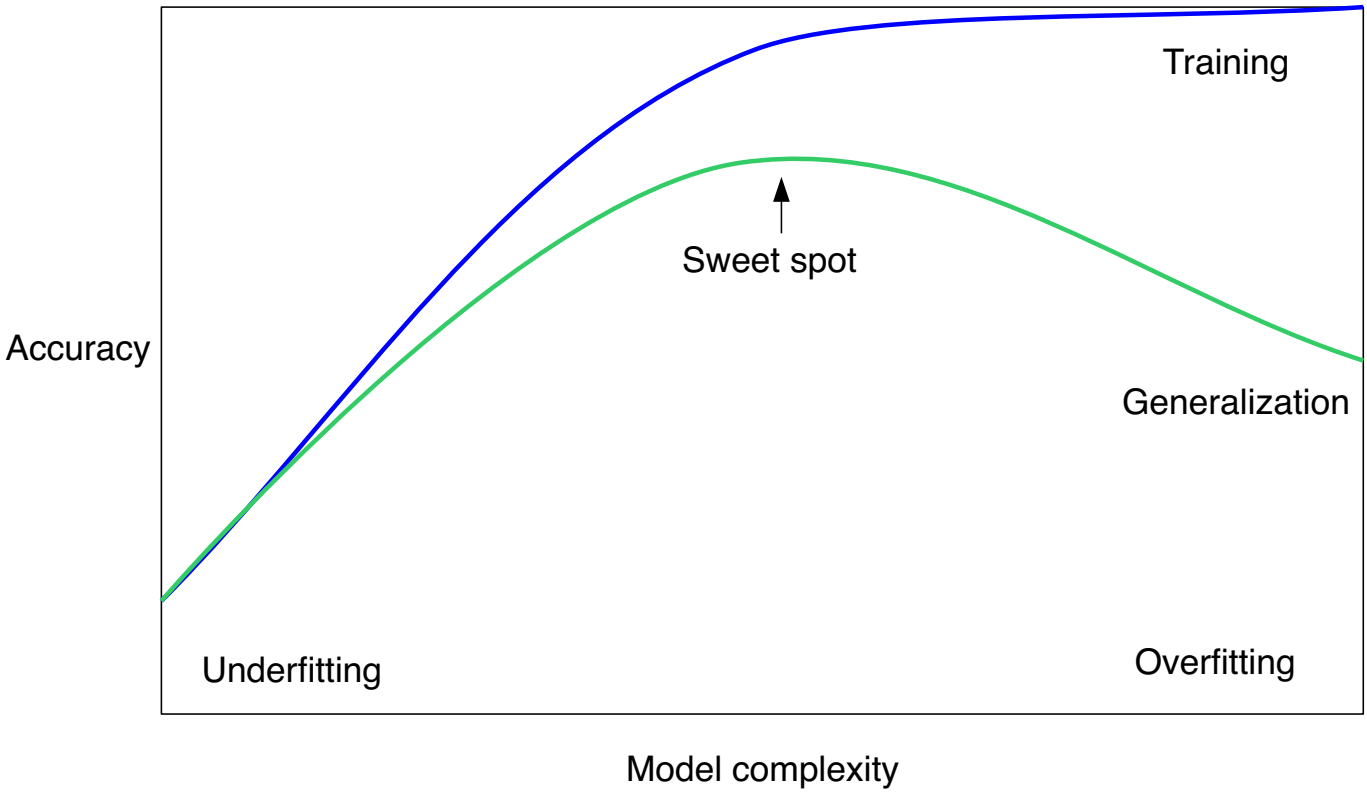
Most models have parameters that influence how complex a model they can learn. Remember using `KNeighborsRegressor`. If we change the number of neighbors we consider, we get a smoother and smoother prediction:



In the above figure, we see fits for three different values of `n_neighbors`. For `n_neighbors=2`, the data is overfit, the model is too flexible and can adjust too much to the noise in the training data. For `n_neighbors=20`, the model is not flexible enough, and can not model the variation in the data appropriately.

In the middle, for `n_neighbors = 5`, we have found a good mid-point. It fits the data fairly well, and does not suffer from the overfit or underfit problems seen in the figures on either side. What we would like is a way to quantitatively identify overfit and underfit, and optimize the hyperparameters (in this case, the polynomial degree d) in order to determine the best algorithm.

We trade off remembering too much about the particularities and noise of the training data vs. not modeling enough of the variability. This is a trade-off that needs to be made in basically every machine learning application and is a central concept, called bias-variance-tradeoff or "overfitting vs underfitting".



Hyperparameters, Over-fitting, and Under-fitting

Unfortunately, there is no general rule how to find the sweet spot, and so machine learning practitioners have to find the best trade-off of model-complexity and generalization by trying several hyperparameter settings. Hyperparameters are the internal knobs or tuning parameters of a machine learning algorithm (in contrast to model parameters that the algorithm learns from the training data -- for example, the weight coefficients of a linear regression model); the number of *k* in K-nearest neighbors is such a hyperparameter.

Most commonly this "hyperparameter tuning" is done using a brute force search, for example over multiple values of `n_neighbors`:

In [2]:

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.neighbors import KNeighborsRegressor
# generate toy dataset:
x = np.linspace(-3, 3, 100)
rng = np.random.RandomState(42)
y = np.sin(4 * x) + x + rng.normal(size=len(x))
X = x[:, np.newaxis]

cv = KFold(shuffle=True)

# for each parameter setting do cross-validation:
for n_neighbors in [1, 3, 5, 10, 20]:
    scores = cross_val_score(KNeighborsRegressor(n_neighbors=n_neighbors), X, y,
cv=cv)
    print("n_neighbors: %d, average score: %f" % (n_neighbors, np.mean(scores)))
```

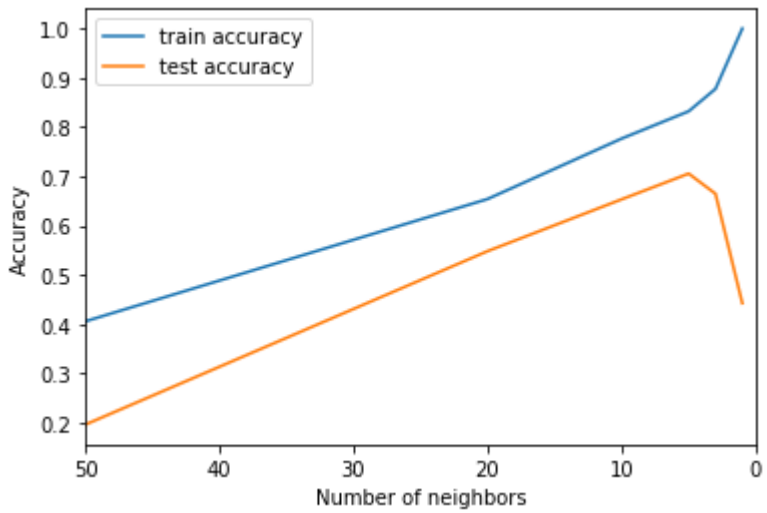
```
n_neighbors: 1, average score: 0.625118
n_neighbors: 3, average score: 0.723130
n_neighbors: 5, average score: 0.738529
n_neighbors: 10, average score: 0.744167
n_neighbors: 20, average score: 0.613944
```

```
/Users/mjburgess/anaconda3/lib/python3.7/site-packages/sklearn/model
_selection/_split.py:431: FutureWarning: The default value of n_spli
t will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(NSPLIT_WARNING, FutureWarning)
```

There is a function in scikit-learn, called `validation_plot` to reproduce the cartoon figure above. It plots one parameter, such as the number of neighbors, against training and validation error (using cross-validation):

In [3]:

```
from sklearn.model_selection import validation_curve
n_neighbors = [1, 3, 5, 10, 20, 50]
train_scores, test_scores = validation_curve(KNeighborsRegressor(), X, y, param_
name="n_neighbors",
                                             param_range=n_neighbors, cv=cv)
plt.plot(n_neighbors, train_scores.mean(axis=1), label="train accuracy")
plt.plot(n_neighbors, test_scores.mean(axis=1), label="test accuracy")
plt.ylabel('Accuracy')
plt.xlabel('Number of neighbors')
plt.xlim([50, 0])
plt.legend(loc="best");
```



Note that many neighbors mean a "smooth" or "simple" model, so the plot uses a reverted x axis.

If multiple parameters are important, like the parameters `C` and `gamma` in an `SVM` (more about that later), all possible combinations are tried:

In [4]:

```
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVR

# each parameter setting do cross-validation:
for C in [0.001, 0.01, 0.1, 1, 10]:
    for gamma in [0.001, 0.01, 0.1, 1]:
        scores = cross_val_score(SVR(C=C, gamma=gamma), X, y, cv=cv)
        print("C: %f, gamma: %f, average score: %f" % (C, gamma, np.mean(scores
)))
```

```
C: 0.001000, gamma: 0.001000, average score: -0.139488
C: 0.001000, gamma: 0.010000, average score: -0.003645
C: 0.001000, gamma: 0.100000, average score: -0.034461
C: 0.001000, gamma: 1.000000, average score: -0.085472
C: 0.010000, gamma: 0.001000, average score: -0.049619
C: 0.010000, gamma: 0.010000, average score: -0.029338
C: 0.010000, gamma: 0.100000, average score: 0.028655
C: 0.010000, gamma: 1.000000, average score: 0.062869
C: 0.100000, gamma: 0.001000, average score: -0.065948
C: 0.100000, gamma: 0.010000, average score: 0.180459
C: 0.100000, gamma: 0.100000, average score: 0.487274
C: 0.100000, gamma: 1.000000, average score: 0.463216
C: 1.000000, gamma: 0.001000, average score: 0.187084
C: 1.000000, gamma: 0.010000, average score: 0.575050
C: 1.000000, gamma: 0.100000, average score: 0.683474
C: 1.000000, gamma: 1.000000, average score: 0.712334
C: 10.000000, gamma: 0.001000, average score: 0.587896
C: 10.000000, gamma: 0.010000, average score: 0.607638
C: 10.000000, gamma: 0.100000, average score: 0.664830
C: 10.000000, gamma: 1.000000, average score: 0.746278
```

As this is such a very common pattern, there is a built-in class for this in scikit-learn, `GridSearchCV`. `GridSearchCV` takes a dictionary that describes the parameters that should be tried and a model to train.

The grid of parameters is defined as a dictionary, where the keys are the parameters and the values are the settings to be tested.

To inspect training score on the different folds, the parameter `return_train_score` is set to `True`.

In [5]:

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}

grid = GridSearchCV(SVR(), param_grid=param_grid, cv=cv, verbose=3, return_train
_score=True)
```

One of the great things about GridSearchCV is that it is a *meta-estimator*. It takes an estimator like SVR above, and creates a new estimator, that behaves exactly the same - in this case, like a regressor. So we can call `fit` on it, to train it:

In [6]:

```
grid.fit(X, y)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
[CV] C=0.001, gamma=0.001
.........................................
[CV]  C=0.001, gamma=0.001, score=(train=0.000, test=-0.043), total=
0.0s
[CV] C=0.001, gamma=0.001
.........................................
[CV]  C=0.001, gamma=0.001, score=(train=-0.001, test=-0.157), total
=   0.0s
[CV] C=0.001, gamma=0.001
.........................................
[CV]  C=0.001, gamma=0.001, score=(train=-0.001, test=-0.041), total
=   0.0s
[CV] C=0.001, gamma=0.01
.........................................
[CV]  C=0.001, gamma=0.01, score=(train=0.002, test=-0.040), total=
0.0s
[CV] C=0.001, gamma=0.01
.........................................
[CV]  C=0.001, gamma=0.01, score=(train=0.001, test=-0.154), total=
0.0s
[CV] C=0.001, gamma=0.01
.........................................
[CV]  C=0.001, gamma=0.01, score=(train=0.001, test=-0.039), total=
0.0s
[CV] C=0.001, gamma=0.1
.........................................
[CV]  C=0.001, gamma=0.1, score=(train=0.010, test=-0.029), total=
0.0s
[CV] C=0.001, gamma=0.1
.........................................
[CV]  C=0.001, gamma=0.1, score=(train=0.010, test=-0.143), total=
0.0s
[CV] C=0.001, gamma=0.1
.........................................
[CV]  C=0.001, gamma=0.1, score=(train=0.011, test=-0.027), total=
0.0s
[CV] C=0.001, gamma=1
.........................................
[CV]  C=0.001, gamma=1, score=(train=0.010, test=-0.032), total=
0.0s
[CV] C=0.001, gamma=1
.........................................
[CV]  C=0.001, gamma=1, score=(train=0.009, test=-0.146), total=
0.0s
[CV] C=0.001, gamma=1
.........................................
[CV]  C=0.001, gamma=1, score=(train=0.009, test=-0.029), total=
0.0s
[CV] C=0.01, gamma=0.001
.........................................
[CV]  C=0.01, gamma=0.001, score=(train=0.002, test=-0.040), total=
0.0s
[CV] C=0.01, gamma=0.001
.........................................
[CV]  C=0.01, gamma=0.001, score=(train=0.001, test=-0.154), total=
0.0s
[CV] C=0.01, gamma=0.001
.........................................
[CV]  C=0.01, gamma=0.001, score=(train=0.002, test=-0.039), total=
0.0s
```

```
[CV] C=0.01, gamma=0.01
.............................................
[CV]  C=0.01, gamma=0.01, score=(train=0.020, test=-0.015), total=
0.0s
[CV] C=0.01, gamma=0.01
.............................................
[CV]  C=0.01, gamma=0.01, score=(train=0.020, test=-0.127), total=
0.0s
[CV] C=0.01, gamma=0.01
.............................................
[CV]  C=0.01, gamma=0.01, score=(train=0.023, test=-0.014), total=
0.0s
[CV] C=0.01, gamma=0.1
.............................................
[CV]  C=0.01, gamma=0.1, score=(train=0.097, test=0.073), total=
0.0s
[CV] C=0.01, gamma=0.1
.............................................
[CV]  C=0.01, gamma=0.1, score=(train=0.101, test=-0.031), total=
0.0s
[CV] C=0.01, gamma=0.1
.............................................
[CV]  C=0.01, gamma=0.1, score=(train=0.107, test=0.076), total=
0.0s
[CV] C=0.01, gamma=1
.............................................
[CV] . C=0.01, gamma=1, score=(train=0.095, test=0.052), total=   0.
0s
[CV] C=0.01, gamma=1
.............................................
[CV]  C=0.01, gamma=1, score=(train=0.095, test=-0.061), total=   0.
0s
[CV] C=0.01, gamma=1
.............................................
[CV] . C=0.01, gamma=1, score=(train=0.095, test=0.064), total=   0.
0s
[CV] C=0.1, gamma=0.001
.............................................
[CV]  C=0.1, gamma=0.001, score=(train=0.021, test=-0.013), total=
0.0s
[CV] C=0.1, gamma=0.001
.............................................
[CV]  C=0.1, gamma=0.001, score=(train=0.021, test=-0.125), total=
0.0s
[CV] C=0.1, gamma=0.001
.............................................
[CV]  C=0.1, gamma=0.001, score=(train=0.026, test=-0.012), total=
0.0s
[CV] C=0.1, gamma=0.01
.............................................
[CV]  C=0.1, gamma=0.01, score=(train=0.172, test=0.167), total=
0.0s
[CV] C=0.1, gamma=0.01
.............................................
[CV]  C=0.1, gamma=0.01, score=(train=0.177, test=0.077), total=
0.0s
[CV] C=0.1, gamma=0.01
.............................................
[CV]  C=0.1, gamma=0.01, score=(train=0.206, test=0.156), total=
0.0s
[CV] C=0.1, gamma=0.1
```

```
.................................................
[CV]  C=0.1, gamma=0.1, score=(train=0.513, test=0.486), total=   0.
0s
[CV] C=0.1, gamma=0.1
.................................................
[CV]  C=0.1, gamma=0.1, score=(train=0.490, test=0.466), total=   0.
0s
[CV] C=0.1, gamma=0.1
.................................................
[CV]  C=0.1, gamma=0.1, score=(train=0.556, test=0.511), total=   0.
0s
[CV] C=0.1, gamma=1
.................................................
[CV] .. C=0.1, gamma=1, score=(train=0.499, test=0.428), total=   0.
0s
[CV] C=0.1, gamma=1
.................................................
[CV] .. C=0.1, gamma=1, score=(train=0.481, test=0.437), total=   0.
0s
[CV] C=0.1, gamma=1
.................................................
[CV] .. C=0.1, gamma=1, score=(train=0.569, test=0.475), total=   0.
0s
[CV] C=1, gamma=0.001
.................................................
[CV]  C=1, gamma=0.001, score=(train=0.183, test=0.179), total=   0.
0s
[CV] C=1, gamma=0.001
.................................................
[CV]  C=1, gamma=0.001, score=(train=0.189, test=0.090), total=   0.
0s
[CV] C=1, gamma=0.001
.................................................
[CV]  C=1, gamma=0.001, score=(train=0.223, test=0.169), total=   0.
0s
[CV] C=1, gamma=0.01
.................................................
[CV] . C=1, gamma=0.01, score=(train=0.569, test=0.575), total=   0.
0s
[CV] C=1, gamma=0.01
.................................................
[CV] . C=1, gamma=0.01, score=(train=0.578, test=0.593), total=   0.
0s
[CV] C=1, gamma=0.01
.................................................
[CV] . C=1, gamma=0.01, score=(train=0.645, test=0.541), total=   0.
0s
[CV] C=1, gamma=0.1
.................................................
[CV] .. C=1, gamma=0.1, score=(train=0.694, test=0.657), total=   0.
0s
[CV] C=1, gamma=0.1
.................................................
[CV] .. C=1, gamma=0.1, score=(train=0.643, test=0.696), total=   0.
0s
[CV] C=1, gamma=0.1
.................................................
[CV] .. C=1, gamma=0.1, score=(train=0.708, test=0.602), total=   0.
0s
[CV] C=1, gamma=1
.................................................
```

```
[CV] .... C=1, gamma=1, score=(train=0.760, test=0.701), total=   0.
0s
[CV] C=1, gamma=1
..................................................
[CV] .... C=1, gamma=1, score=(train=0.679, test=0.715), total=   0.
0s
[CV] C=1, gamma=1
..................................................
[CV] .... C=1, gamma=1, score=(train=0.785, test=0.604), total=   0.
0s
[CV] C=10, gamma=0.001
..............................................
[CV]  C=10, gamma=0.001, score=(train=0.566, test=0.578), total=
0.0s
[CV] C=10, gamma=0.001
..............................................
[CV]  C=10, gamma=0.001, score=(train=0.573, test=0.592), total=
0.0s
[CV] C=10, gamma=0.001
..............................................
[CV]  C=10, gamma=0.001, score=(train=0.645, test=0.527), total=
0.0s
[CV] C=10, gamma=0.01
..............................................
[CV]  C=10, gamma=0.01, score=(train=0.637, test=0.648), total=   0.
0s
[CV] C=10, gamma=0.01
..............................................
[CV]  C=10, gamma=0.01, score=(train=0.606, test=0.638), total=   0.
0s
[CV] C=10, gamma=0.01
..............................................
[CV]  C=10, gamma=0.01, score=(train=0.686, test=0.532), total=   0.
0s
[CV] C=10, gamma=0.1
..............................................
[CV] . C=10, gamma=0.1, score=(train=0.706, test=0.644), total=   0.
0s
[CV] C=10, gamma=0.1
..............................................
[CV] . C=10, gamma=0.1, score=(train=0.643, test=0.713), total=   0.
0s
[CV] C=10, gamma=0.1
..............................................
[CV] . C=10, gamma=0.1, score=(train=0.711, test=0.596), total=   0.
0s
[CV] C=10, gamma=1
..................................................
[CV] ... C=10, gamma=1, score=(train=0.838, test=0.747), total=   0.
0s
[CV] C=10, gamma=1
..................................................
[CV] ... C=10, gamma=1, score=(train=0.748, test=0.852), total=   0.
0s
[CV] C=10, gamma=1
..................................................
[CV] ... C=10, gamma=1, score=(train=0.844, test=0.643), total=   0.
0s
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaini
ng:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaini
ng:    0.0s
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed:    0.1s finishe
d
/Users/mjburgess/anaconda3/lib/python3.7/site-packages/sklearn/model
_selection/_search.py:813: DeprecationWarning: The default of the `i
id` parameter will change from True to False in version 0.22 and wil
l be removed in 0.24. This will change numeric results when test-set
sizes are unequal.
  DeprecationWarning)
```

Out[6]:

```
GridSearchCV(cv=KFold(n_splits=3, random_state=None, shuffle=True),
             error_score='raise-deprecating',
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=
3,
                           epsilon=0.1, gamma='auto_deprecated', ker
nel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='warn', n_jobs=None,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10],
                         'gamma': [0.001, 0.01, 0.1, 1]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=True,
             scoring=None, verbose=3)
```

What `fit` does is a bit more involved then what we did above. First, it runs the same loop with cross-validation, to find the best parameter combination. Once it has the best combination, it runs fit again on all data passed to fit (without cross-validation), to built a single new model using the best parameter setting.

Then, as with all models, we can use `predict` or `score`:

In [7]:

```
grid.predict(X)
```

Out[7]:

```
array([-1.79762875, -1.74054091, -1.71412904, -1.72272347, -1.768802
47,
       -1.8527208 , -1.97255382, -2.12407501, -2.30087676, -2.494634
29,
       -2.695503  , -2.89262935, -3.07474705, -3.23082299, -3.350713
14,
       -3.42578612, -3.44947391, -3.41771237, -3.32924127, -3.185742
05,
       -2.9918017 , -2.75470244, -2.48404785, -2.19124658, -1.888883
88,
       -1.59001819, -1.30744475, -1.05297034, -0.8367425 , -0.666673
33,
       -0.54799235, -0.4829551 , -0.4707249 , -0.50743515, -0.586428
52,
       -0.69865919, -0.83323456, -0.97806438, -1.12057877, -1.248472
61,
       -1.35043139, -1.41679516, -1.44012026, -1.41560488, -1.341352
5 ,
       -1.21845724, -1.05090633, -0.84530623, -0.61045003, -0.356753
98,
       -0.09559933,  0.16137852,  0.40300817,  0.61926205,  0.801855
31,
        0.94472644,  1.04437082,  1.10000798,  1.11357463,  1.089546
95,
        1.03460678,  0.95717608,  0.86685224,  0.7737823 ,  0.688017
51,
        0.61888941,  0.57444669,  0.56098656,  0.58270777,  0.641503
52,
        0.73690334,  0.86616306,  1.02449275,  1.20540425,  1.401152
66,
        1.60324152,  1.80295801,  1.99190412,  2.16249073,  2.308365
4 ,
        2.42474939,  2.50866621,  2.55905134,  2.57674055,  2.564341
91,
        2.52600389,  2.46709789,  2.39383845,  2.31286721,  2.230828
34,
        2.15396216,  2.0877418 ,  2.03657355,  2.00357681,  1.990453
42,
        1.99745004,  2.02341108,  2.06591373,  2.12147209,  2.185793
47])
```

You can inspect the best parameters found by `GridSearchCV` in the `best_params_` attribute, and the best score in the `best_score_` attribute:

In [8]:

```
print(grid.best_score_)
```

```
0.747200432009177
```

In [9]:

```
print(grid.best_params_)
```

```
{'C': 10, 'gamma': 1}
```

But you can investigate the performance and much more for each set of parameter values by accessing the `cv_results_` attributes. The `cv_results_` attribute is a dictionary where each key is a string and each value is array. It can therefore be used to make a pandas DataFrame.

In [10]:

```
type(grid.cv_results_)
```

Out[10]:

```
dict
```

In [11]:

```
print(grid.cv_results_.keys())
```

```
dict_keys(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_
score_time', 'param_C', 'param_gamma', 'params', 'split0_test_scor
e', 'split1_test_score', 'split2_test_score', 'mean_test_score', 'st
d_test_score', 'rank_test_score', 'split0_train_score', 'split1_trai
n_score', 'split2_train_score', 'mean_train_score', 'std_train_scor
e'])
```

In [12]:

```
import pandas as pd

cv_results = pd.DataFrame(grid.cv_results_)
cv_results.head()
```

Out[12]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	p
0	0.000664	0.000357	0.000410	0.000029	0.001	0.001	'ga
1	0.000372	0.000008	0.000385	0.000002	0.001	0.01	'ga
2	0.000348	0.000030	0.000385	0.000008	0.001	0.1	'ga
3	0.000388	0.000022	0.000387	0.000007	0.001	1	'ga
4	0.000370	0.000009	0.000384	0.000002	0.01	0.001	'ga

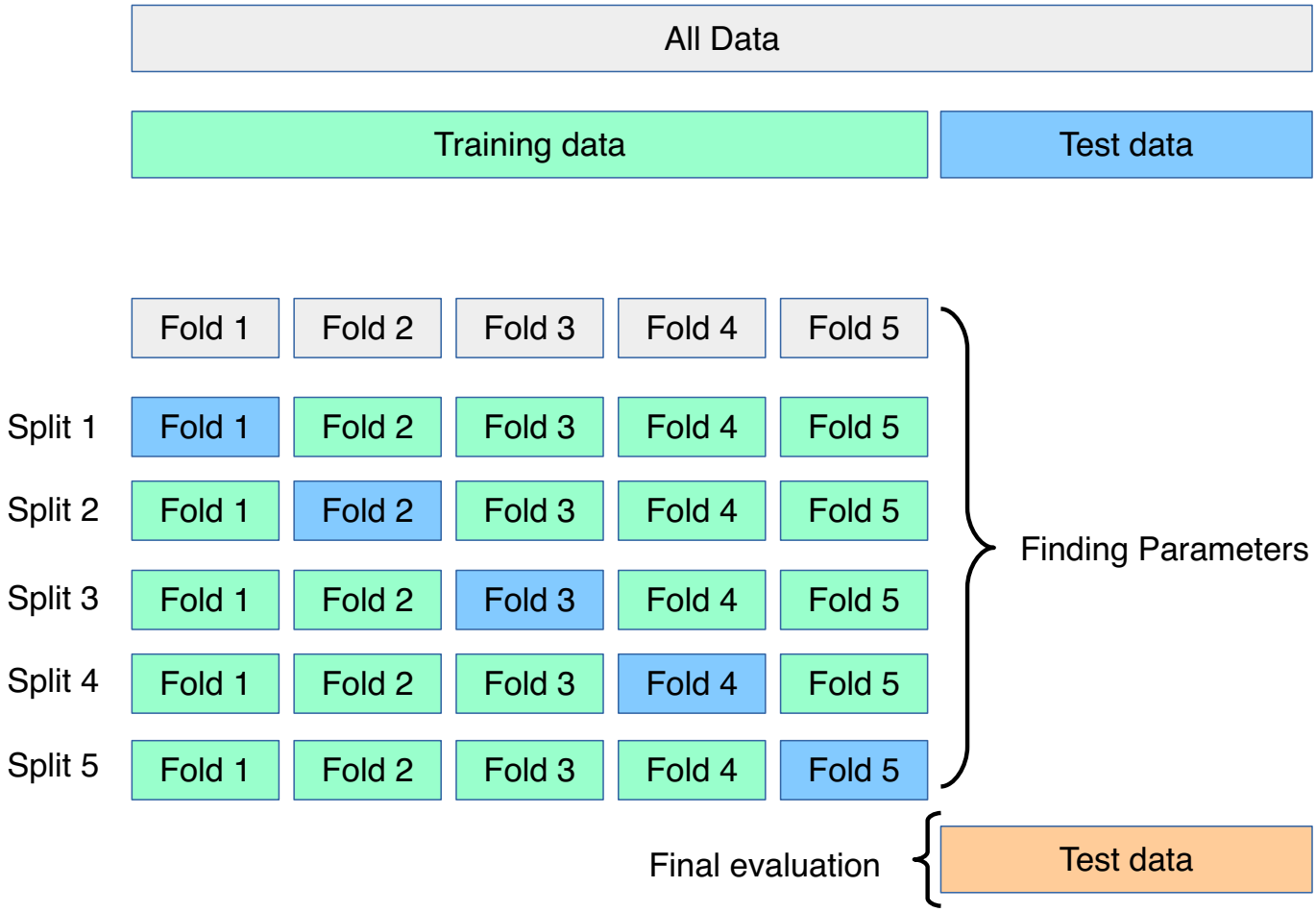
In [13]:

```
cv_results_tiny = cv_results[['param_C', 'param_gamma', 'mean_test_score']]
cv_results_tiny.sort_values(by='mean_test_score', ascending=False).head()
```

Out[13]:

	param_C	param_gamma	mean_test_score
19	10	1	0.747200
15	1	1	0.673437
14	1	0.1	0.651655
18	10	0.1	0.650655
17	10	0.01	0.606453

There is a problem with using this score for evaluation, however. You might be making what is called a multiple hypothesis testing error. If you try very many parameter settings, some of them will work better just by chance, and the score that you obtained might not reflect how your model would perform on new unseen data. Therefore, it is good to split off a separate test-set before performing grid-search. This pattern can be seen as a training-validation-test split, and is common in machine learning:



We can do this very easily by splitting of some test data using `train_test_split`, training `GridSearchCV` on the training set, and applying the `score` method to the test set:

In [14]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}
cv = KFold(n_splits=10, shuffle=True)

grid = GridSearchCV(SVR(), param_grid=param_grid, cv=cv)

grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

```
/Users/mjburgess/anaconda3/lib/python3.7/site-packages/sklearn/model
_selection/_search.py:813: DeprecationWarning: The default of the `i
id` parameter will change from True to False in version 0.22 and wil
l be removed in 0.24. This will change numeric results when test-set
sizes are unequal.
  DeprecationWarning)
```

Out[14]:

```
0.7262035177984737
```

We can also look at the parameters that were selected:

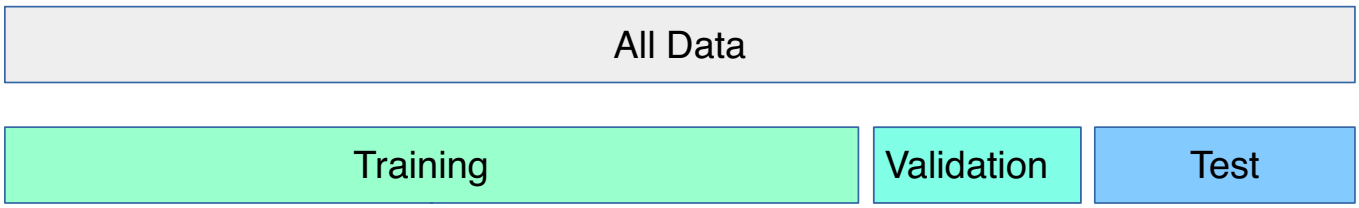
In [15]:

```
grid.best_params_
```

Out[15]:

```
{'C': 10, 'gamma': 1}
```

Some practitioners go for an easier scheme, splitting the data simply into three parts, training, validation and testing. This is a possible alternative if your training set is very large, or it is infeasible to train many models using cross-validation because training a model takes very long. You can do this with scikit-learn for example by splitting of a test-set and then applying GridSearchCV with ShuffleSplit cross-validation with a single iteration:



In [16]:

```
from sklearn.model_selection import train_test_split, ShuffleSplit

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma': [0.001, 0.01, 0.1, 1]}
single_split_cv = ShuffleSplit(n_splits=1)

grid = GridSearchCV(SVR(), param_grid=param_grid, cv=single_split_cv, verbose=3)

grid.fit(X_train, y_train)
grid.score(X_test, y_test)
```

```
Fitting 1 folds for each of 20 candidates, totalling 20 fits
[CV] C=0.001, gamma=0.001
.........................................
[CV] ............... C=0.001, gamma=0.001, score=-0.200, total=   0.
0s
[CV] C=0.001, gamma=0.01
.........................................
[CV] ............... C=0.001, gamma=0.01, score=-0.198, total=   0.
0s
[CV] C=0.001, gamma=0.1
.........................................
[CV] ................. C=0.001, gamma=0.1, score=-0.189, total=   0.
0s
[CV] C=0.001, gamma=1
.........................................
[CV] ................... C=0.001, gamma=1, score=-0.191, total=   0.
0s
[CV] C=0.01, gamma=0.001
.........................................
[CV] ............... C=0.01, gamma=0.001, score=-0.198, total=   0.
0s
[CV] C=0.01, gamma=0.01
.........................................
[CV] ................. C=0.01, gamma=0.01, score=-0.180, total=   0.
0s
[CV] C=0.01, gamma=0.1
.........................................
[CV] ................... C=0.01, gamma=0.1, score=-0.126, total=   0.
0s
[CV] C=0.01, gamma=1
.........................................
[CV] ..................... C=0.01, gamma=1, score=-0.120, total=   0.
0s
[CV] C=0.1, gamma=0.001
.........................................
[CV] ................. C=0.1, gamma=0.001, score=-0.179, total=   0.
0s
[CV] C=0.1, gamma=0.01
.........................................
[CV] ................... C=0.1, gamma=0.01, score=-0.030, total=   0.
0s
[CV] C=0.1, gamma=0.1
.........................................
[CV] ..................... C=0.1, gamma=0.1, score=0.421, total=   0.
0s
[CV] C=0.1, gamma=1
.........................................
[CV] ....................... C=0.1, gamma=1, score=0.370, total=   0.
0s
[CV] C=1, gamma=0.001
.........................................
[CV] ................... C=1, gamma=0.001, score=-0.027, total=   0.
0s
[CV] C=1, gamma=0.01
.........................................
[CV] ..................... C=1, gamma=0.01, score=0.540, total=   0.
0s
[CV] C=1, gamma=0.1
.........................................
[CV] ..................... C=1, gamma=0.1, score=0.647, total=   0.
0s
```

```
[CV] C=1, gamma=1
.................................................
[CV] ..................... C=1, gamma=1, score=0.656, total=   0.
0s
[CV] C=10, gamma=0.001
.................................................
[CV] ................... C=10, gamma=0.001, score=0.528, total=   0.
0s
[CV] C=10, gamma=0.01
.................................................
[CV] ................... C=10, gamma=0.01, score=0.592, total=   0.
0s
[CV] C=10, gamma=0.1
.................................................
[CV] ................... C=10, gamma=0.1, score=0.670, total=   0.
0s
[CV] C=10, gamma=1
.................................................
[CV] ..................... C=10, gamma=1, score=0.734, total=   0.
0s

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurr
ent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaini
ng:    0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaini
ng:    0.0s
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    0.0s finishe
d
```

Out[16]:

```
0.7262035177984737
```

This is much faster, but might result in worse hyperparameters and therefore worse results.

In [17]:

```
clf = GridSearchCV(SVR(), param_grid=param_grid)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
/Users/mjburgess/anaconda3/lib/python3.7/site-packages/sklearn/model
_selection/_split.py:1978: FutureWarning: The default value of cv wi
ll change from 3 to 5 in version 0.22. Specify it explicitly to sile
nce this warning.
  warnings.warn(CV_WARNING, FutureWarning)
```

Out[17]:

```
0.7262035177984737
```

EXERCISE:

- Apply grid-search to find the best setting for the number of neighbors in ``KNeighborsClassifier``, and apply it to the digits dataset.

In [18]:

```
# %load solutions/14_grid_search.py
```