08_hyperparameter-optimization

June 23, 2022

CPSC 330
Applied Machine Learning

1 Lecture 8: Hyperparameter Optimization and Optimization Bias

UBC 2022 Summer

Instructor: Mehrdad Oveisi

1.1 Imports

[1]:
```
import os
import sys

sys.path.append("code/.")

import IPython
import ipywidgets as widgets
import matplotlib.pyplot as plt
import mglearn
import numpy as np
import pandas as pd
from IPython.display import HTML, display
from ipywidgets import interact, interactive
from plotting_functions import *
from sklearn.dummy import DummyClassifier
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.impute import SimpleImputer
from sklearn.model_selection import cross_val_score, cross_validate,␣
 ↪train_test_split
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from utils import *

%matplotlib inline
pd.set_option("display.max_colwidth", 200)
```

1.1.1 Learning outcomes

From this lecture, you will be able to

- explain the need for hyperparameter optimization

- carry out hyperparameter optimization using `sklearn`'s `GridSearchCV` and `RandomizedSearchCV`
- explain different hyperparameters of `GridSearchCV`
- explain the importance of selecting a good range for the values.
- explain optimization bias
- identify and reason when to trust and not trust reported accuracies

1.2 Hyperparameter optimization motivation

1.2.1 Motivation

- Remember that the **fundamental goal** of supervised machine learning is to **generalize** beyond what we see in the training examples.
- We have been using data splitting and cross-validation to provide a framework to **approximate generalization error**.

- With this framework, we can improve the model's generalization performance by ***tuning model hyperparameters*** using cross-validation on the training set.

1.2.2 Hyperparameters: the problem

- In order to improve the generalization performance, finding the best values for the important hyperparameters of a model is necessary for almost all models and datasets.
- Picking **good hyperparameters** is important because if we don't do it, we might end up with an **underfit** or **overfit** model.

1.2.3 Some ways to pick hyperparameters:

- **Manual** or expert knowledge or heuristics based optimization
- Data-driven or **automated** optimization

Manual hyperparameter optimization

- Advantage: we may have some intuition about what might work.
  - E.g. if I'm massively overfitting, try decreasing `max_depth` or `C`.
- Disadvantages
  - it takes a lot of work
  - not reproducible

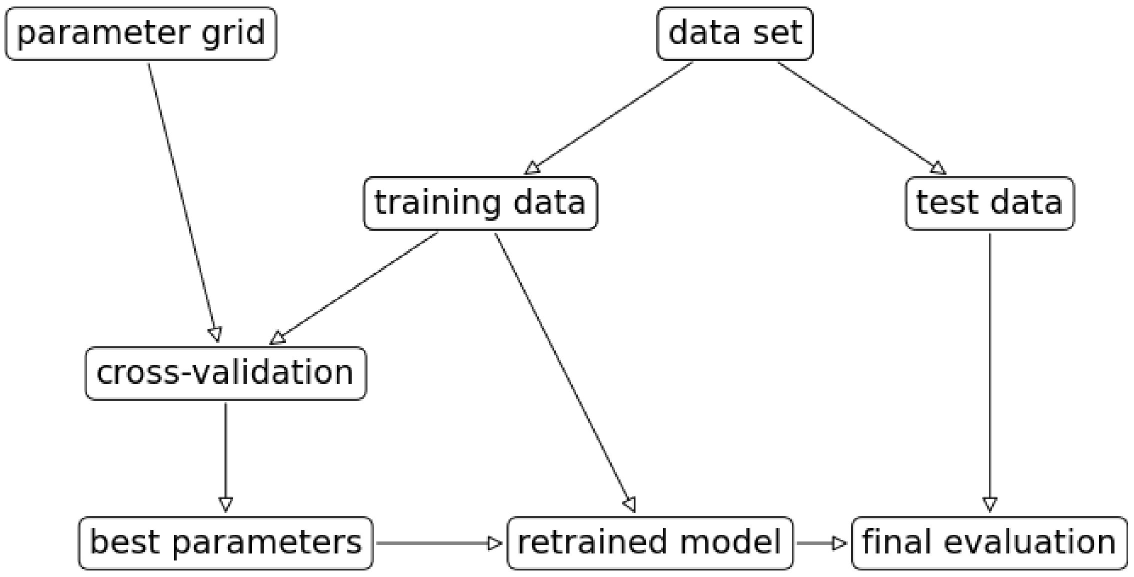
– in very complicated cases, our intuition might be worse than a data-driven approach

1.2.4 Automated hyperparameter optimization

- Formulate the hyperparamter optimization as a **one big search problem**.
- Often we have many hyperparameters of different types: Categorical, integer, and continuous.
- Often, the search space is quite big and systematic search for optimal values is infeasible.

In homework assignments, we have been carrying out hyperparameter search by exhaustively trying different possible combinations of the hyperparameters of interest.

[2]:
```
mglearn.plots.plot_grid_search_overview()
```



Let's look at an example of tuning `max_depth` of the `DecisionTreeClassifier` on the Spotify dataset.

[3]:
```
spotify_df = pd.read_csv("data/spotify.csv", index_col=0)
X_spotify = spotify_df.drop(columns=["target", "song_title", "artist"])
y_spotify = spotify_df["target"]
X_spotify.head()
```

[3]:
```
   acousticness  danceability  duration_ms  energy  instrumentalness  key  \
0        0.0102         0.833       204600   0.434          0.021900    2
1        0.1990         0.743       326933   0.359          0.006110    1
2        0.0344         0.838       185707   0.412          0.000234    2
3        0.6040         0.494       199413   0.338          0.510000    5
```

```
4        0.1800         0.678       392893   0.561          0.512000    5

   liveness  loudness  mode  speechiness    tempo  time_signature  valence
0    0.1650    -8.795     1       0.4310  150.062             4.0    0.286
1    0.1370   -10.401     1       0.0794  160.083             4.0    0.588
2    0.1590    -7.148     1       0.2890   75.044             4.0    0.173
3    0.0922   -15.236     1       0.0261   86.468             4.0    0.230
4    0.4390   -11.648     0       0.0694  174.004             4.0    0.904
```

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_spotify, y_spotify, test_size=0.2, random_state=123
)
```

[5]:
```
best_score = 0

param_grid = {"max_depth": np.arange(1, 20, 2)}

results_dict = {"max_depth": [], "mean_cv_score": []}

for depth in param_grid["max_depth"]:
    dt = DecisionTreeClassifier(max_depth=depth)
    scores = cross_val_score(dt, X_train, y_train)  # perform cross-validation
    mean_score = np.mean(scores)  # compute mean cross-validation accuracy
    if mean_score > best_score:  # if we got a better score, store the score␣
 ↪and parameters
        best_score = mean_score
        best_params = {"max_depth": depth}
    results_dict["max_depth"].append(depth)
    results_dict["mean_cv_score"].append(mean_score)
```

[6]:
```
best_params
```

[6]:
```
{'max_depth': 5}
```

[7]:
```
best_score
```

[7]:
```
0.71792204295906
```

Let's try SVM RBF and tuning `C` and `gamma` on the same dataset.

[8]:
```
pipe_svm = make_pipeline(StandardScaler(), SVC())  # We need scaling for SVM RBF
pipe_svm.fit(X_train, y_train)
```

[8]:
```
Pipeline(steps=[('standardscaler', StandardScaler()), ('svc', SVC())])
```

Let's try cross-validation with **default hyperparameters** of SVC.

[9]:
```
scores = cross_validate(pipe_svm, X_train, y_train, return_train_score=True)
pd.DataFrame(scores).mean().rename('mean').to_frame().T
```

```
[9]:       fit_time  score_time  test_score  train_score
     mean  0.199797     0.06765    0.738998     0.814011
```

Now let's try **exhaustive hyperparameter search** using for loops.

This is what we have been doing for this:

```
for gamma in [0.01, 1, 10, 100]: # for some values of gamma
    for C in [0.01, 1, 10, 100]: # for some values of C
        for fold in folds:
            fit within training portion using the given combination (gamma, C)
            score on validation portion
        compute average score

pick hyperparameter values (gamma, C), yielding best average score
```

```
[68]: param_grid = {
          "C": [0.001, 0.01, 0.1, 1, 10, 100],
          "gamma": [0.001, 0.01, 0.1, 1, 10, 100],
      }

      results_dict = {"C": [], "gamma": [], "mean_cv_score": []}

      for gamma in param_grid["gamma"]:
          for C in param_grid["C"]:  # for each combination of parameters, train an␣
       ↪SVC
              pipe_svm = make_pipeline(StandardScaler(), SVC(gamma=gamma, C=C))
              scores = cross_val_score(pipe_svm, X_train, y_train)  # perform␣
       ↪cross-validation
              mean_score = np.mean(scores)  # compute mean cross-validation accuracy
              # we can find the best_score here, or later using results_dict
              results_dict["C"].append(C)
              results_dict["gamma"].append(gamma)
              results_dict["mean_cv_score"].append(mean_score)
```

```
[11]: results_df = pd.DataFrame(results_dict)
      results_df.sort_values(by="mean_cv_score", ascending=False).head()
```

```
[11]:         C  gamma  mean_cv_score
      15    1.0   0.10       0.743961
      11  100.0   0.01       0.732792
      16   10.0   0.10       0.729091
      10   10.0   0.01       0.720391
      17  100.0   0.10       0.711715
```

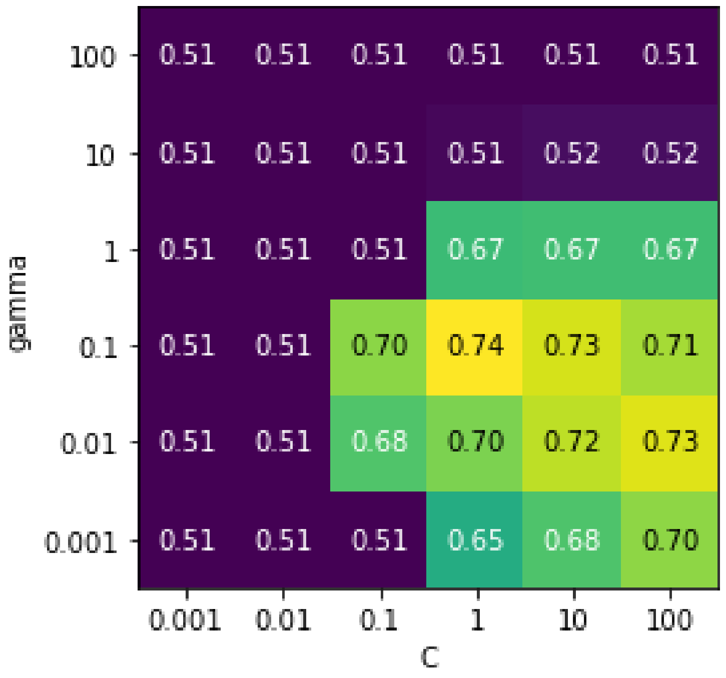
```
[69]: best_score_index = results_df['mean_cv_score'].argmax()
      best_score = results_df.loc[best_score_index, 'mean_cv_score']
      best_score
```

```
[69]: 0.7439609253312309
```

```
[13]: best_parameters = results_df.loc[best_score_index, ['C', 'gamma']].to_dict()
      best_parameters
```

```
[13]: {'C': 1.0, 'gamma': 0.1}
```

```
[14]: scores = np.array(results_df.mean_cv_score).reshape(6, 6)

      mglearn.tools.heatmap(
          scores,
          xlabel="C",
          xticklabels=param_grid["C"],
          ylabel="gamma",
          yticklabels=param_grid["gamma"],
          cmap="viridis",
      );
      # plot the mean cross-validation scores
```



- Each point in the heat map corresponds to one run of cross-validation, with a particular setting
- Colour encodes cross-validation accuracy
  - Lighter colour means high accuracy
  - Darker colour means low accuracy
- SVC is quite sensitive to hyperparameter settings.

- Adjusting hyperparameters can change the accuracy from 0.51 to 0.74!

[15]:
```
print("Grid size:", np.prod([len(v) for v in param_grid.values()]))
param_grid
```

```
Grid size: 36
```

[15]:
```
{'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100]}
```

- We have 6 possible values for `C` and 6 possible values for `gamma`.
- In 5-fold cross-validation, for each combination of parameter values, five accuracies are computed.
- So to evaluate the accuracy of the SVM using 6 values of `C` and 6 values of `gamma` using five-fold cross-validation, we need to train 6 * 6 * 5 = 180 models!

Python Side Note

Let's use `best_parameters` as an example dictionary

[16]:
```
best_parameters
```

[16]:
```
{'C': 1.0, 'gamma': 0.1}
```

Example 1: unpacking a dictionary

[17]:
```
{'A': 'abc', **best_parameters, 'D': 'def'}
```

[17]:
```
{'A': 'abc', 'C': 1.0, 'gamma': 0.1, 'D': 'def'}
```

Example 2: unpacking a dictionary used as function arguments

[18]:
```
def test1(C, gamma):
    print(f"C is {C} and gamma is {gamma}")

test1(**best_parameters)
```

```
C is 1.0 and gamma is 0.1
```

Example 3: unpacking using a single asterisk can be used for lists and tuples

[19]:
```
[*best_parameters]
```

[19]:
```
['C', 'gamma']
```

[20]:
```
(*best_parameters,)  # note use of comma ',' here to enforce () be interpreted␣
↪as a tuple
```

[20]:
```
('C', 'gamma')
```

[21]:
```
(*best_parameters, 'Something else', 5)  # last comma is not required anymore␣
↪(but still optional)
```

[21]:
```
('C', 'gamma', 'Something else', 5)
```

This single and double asterisks operators can be used in function definitions too

[22]:
```
def test2(*args, **kwargs):
    print("Positional arguments:", args, "which also can be unpacked:", *args)
    print("Keyword arguments:", kwargs)  # but cannot unpack here as **kwargs;␣
 ↪why? Give it a try!

test2(11, 123, a_kw_param=456, **best_parameters, another_kw_param=789)
```

```
Positional arguments: (11, 123) which also can be unpacked: 11 123
Keyword arguments: {'a_kw_param': 456, 'C': 1.0, 'gamma': 0.1,
'another_kw_param': 789}
```

Python Side Note End

Once we have optimized hyperparameters, we retrain a model on the full training set with these optimized hyperparameters.

[23]:
```
best_parameters
```

[23]:
```
{'C': 1.0, 'gamma': 0.1}
```

[24]:
```
# Retrain a model with optimized hyperparameters on the combined training and␣
 ↪validation set
pipe_svm = make_pipeline(StandardScaler(), SVC(**best_parameters))
pipe_svm.fit(X_train, y_train)
```

[24]:
```
Pipeline(steps=[('standardscaler', StandardScaler()), ('svc', SVC(gamma=0.1))])
```

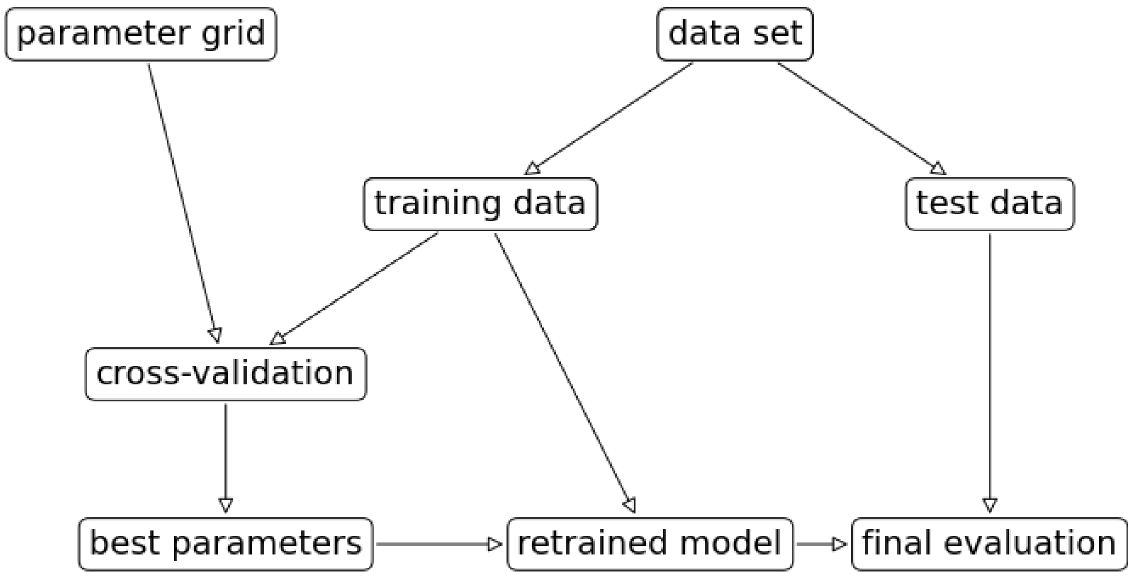
And finally evaluate the performance of this model on the test set.

[25]:
```
pipe_svm.score(X_test, y_test)  # Final evaluation on the test data
```

[25]:
```
0.7376237623762376
```

This process is so common that there are some **standard methods** in `scikit-learn` where we can carry out all of this in a more compact way.

[26]:
```
mglearn.plots.plot_grid_search_overview()
```



In this lecture we are going to talk about two such most commonly used **automated optimizations methods** from `scikit-learn`.

- Exhaustive grid search: `sklearn.model_selection.GridSearchCV`
- Randomized search: `sklearn.model_selection.RandomizedSearchCV`

The `CV` stands for cross-validation; these methods have built-in cross-validation.

1.3 Exhaustive grid search: `sklearn.model_selection.GridSearchCV`

[27]:
```
from sklearn import set_config

set_config(display="diagram")
```

- For `GridSearchCV` we need
  - an instantiated model or a pipeline
  - a parameter grid: A user specifies a set of values for each hyperparameter.
  - other optional arguments

The method considers product of the sets and then evaluates each combination one by one.

[28]:
```
from sklearn.model_selection import GridSearchCV

pipe_svm = make_pipeline(StandardScaler(), SVC())

param_grid = {
    "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
    "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
}

grid_search = GridSearchCV(
    pipe_svm, param_grid, cv=5, n_jobs=-1, return_train_score=True
)
```

1.3.1 `n_jobs=-1`

- Note the `n_jobs=-1` above.
- Hyperparameter optimization can be done *in parallel* for each of the configurations.
- This is very useful when scaling up to large numbers of machines in the cloud.

1.3.2 The `__` syntax

- Above: we have a nesting of transformers.
- We can access the parameters of the "inner" objects by using ___ to go "deeper":
- `svc__gamma`: the `gamma` of the `svc` of the pipeline
- `svc__C`: the `C` of the `svc` of the pipeline

The `GridSearchCV` object above behaves like a classifier. We can call `fit`, `predict` or `score` on it.

[29]:
```
grid_search.fit(X_train, y_train) # all the work is done here
grid_search
```

[29]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('standardscaler', StandardScaler()),
                                       ('svc', SVC())]),
             n_jobs=-1,
             param_grid={'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100],
                         'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100]},
             return_train_score=True)
```

Fitting the `GridSearchCV` object - Searches for the best hyperparameter values - You can access the best score and the best hyperparameters using `best_score_` and `best_params_` attributes, respectively.

[30]:
```
grid_search.best_score_
```

[30]:
```
0.7439609253312309
```

[31]:
```
grid_search.best_params_
```

[31]:
```
{'svc__C': 1.0, 'svc__gamma': 0.1}
```

- It is often helpful to visualize results of all cross-validation experiments.
- You can access this information using `cv_results_` attribute of a fitted `GridSearchCV` object.

```
[32]: results = pd.DataFrame(grid_search.cv_results_)
      results.T
```

```
[32]:                                                          0   \
      mean_fit_time                                     0.319606
      std_fit_time                                      0.023713
      mean_score_time                                   0.095841
      std_score_time                                    0.013952
      param_svc__C                                         0.001
      param_svc__gamma                                     0.001
      params                {'svc__C': 0.001, 'svc__gamma': 0.001}
      split0_test_score                                  0.50774
      split1_test_score                                  0.50774
      split2_test_score                                  0.50774
      split3_test_score                                 0.506211
      split4_test_score                                 0.509317
      mean_test_score                                    0.50775
      std_test_score                                    0.000982
      rank_test_score                                         21
      split0_train_score                                0.507752
      split1_train_score                                0.507752
      split2_train_score                                0.507752
      split3_train_score                                0.508133
      split4_train_score                                0.507359
      mean_train_score                                   0.50775
      std_train_score                                   0.000245

                                                              1   \
      mean_fit_time                                    0.321064
      std_fit_time                                     0.033357
      mean_score_time                                  0.106591
      std_score_time                                   0.016529
      param_svc__C                                        0.001
      param_svc__gamma                                     0.01
      params                {'svc__C': 0.001, 'svc__gamma': 0.01}
      split0_test_score                                 0.50774
      split1_test_score                                 0.50774
      split2_test_score                                 0.50774
      split3_test_score                                0.506211
      split4_test_score                                0.509317
      mean_test_score                                   0.50775
      std_test_score                                   0.000982
      rank_test_score                                        21
      split0_train_score                               0.507752
      split1_train_score                               0.507752
      split2_train_score                               0.507752
      split3_train_score                               0.508133
```

```
split4_train_score                                    0.507359
mean_train_score                                       0.50775
std_train_score                                       0.000245

                                                             2  \
mean_fit_time                                       0.284587
std_fit_time                                        0.036894
mean_score_time                                     0.101623
std_score_time                                      0.003895
param_svc__C                                           0.001
param_svc__gamma                                         0.1
params                {'svc__C': 0.001, 'svc__gamma': 0.1}
split0_test_score                                    0.50774
split1_test_score                                    0.50774
split2_test_score                                    0.50774
split3_test_score                                   0.506211
split4_test_score                                   0.509317
mean_test_score                                      0.50775
std_test_score                                      0.000982
rank_test_score                                           21
split0_train_score                                  0.507752
split1_train_score                                  0.507752
split2_train_score                                  0.507752
split3_train_score                                  0.508133
split4_train_score                                  0.507359
mean_train_score                                     0.50775
std_train_score                                     0.000245

                                                             3  \
mean_fit_time                                       0.236678
std_fit_time                                        0.036106
mean_score_time                                     0.085878
std_score_time                                      0.010228
param_svc__C                                           0.001
param_svc__gamma                                         1.0
params                {'svc__C': 0.001, 'svc__gamma': 1.0}
split0_test_score                                    0.50774
split1_test_score                                    0.50774
split2_test_score                                    0.50774
split3_test_score                                   0.506211
split4_test_score                                   0.509317
mean_test_score                                      0.50775
std_test_score                                      0.000982
rank_test_score                                           21
split0_train_score                                  0.507752
split1_train_score                                  0.507752
split2_train_score                                  0.507752
```

```
split3_train_score                                 0.508133
split4_train_score                                 0.507359
mean_train_score                                    0.50775
std_train_score                                    0.000245

                                                        4   \
mean_fit_time                                    0.298828
std_fit_time                                     0.026777
mean_score_time                                  0.106827
std_score_time                                   0.015524
param_svc__C                                        0.001
param_svc__gamma                                       10
params                {'svc__C': 0.001, 'svc__gamma': 10}
split0_test_score                                 0.50774
split1_test_score                                 0.50774
split2_test_score                                 0.50774
split3_test_score                                0.506211
split4_test_score                                0.509317
mean_test_score                                   0.50775
std_test_score                                   0.000982
rank_test_score                                        21
split0_train_score                               0.507752
split1_train_score                               0.507752
split2_train_score                               0.507752
split3_train_score                               0.508133
split4_train_score                               0.507359
mean_train_score                                  0.50775
std_train_score                                  0.000245

                                                         5   \
mean_fit_time                                     0.276166
std_fit_time                                      0.029502
mean_score_time                                   0.096781
std_score_time                                    0.013255
param_svc__C                                         0.001
param_svc__gamma                                       100
params                {'svc__C': 0.001, 'svc__gamma': 100}
split0_test_score                                  0.50774
split1_test_score                                  0.50774
split2_test_score                                  0.50774
split3_test_score                                 0.506211
split4_test_score                                 0.509317
mean_test_score                                    0.50775
std_test_score                                    0.000982
rank_test_score                                         21
split0_train_score                                0.507752
split1_train_score                                0.507752
```

```
split2_train_score                                   0.507752
split3_train_score                                   0.508133
split4_train_score                                   0.507359
mean_train_score                                      0.50775
std_train_score                                      0.000245

                                                           6   \
mean_fit_time                                         0.264388
std_fit_time                                          0.038898
mean_score_time                                       0.100339
std_score_time                                        0.022026
param_svc__C                                              0.01
param_svc__gamma                                         0.001
params                {'svc__C': 0.01, 'svc__gamma': 0.001}
split0_test_score                                      0.50774
split1_test_score                                      0.50774
split2_test_score                                      0.50774
split3_test_score                                     0.506211
split4_test_score                                     0.509317
mean_test_score                                        0.50775
std_test_score                                        0.000982
rank_test_score                                             21
split0_train_score                                    0.507752
split1_train_score                                    0.507752
split2_train_score                                    0.507752
split3_train_score                                    0.508133
split4_train_score                                    0.507359
mean_train_score                                       0.50775
std_train_score                                       0.000245

                                                          7   \
mean_fit_time                                        0.230133
std_fit_time                                         0.022421
mean_score_time                                      0.082995
std_score_time                                       0.009681
param_svc__C                                             0.01
param_svc__gamma                                         0.01
params                {'svc__C': 0.01, 'svc__gamma': 0.01}
split0_test_score                                     0.50774
split1_test_score                                     0.50774
split2_test_score                                     0.50774
split3_test_score                                    0.506211
split4_test_score                                    0.509317
mean_test_score                                       0.50775
std_test_score                                       0.000982
rank_test_score                                            21
split0_train_score                                   0.507752
```

```
split1_train_score                                       0.507752
split2_train_score                                       0.507752
split3_train_score                                       0.508133
split4_train_score                                       0.507359
mean_train_score                                          0.50775
std_train_score                                          0.000245

                                                              8   \
mean_fit_time                                          0.256758
std_fit_time                                           0.029509
mean_score_time                                        0.100254
std_score_time                                         0.016962
param_svc__C                                               0.01
param_svc__gamma                                            0.1
params              {'svc__C': 0.01, 'svc__gamma': 0.1}
split0_test_score                                       0.50774
split1_test_score                                       0.50774
split2_test_score                                       0.50774
split3_test_score                                      0.506211
split4_test_score                                      0.509317
mean_test_score                                         0.50775
std_test_score                                         0.000982
rank_test_score                                              21
split0_train_score                                     0.507752
split1_train_score                                     0.507752
split2_train_score                                     0.507752
split3_train_score                                     0.508133
split4_train_score                                     0.507359
mean_train_score                                        0.50775
std_train_score                                        0.000245

                                                              9  …  \
mean_fit_time                                          0.251678  …
std_fit_time                                           0.044128  …
mean_score_time                                         0.08771  …
std_score_time                                         0.009657  …
param_svc__C                                               0.01  …
param_svc__gamma                                            1.0  …
params              {'svc__C': 0.01, 'svc__gamma': 1.0}  …
split0_test_score                                       0.50774  …
split1_test_score                                       0.50774  …
split2_test_score                                       0.50774  …
split3_test_score                                      0.506211  …
split4_test_score                                      0.509317  …
mean_test_score                                         0.50775  …
std_test_score                                         0.000982  …
rank_test_score                                              21  …
```

```
split0_train_score                                  0.507752  …
split1_train_score                                  0.507752  …
split2_train_score                                  0.507752  …
split3_train_score                                  0.508133  …
split4_train_score                                  0.507359  …
mean_train_score                                     0.50775  …
std_train_score                                     0.000245  …

                                                    26  \
mean_fit_time                                 0.304863
std_fit_time                                  0.086008
mean_score_time                               0.077875
std_score_time                                 0.02544
param_svc__C                                        10
param_svc__gamma                                   0.1
params              {'svc__C': 10, 'svc__gamma': 0.1}
split0_test_score                             0.702786
split1_test_score                             0.767802
split2_test_score                             0.693498
split3_test_score                             0.736025
split4_test_score                             0.745342
mean_test_score                               0.729091
std_test_score                                0.027457
rank_test_score                                      3
split0_train_score                            0.923256
split1_train_score                             0.91938
split2_train_score                            0.925581
split3_train_score                            0.923315
split4_train_score                            0.915569
mean_train_score                               0.92142
std_train_score                                0.00354

                                                    27  \
mean_fit_time                                 0.383871
std_fit_time                                  0.143387
mean_score_time                               0.124079
std_score_time                                0.039681
param_svc__C                                        10
param_svc__gamma                                   1.0
params              {'svc__C': 10, 'svc__gamma': 1.0}
split0_test_score                             0.671827
split1_test_score                             0.671827
split2_test_score                             0.662539
split3_test_score                             0.667702
split4_test_score                              0.68323
mean_test_score                               0.671425
std_test_score                                0.006819
```

```
rank_test_score                                    11
split0_train_score                                1.0
split1_train_score                                1.0
split2_train_score                           0.999225
split3_train_score                           0.999225
split4_train_score                           0.999225
mean_train_score                             0.999535
std_train_score                               0.00038

                                                  28  \
mean_fit_time                                0.360031
std_fit_time                                 0.132233
mean_score_time                              0.098897
std_score_time                               0.034097
param_svc__C                                       10
param_svc__gamma                                   10
params                 {'svc__C': 10, 'svc__gamma': 10}
split0_test_score                            0.517028
split1_test_score                            0.510836
split2_test_score                            0.517028
split3_test_score                            0.515528
split4_test_score                            0.515528
mean_test_score                               0.51519
std_test_score                               0.002278
rank_test_score                                    15
split0_train_score                                1.0
split1_train_score                                1.0
split2_train_score                           0.999225
split3_train_score                           0.999225
split4_train_score                           0.999225
mean_train_score                             0.999535
std_train_score                               0.00038

                                                   29  \
mean_fit_time                                 0.528295
std_fit_time                                  0.157329
mean_score_time                                0.12621
std_score_time                                 0.04328
param_svc__C                                        10
param_svc__gamma                                   100
params                 {'svc__C': 10, 'svc__gamma': 100}
split0_test_score                             0.504644
split1_test_score                             0.510836
split2_test_score                              0.50774
split3_test_score                             0.509317
split4_test_score                             0.509317
mean_test_score                               0.508371
```

```
std_test_score                                   0.002105
rank_test_score                                        18
split0_train_score                                    1.0
split1_train_score                                    1.0
split2_train_score                               0.999225
split3_train_score                               0.999225
split4_train_score                               0.999225
mean_train_score                                 0.999535
std_train_score                                   0.00038

                                                       30  \
mean_fit_time                                    0.242858
std_fit_time                                      0.04517
mean_score_time                                  0.066413
std_score_time                                   0.010017
param_svc__C                                          100
param_svc__gamma                                    0.001
params              {'svc__C': 100, 'svc__gamma': 0.001}
split0_test_score                                 0.73065
split1_test_score                                0.708978
split2_test_score                                0.662539
split3_test_score                                0.723602
split4_test_score                                0.695652
mean_test_score                                  0.704284
std_test_score                                   0.024115
rank_test_score                                         6
split0_train_score                               0.703876
split1_train_score                               0.712403
split2_train_score                               0.726357
split3_train_score                               0.710302
split4_train_score                                0.71495
mean_train_score                                 0.713577
std_train_score                                  0.007368

                                                      31  \
mean_fit_time                                   0.348261
std_fit_time                                    0.044605
mean_score_time                                 0.067513
std_score_time                                  0.019459
param_svc__C                                         100
param_svc__gamma                                    0.01
params              {'svc__C': 100, 'svc__gamma': 0.01}
split0_test_score                                0.73065
split1_test_score                               0.758514
split2_test_score                                0.71517
split3_test_score                               0.720497
split4_test_score                                0.73913
```

```
mean_test_score                                       0.732792
std_test_score                                        0.015284
rank_test_score                                              2
split0_train_score                                     0.80155
split1_train_score                                    0.796899
split2_train_score                                    0.813953
split3_train_score                                    0.797831
split4_train_score                                    0.793958
mean_train_score                                      0.800838
std_train_score                                       0.006992

                                                           32  \
mean_fit_time                                        0.666763
std_fit_time                                         0.123464
mean_score_time                                       0.06585
std_score_time                                       0.014379
param_svc__C                                              100
param_svc__gamma                                          0.1
params                   {'svc__C': 100, 'svc__gamma': 0.1}
split0_test_score                                    0.705882
split1_test_score                                     0.76161
split2_test_score                                    0.671827
split3_test_score                                    0.708075
split4_test_score                                     0.71118
mean_test_score                                      0.711715
std_test_score                                       0.028734
rank_test_score                                             5
split0_train_score                                   0.990698
split1_train_score                                   0.987597
split2_train_score                                   0.989147
split3_train_score                                   0.984508
split4_train_score                                   0.986057
mean_train_score                                     0.987601
std_train_score                                      0.002188

                                                           33  \
mean_fit_time                                        0.299643
std_fit_time                                         0.044458
mean_score_time                                      0.097411
std_score_time                                        0.03017
param_svc__C                                              100
param_svc__gamma                                          1.0
params                   {'svc__C': 100, 'svc__gamma': 1.0}
split0_test_score                                    0.671827
split1_test_score                                    0.671827
split2_test_score                                    0.662539
split3_test_score                                    0.667702
```

```
split4_test_score                                       0.68323
mean_test_score                                        0.671425
std_test_score                                         0.006819
rank_test_score                                              11
split0_train_score                                          1.0
split1_train_score                                          1.0
split2_train_score                                     0.999225
split3_train_score                                     0.999225
split4_train_score                                     0.999225
mean_train_score                                       0.999535
std_train_score                                         0.00038

                                                           34  \
mean_fit_time                                        0.444609
std_fit_time                                         0.131219
mean_score_time                                      0.130393
std_score_time                                       0.039727
param_svc__C                                              100
param_svc__gamma                                           10
params              {'svc__C': 100, 'svc__gamma': 10}
split0_test_score                                    0.517028
split1_test_score                                    0.510836
split2_test_score                                    0.517028
split3_test_score                                    0.515528
split4_test_score                                    0.515528
mean_test_score                                       0.51519
std_test_score                                       0.002278
rank_test_score                                            15
split0_train_score                                        1.0
split1_train_score                                        1.0
split2_train_score                                   0.999225
split3_train_score                                   0.999225
split4_train_score                                   0.999225
mean_train_score                                     0.999535
std_train_score                                       0.00038

                                                           35
mean_fit_time                                         0.372586
std_fit_time                                          0.107677
mean_score_time                                       0.147969
std_score_time                                        0.066615
param_svc__C                                               100
param_svc__gamma                                           100
params              {'svc__C': 100, 'svc__gamma': 100}
split0_test_score                                     0.504644
split1_test_score                                     0.510836
split2_test_score                                      0.50774
```

```
split3_test_score                               0.509317
split4_test_score                               0.509317
mean_test_score                                 0.508371
std_test_score                                  0.002105
rank_test_score                                       18
split0_train_score                                   1.0
split1_train_score                                   1.0
split2_train_score                              0.999225
split3_train_score                              0.999225
split4_train_score                              0.999225
mean_train_score                                0.999535
std_train_score                                  0.00038

[22 rows x 36 columns]
```

```
[33]: results = pd.DataFrame(grid_search.cv_results_).set_index("rank_test_score").
        ↪sort_index()
      results.T
```

```
[33]: rank_test_score                                       1    \
      mean_fit_time                                 0.202746
      std_fit_time                                  0.027272
      mean_score_time                               0.061584
      std_score_time                                0.008265
      param_svc__C                                       1.0
      param_svc__gamma                                   0.1
      params              {'svc__C': 1.0, 'svc__gamma': 0.1}
      split0_test_score                             0.755418
      split1_test_score                             0.755418
      split2_test_score                             0.712074
      split3_test_score                             0.754658
      split4_test_score                             0.742236
      mean_test_score                               0.743961
      std_test_score                                0.016713
      split0_train_score                            0.827132
      split1_train_score                            0.829457
      split2_train_score                             0.83876
      split3_train_score                            0.831913
      split4_train_score                            0.832688
      mean_train_score                               0.83199
      std_train_score                               0.003907

      rank_test_score                                        2    \
      mean_fit_time                                  0.348261
      std_fit_time                                   0.044605
      mean_score_time                                0.067513
      std_score_time                                 0.019459
```

```
param_svc__C                                              100
param_svc__gamma                                         0.01
params              {'svc__C': 100, 'svc__gamma': 0.01}
split0_test_score                                     0.73065
split1_test_score                                    0.758514
split2_test_score                                     0.71517
split3_test_score                                    0.720497
split4_test_score                                     0.73913
mean_test_score                                      0.732792
std_test_score                                       0.015284
split0_train_score                                    0.80155
split1_train_score                                   0.796899
split2_train_score                                   0.813953
split3_train_score                                   0.797831
split4_train_score                                   0.793958
mean_train_score                                     0.800838
std_train_score                                      0.006992

rank_test_score                                          3   \
mean_fit_time                                    0.304863
std_fit_time                                     0.086008
mean_score_time                                  0.077875
std_score_time                                    0.02544
param_svc__C                                           10
param_svc__gamma                                      0.1
params              {'svc__C': 10, 'svc__gamma': 0.1}
split0_test_score                                0.702786
split1_test_score                                0.767802
split2_test_score                                0.693498
split3_test_score                                0.736025
split4_test_score                                0.745342
mean_test_score                                  0.729091
std_test_score                                   0.027457
split0_train_score                               0.923256
split1_train_score                                0.91938
split2_train_score                               0.925581
split3_train_score                               0.923315
split4_train_score                               0.915569
mean_train_score                                  0.92142
std_train_score                                   0.00354

rank_test_score                                           4   \
mean_fit_time                                     0.223569
std_fit_time                                      0.034292
mean_score_time                                   0.067093
std_score_time                                    0.010337
param_svc__C                                            10
```

```
param_svc__gamma                                     0.01
params               {'svc__C': 10, 'svc__gamma': 0.01}
split0_test_score                                0.739938
split1_test_score                                0.733746
split2_test_score                                0.696594
split3_test_score                                0.720497
split4_test_score                                 0.71118
mean_test_score                                  0.720391
std_test_score                                   0.015566
split0_train_score                               0.751938
split1_train_score                               0.757364
split2_train_score                               0.760465
split3_train_score                               0.759876
split4_train_score                                0.75213
mean_train_score                                 0.756355
std_train_score                                  0.003679

rank_test_score                                         5  \
mean_fit_time                                    0.666763
std_fit_time                                     0.123464
mean_score_time                                   0.06585
std_score_time                                   0.014379
param_svc__C                                          100
param_svc__gamma                                      0.1
params               {'svc__C': 100, 'svc__gamma': 0.1}
split0_test_score                                0.705882
split1_test_score                                 0.76161
split2_test_score                                0.671827
split3_test_score                                0.708075
split4_test_score                                 0.71118
mean_test_score                                  0.711715
std_test_score                                   0.028734
split0_train_score                               0.990698
split1_train_score                               0.987597
split2_train_score                               0.989147
split3_train_score                               0.984508
split4_train_score                               0.986057
mean_train_score                                 0.987601
std_train_score                                  0.002188

rank_test_score                                           6  \
mean_fit_time                                      0.242858
std_fit_time                                        0.04517
mean_score_time                                    0.066413
std_score_time                                     0.010017
param_svc__C                                            100
param_svc__gamma                                      0.001
```

```
params              {'svc__C': 100, 'svc__gamma': 0.001}
split0_test_score                                0.73065
split1_test_score                               0.708978
split2_test_score                               0.662539
split3_test_score                               0.723602
split4_test_score                               0.695652
mean_test_score                                 0.704284
std_test_score                                  0.024115
split0_train_score                              0.703876
split1_train_score                              0.712403
split2_train_score                              0.726357
split3_train_score                              0.710302
split4_train_score                               0.71495
mean_train_score                                0.713577
std_train_score                                 0.007368

rank_test_score                                      7   \
mean_fit_time                                 0.235208
std_fit_time                                  0.041942
mean_score_time                                0.08389
std_score_time                                0.007516
param_svc__C                                       0.1
param_svc__gamma                                   0.1
params              {'svc__C': 0.1, 'svc__gamma': 0.1}
split0_test_score                             0.705882
split1_test_score                             0.718266
split2_test_score                             0.690402
split3_test_score                             0.692547
split4_test_score                             0.708075
mean_test_score                               0.703034
std_test_score                                0.010345
split0_train_score                            0.730233
split1_train_score                            0.724806
split2_train_score                            0.737984
split3_train_score                            0.729667
split4_train_score                            0.725019
mean_train_score                              0.729542
std_train_score                               0.004789

rank_test_score                                       8   \
mean_fit_time                                  0.227487
std_fit_time                                   0.031567
mean_score_time                                0.075283
std_score_time                                  0.00862
param_svc__C                                        1.0
param_svc__gamma                                   0.01
params              {'svc__C': 1.0, 'svc__gamma': 0.01}
```

```
split0_test_score                                   0.702786
split1_test_score                                   0.705882
split2_test_score                                   0.659443
split3_test_score                                   0.726708
split4_test_score                                   0.692547
mean_test_score                                     0.697473
std_test_score                                      0.022019
split0_train_score                                  0.708527
split1_train_score                                  0.700775
split2_train_score                                  0.723256
split3_train_score                                  0.701007
split4_train_score                                  0.718048
mean_train_score                                    0.710323
std_train_score                                     0.009034

rank_test_score                                            9  \
mean_fit_time                                       0.240102
std_fit_time                                        0.039124
mean_score_time                                     0.087023
std_score_time                                      0.014581
param_svc__C                                             0.1
param_svc__gamma                                        0.01
params                {'svc__C': 0.1, 'svc__gamma': 0.01}
split0_test_score                                   0.693498
split1_test_score                                   0.702786
split2_test_score                                   0.653251
split3_test_score                                   0.664596
split4_test_score                                   0.680124
mean_test_score                                     0.678851
std_test_score                                      0.018153
split0_train_score                                  0.675194
split1_train_score                                  0.676744
split2_train_score                                  0.683721
split3_train_score                                  0.681642
split4_train_score                                  0.680868
mean_train_score                                    0.679634
std_train_score                                     0.003172

rank_test_score                                           10  …  \
mean_fit_time                                       0.211701  …
std_fit_time                                        0.032375  …
mean_score_time                                     0.069995  …
std_score_time                                      0.012089  …
param_svc__C                                              10  …
param_svc__gamma                                       0.001  …
params                {'svc__C': 10, 'svc__gamma': 0.001}  …
split0_test_score                                    0.69969  …
```

```
split1_test_score                                      0.674923  …
split2_test_score                                      0.653251  …
split3_test_score                                      0.680124  …
split4_test_score                                       0.68323  …
mean_test_score                                        0.678244  …
std_test_score                                         0.014994  …
split0_train_score                                      0.67907  …
split1_train_score                                     0.682946  …
split2_train_score                                     0.694574  …
split3_train_score                                     0.688613  …
split4_train_score                                     0.687064  …
mean_train_score                                       0.686453  …
std_train_score                                         0.00525  …

rank_test_score                                              21  \
mean_fit_time                                          0.256758
std_fit_time                                           0.029509
mean_score_time                                        0.100254
std_score_time                                         0.016962
param_svc__C                                               0.01
param_svc__gamma                                            0.1
params                {'svc__C': 0.01, 'svc__gamma': 0.1}
split0_test_score                                       0.50774
split1_test_score                                       0.50774
split2_test_score                                       0.50774
split3_test_score                                      0.506211
split4_test_score                                      0.509317
mean_test_score                                         0.50775
std_test_score                                         0.000982
split0_train_score                                     0.507752
split1_train_score                                     0.507752
split2_train_score                                     0.507752
split3_train_score                                     0.508133
split4_train_score                                     0.507359
mean_train_score                                        0.50775
std_train_score                                        0.000245

rank_test_score                                               21  \
mean_fit_time                                           0.230133
std_fit_time                                            0.022421
mean_score_time                                         0.082995
std_score_time                                          0.009681
param_svc__C                                                0.01
param_svc__gamma                                            0.01
params                {'svc__C': 0.01, 'svc__gamma': 0.01}
split0_test_score                                        0.50774
split1_test_score                                        0.50774
```

```
split2_test_score                                 0.50774
split3_test_score                                0.506211
split4_test_score                                0.509317
mean_test_score                                   0.50775
std_test_score                                   0.000982
split0_train_score                               0.507752
split1_train_score                               0.507752
split2_train_score                               0.507752
split3_train_score                               0.508133
split4_train_score                               0.507359
mean_train_score                                  0.50775
std_train_score                                  0.000245

rank_test_score                                        21  \
mean_fit_time                                    0.264388
std_fit_time                                     0.038898
mean_score_time                                  0.100339
std_score_time                                   0.022026
param_svc__C                                         0.01
param_svc__gamma                                    0.001
params              {'svc__C': 0.01, 'svc__gamma': 0.001}
split0_test_score                                 0.50774
split1_test_score                                 0.50774
split2_test_score                                 0.50774
split3_test_score                                0.506211
split4_test_score                                0.509317
mean_test_score                                   0.50775
std_test_score                                   0.000982
split0_train_score                               0.507752
split1_train_score                               0.507752
split2_train_score                               0.507752
split3_train_score                               0.508133
split4_train_score                               0.507359
mean_train_score                                  0.50775
std_train_score                                  0.000245

rank_test_score                                       21  \
mean_fit_time                                   0.276166
std_fit_time                                    0.029502
mean_score_time                                 0.096781
std_score_time                                  0.013255
param_svc__C                                       0.001
param_svc__gamma                                     100
params              {'svc__C': 0.001, 'svc__gamma': 100}
split0_test_score                                0.50774
split1_test_score                                0.50774
split2_test_score                                0.50774
```

```
split3_test_score                                    0.506211
split4_test_score                                    0.509317
mean_test_score                                       0.50775
std_test_score                                       0.000982
split0_train_score                                   0.507752
split1_train_score                                   0.507752
split2_train_score                                   0.507752
split3_train_score                                   0.508133
split4_train_score                                   0.507359
mean_train_score                                      0.50775
std_train_score                                      0.000245

rank_test_score                                          21  \
mean_fit_time                                      0.298828
std_fit_time                                       0.026777
mean_score_time                                    0.106827
std_score_time                                     0.015524
param_svc__C                                          0.001
param_svc__gamma                                         10
params              {'svc__C': 0.001, 'svc__gamma': 10}
split0_test_score                                   0.50774
split1_test_score                                   0.50774
split2_test_score                                   0.50774
split3_test_score                                  0.506211
split4_test_score                                  0.509317
mean_test_score                                     0.50775
std_test_score                                     0.000982
split0_train_score                                 0.507752
split1_train_score                                 0.507752
split2_train_score                                 0.507752
split3_train_score                                 0.508133
split4_train_score                                 0.507359
mean_train_score                                    0.50775
std_train_score                                    0.000245

rank_test_score                                           21  \
mean_fit_time                                       0.236678
std_fit_time                                        0.036106
mean_score_time                                     0.085878
std_score_time                                      0.010228
param_svc__C                                           0.001
param_svc__gamma                                         1.0
params              {'svc__C': 0.001, 'svc__gamma': 1.0}
split0_test_score                                    0.50774
split1_test_score                                    0.50774
split2_test_score                                    0.50774
split3_test_score                                   0.506211
```

```
split4_test_score                                0.509317
mean_test_score                                   0.50775
std_test_score                                   0.000982
split0_train_score                               0.507752
split1_train_score                               0.507752
split2_train_score                               0.507752
split3_train_score                               0.508133
split4_train_score                               0.507359
mean_train_score                                  0.50775
std_train_score                                  0.000245

rank_test_score                                        21  \
mean_fit_time                                    0.284587
std_fit_time                                     0.036894
mean_score_time                                  0.101623
std_score_time                                   0.003895
param_svc__C                                        0.001
param_svc__gamma                                      0.1
params              {'svc__C': 0.001, 'svc__gamma': 0.1}
split0_test_score                                 0.50774
split1_test_score                                 0.50774
split2_test_score                                 0.50774
split3_test_score                                0.506211
split4_test_score                                0.509317
mean_test_score                                   0.50775
std_test_score                                   0.000982
split0_train_score                               0.507752
split1_train_score                               0.507752
split2_train_score                               0.507752
split3_train_score                               0.508133
split4_train_score                               0.507359
mean_train_score                                  0.50775
std_train_score                                  0.000245

rank_test_score                                         21  \
mean_fit_time                                     0.321064
std_fit_time                                      0.033357
mean_score_time                                   0.106591
std_score_time                                    0.016529
param_svc__C                                         0.001
param_svc__gamma                                      0.01
params              {'svc__C': 0.001, 'svc__gamma': 0.01}
split0_test_score                                  0.50774
split1_test_score                                  0.50774
split2_test_score                                  0.50774
split3_test_score                                 0.506211
split4_test_score                                 0.509317
```

```
mean_test_score                                      0.50775
std_test_score                                      0.000982
split0_train_score                                  0.507752
split1_train_score                                  0.507752
split2_train_score                                  0.507752
split3_train_score                                  0.508133
split4_train_score                                  0.507359
mean_train_score                                     0.50775
std_train_score                                     0.000245

rank_test_score                                     21  \
mean_fit_time                                 0.284478
std_fit_time                                  0.041944
mean_score_time                               0.088089
std_score_time                                0.014612
param_svc__C                                       0.1
param_svc__gamma                                    10
params              {'svc__C': 0.1, 'svc__gamma': 10}
split0_test_score                              0.50774
split1_test_score                              0.50774
split2_test_score                              0.50774
split3_test_score                             0.506211
split4_test_score                             0.509317
mean_test_score                                0.50775
std_test_score                                0.000982
split0_train_score                            0.507752
split1_train_score                            0.507752
split2_train_score                            0.507752
split3_train_score                            0.508133
split4_train_score                            0.507359
mean_train_score                               0.50775
std_train_score                               0.000245

rank_test_score                                     21
mean_fit_time                                 0.340115
std_fit_time                                  0.045918
mean_score_time                               0.107322
std_score_time                                0.016258
param_svc__C                                       0.1
param_svc__gamma                                   100
params              {'svc__C': 0.1, 'svc__gamma': 100}
split0_test_score                              0.50774
split1_test_score                              0.50774
split2_test_score                              0.50774
split3_test_score                             0.506211
split4_test_score                             0.509317
mean_test_score                                0.50775
```

```
std_test_score                                0.000982
split0_train_score                            0.507752
split1_train_score                            0.507752
split2_train_score                            0.507752
split3_train_score                            0.508133
split4_train_score                            0.507359
mean_train_score                               0.50775
std_train_score                               0.000245

[21 rows x 36 columns]
```

Let’s only look at the most relevant rows.

[34]:
```
relevant = [
    "mean_test_score",
    "param_svc__gamma",
    "param_svc__C",
    "mean_fit_time",
    "rank_test_score",
]
results = pd.DataFrame(grid_search.cv_results_)[relevant].
 ↪set_index("rank_test_score").sort_index()
results.T
```

[34]:
```
rank_test_score          1         2         3         4         5         6  \
mean_test_score   0.743961  0.732792  0.729091  0.720391  0.711715  0.704284
param_svc__gamma       0.1      0.01       0.1      0.01       0.1     0.001
param_svc__C           1.0       100        10        10       100       100
mean_fit_time     0.202746  0.348261  0.304863  0.223569  0.666763  0.242858

rank_test_score          7         8         9        10  …       21  \
mean_test_score   0.703034  0.697473  0.678851  0.678244  …  0.50775
param_svc__gamma       0.1      0.01      0.01     0.001  …      0.1
param_svc__C           0.1       1.0       0.1        10  …     0.01
mean_fit_time     0.235208  0.227487  0.240102  0.211701  … 0.256758

rank_test_score         21        21        21        21        21        21  \
mean_test_score    0.50775   0.50775   0.50775   0.50775   0.50775   0.50775
param_svc__gamma      0.01     0.001       100        10       1.0       0.1
param_svc__C          0.01      0.01     0.001     0.001     0.001     0.001
mean_fit_time     0.230133  0.264388  0.276166  0.298828  0.236678  0.284587

rank_test_score         21        21        21
mean_test_score    0.50775   0.50775   0.50775
param_svc__gamma      0.01        10       100
param_svc__C         0.001       0.1       0.1
mean_fit_time     0.321064  0.284478  0.340115
```

```
[4 rows x 36 columns]
```

- Other than searching for best hyperparameter values, `GridSearchCV` also fits a new model on the whole training set with the parameters that yielded the best results.
- So we can conveniently call `score` on the test set with a fitted `GridSearchCV` object.

[35]:
```
grid_search.score(X_test, y_test)
```

[35]:
```
0.7376237623762376
```

Why `best_score_` and the score above are different?

[36]:
```
grid_search.best_score_
```

[36]:
```
0.7439609253312309
```

Because one is using test data and the other one is using train data

Let us make a `SVC` pipeline using `best_params_` and see if it matches results from `grid_search.score`

[37]:
```
grid_search.best_params_
```

[37]:
```
{'svc__C': 1.0, 'svc__gamma': 0.1}
```

[38]:
```
best_svm = make_pipeline(StandardScaler(), SVC(C=1.0, gamma=0.1))  # using␣
  ↪grid_search.best_params_
best_svm.fit(X_train, y_train)
best_svm.score(X_train, y_train)
```

[38]:
```
0.8245505269683819
```

[39]:
```
best_svm.score(X_train, y_train) == grid_search.score(X_train, y_train)
```

[39]:
```
True
```

[40]:
```
best_svm.score(X_test, y_test) == grid_search.score(X_test, y_test)
```

[40]:
```
True
```

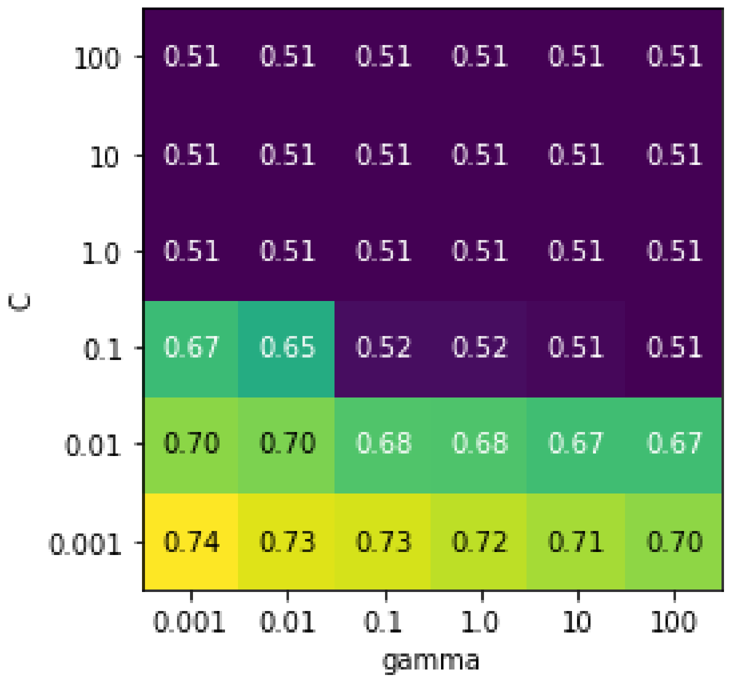
1.3.3 Visualizing the parameter grid as a heatmap

[41]:
```
param_grid
```

[41]:
```
{'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100],
 'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100]}
```

[42]:
```
scores = np.array(results.mean_test_score).reshape(6, 6)

# plot the mean cross-validation scores
```

```python
mglearn.tools.heatmap(
    scores,
    xlabel="gamma",
    xticklabels=param_grid["svc__gamma"],
    ylabel="C",
    yticklabels=param_grid["svc__C"],
    cmap="viridis",
);
```



- Note that the range we pick for the parameters play an important role in hyperparameter optimization.
- For example, consider the following grid and the corresponding results.

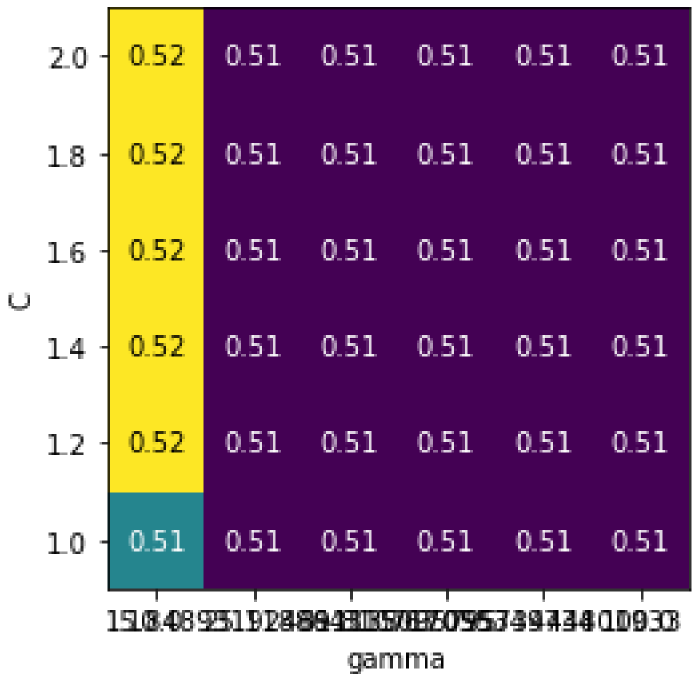
[43]:
```python
def display_heatmap(param_grid, pipe, X_train, y_train):
    grid_search = GridSearchCV(
        pipe, param_grid, cv=5, n_jobs=-1, return_train_score=True
    )
    grid_search.fit(X_train, y_train)
    results = pd.DataFrame(grid_search.cv_results_)
    scores = np.array(results.mean_test_score).reshape(6, 6)

    # plot the mean cross-validation scores
    mglearn.tools.heatmap(
        scores,
        xlabel="gamma",
        xticklabels=param_grid["svc__gamma"],
        ylabel="C",
        yticklabels=param_grid["svc__C"],
        cmap="viridis",
    );
```

1.3.4 Bad range for hyperparameters

[44]:
```
param_grid2 = {"svc__C": np.linspace(1, 2, 6), "svc__gamma": np.logspace(1, 2,␣
↪6), }
param_grid2
```

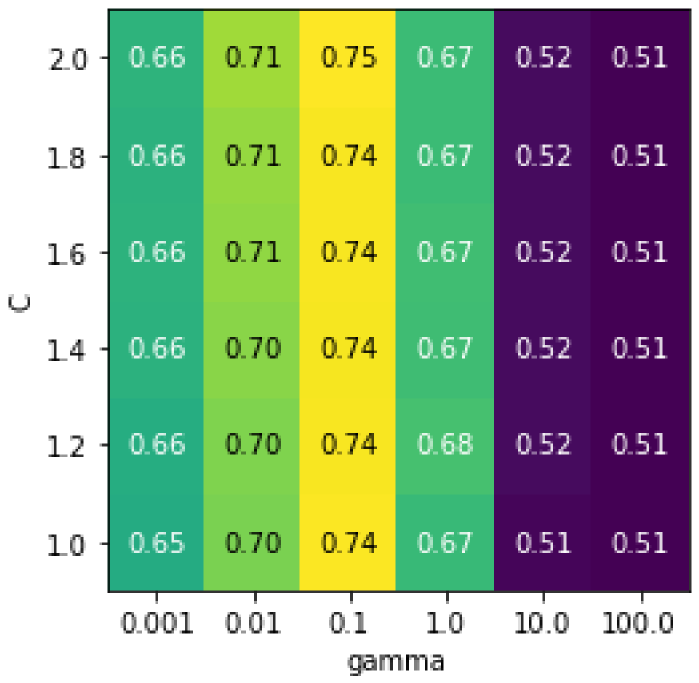
[44]:
```
{'svc__C': array([1. , 1.2, 1.4, 1.6, 1.8, 2. ]),
 'svc__gamma': array([ 10.        ,  15.84893192,  25.11886432,  39.81071706,
         63.09573445, 100.        ])}
```

[45]:
```
display_heatmap(param_grid2, pipe_svm, X_train, y_train)
```



1.3.5 Different range for hyperparameters yields better results!

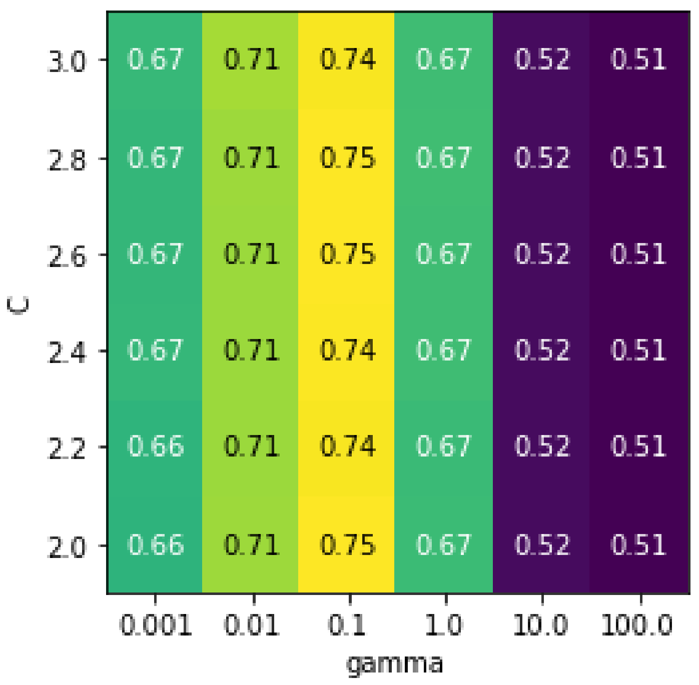
[46]:
```
param_grid3 = {"svc__C": np.linspace(1, 2, 6), "svc__gamma": np.logspace(-3, 2,␣
↪6)}

display_heatmap(param_grid3, pipe_svm, X_train, y_train)
```



It seems like we are getting even better cross-validation results with `C` = 2.0 and `gamma` = 0.1

How about exploring different values of `C` close to 2.0?

[47]:
```
param_grid4 = {"svc__C": np.linspace(2, 3, 6), "svc__gamma": np.logspace(-3, 2,␣
↪6)}

display_heatmap(param_grid4, pipe_svm, X_train, y_train)
```



That's good! We are finding some more options for `C` where the accuracy is 0.75.

The tricky part is we do not know in advance what range of hyperparameters might work the best for the given problem, model, and the dataset.

1.3.6 True/False

- If you get optimal results at the edges of your parameter grid, it might be a good idea to adjust the range of values in your parameter grid. *TRUE*
- Grid search is guaranteed to find best hyperparameters values. *FALSE*

Note `GridSearchCV` allows the `param_grid` to be a list of dictionaries. Sometimes some hyperparameters are applicable only for certain models. For example, in the context of `SVC`, `C` and `gamma` are applicable when the kernel is `rbf` whereas only `C` is applicable for `kernel="linear"`.

1.3.7 Problems with exhaustive grid search

- Required number of models to evaluate grows **exponentially with the dimensionality** of the configuration space.
- Example: Suppose you have
  - 5 hyperparameters
  - 10 different values for each hyperparameter
  - You'll be evaluating $10^5 = 100,000$ models! That is you'll be calling `cross_validate` 100,000 times!
- Exhaustive search may become **infeasible fairly quickly**.
- Other options?

1.4 Randomized hyperparameter search

- Randomized hyperparameter optimization
  - `sklearn.model_selection.RandomizedSearchCV`
- Samples configurations at **random until certain budget** (e.g., time) is exhausted

[48]:
```
from sklearn.model_selection import RandomizedSearchCV


param_grid = {
    "svc__gamma": [0.001, 0.01, 0.1, 1.0, 10, 100],
    "svc__C": [0.001, 0.01, 0.1, 1.0, 10, 100],
}

print("Grid size:", np.prod([len(v) for v in param_grid.values()]))
param_grid
```

```
Grid size: 36
```

[48]:
```
{'svc__gamma': [0.001, 0.01, 0.1, 1.0, 10, 100],
 'svc__C': [0.001, 0.01, 0.1, 1.0, 10, 100]}
```

[49]:
```
random_search = RandomizedSearchCV(
    pipe_svm, param_distributions=param_grid, n_jobs=-1, n_iter=10, cv=5,␣
  ↪random_state=123
)
random_search.fit(X_train, y_train);
```

[50]:
```
pd.DataFrame(random_search.cv_results_)[relevant].set_index("rank_test_score").
  ↪sort_index().T
```

[50]:
```
rank_test_score          1         2         3         4         5         6  \
mean_test_score   0.732792  0.711715  0.678851  0.652824  0.508371   0.50775
param_svc__gamma      0.01       0.1      0.01     0.001       100     0.001
param_svc__C           100       100       0.1       1.0       1.0      0.01
mean_fit_time     0.361828  0.765091  0.348906  0.205961  0.425573  0.331889

rank_test_score         6        6        6         6
mean_test_score   0.50775  0.50775  0.50775   0.50775
param_svc__gamma      0.1      100      100     0.001
param_svc__C         0.01     0.01    0.001       0.1
mean_fit_time     0.36394  0.35472  0.58357  0.239231
```

1.4.1 `n_iter`

- Note the `n_iter`, we didn't need this for `GridSearchCV`.
- Larger `n_iter` will take longer but it'll do more searching.
  - Remember you still need to multiply by number of folds!
  - Thus, number of models to train will be `n_iter * cv`

- I have also set `random_state` but you don't have to do it.

1.4.2 Range of `C`

- Note the exponential range for `C`. This is quite common.
- There is no point trying $C = \{1, 2, 3 \ldots, 100\}$ because $C = 1, 2, 3$ are too similar to each other.
- Often we're trying to find an order of magnitude, e.g. $C = \{0.01, 0.1, 1, 10, 100\}$.
- We can also write that as $C = \{10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}\}$.
- Or, in other words, C values to try are 10^n for $n = -2, -1, 0, 1, 2$ which is basically what we have above.

(Optional) Another thing we can do is give probability distributions to draw from:

[51]:
```
from scipy.stats import expon, lognorm, loguniform, randint, uniform
```

[52]:
```
param_dist = {
    "svc__C": uniform(0.1, 1e4),  # loguniform(1e-3, 1e3),
    "svc__gamma": loguniform(1e-5, 1e3),
}
```

[53]:
```
random_search = RandomizedSearchCV(
    pipe_svm, param_dist, n_iter=100, verbose=1, n_jobs=-1, random_state=123
)
```

[54]:
```
random_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

[54]:
```
RandomizedSearchCV(estimator=Pipeline(steps=[('standardscaler',
                                              StandardScaler()),
                                             ('svc', SVC())]),
                   n_iter=100, n_jobs=-1,
                   param_distributions={'svc__C':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f51b0072ef0>,
                                        'svc__gamma':
<scipy.stats._distn_infrastructure.rv_frozen object at 0x7f51affe9ae0>},
                   random_state=123, verbose=1)
```

[55]:
```
random_search.best_score_
```

[55]:
```
0.7383804780493433
```

[56]:
```
pd.DataFrame(random_search.cv_results_)[relevant].set_index("rank_test_score").
↪sort_index().T
```

[56]:
```
rank_test_score            1            2            3            4   \
mean_test_score      0.73838       0.7359     0.735277     0.733415
param_svc__gamma     0.00271     0.001946      0.00283     0.003148
param_svc__C     3427.738338  6964.791856  2865.466167  4258.402903
```

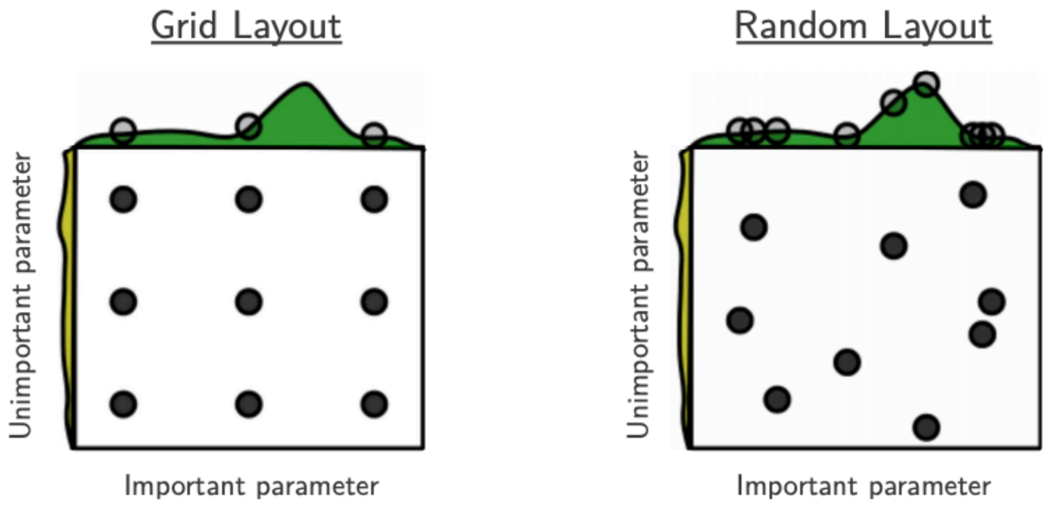
```
mean_fit_time        1.800691     2.129358     1.229454     2.363242

rank_test_score             5            6            7            8   \
mean_test_score      0.731556     0.729716     0.729068     0.728454
param_svc__gamma     0.003834     0.015524      0.00067     0.005593
param_svc__C      7224.533826  1511.374523  3617.986556  2408.658977
mean_fit_time        4.471639      3.34881     0.882294     2.175671

rank_test_score             9            9  …            82           82  \
mean_test_score       0.72845      0.72845  …      0.508371     0.508371
param_svc__gamma      0.00089     0.000458  …     33.017913    51.754341
param_svc__C      7636.928414  9053.515757  …   5826.810879  7087.073954
mean_fit_time        1.967938     0.846148  …      0.356378     0.358163

rank_test_score            82           82           82           96  \
mean_test_score      0.508371     0.508371     0.508371      0.50713
param_svc__gamma   126.722678    84.511472   120.952433   766.440217
param_svc__C      6648.824488  6007.085678   957.225166  4830.442643
mean_fit_time        0.354666     0.345583     0.454863     0.313766

rank_test_score            96           96           96          96
mean_test_score       0.50713      0.50713      0.50713     0.50713
param_svc__gamma   739.685456   918.053216   263.398099   806.79529
param_svc__C      9019.213727  7049.688305  3370.763834   26.980646
mean_fit_time        0.286004     0.282581     0.290457      0.2878

[4 rows x 100 columns]
```

- This is a bit fancy. What's nice is that you can have it concentrate more on certain values by setting the distribution.

1.4.3 Advantages of `RandomizedSearchCV`

- Faster compared to `GridSearchCV`.
- Adding parameters that do not influence the performance does not affect efficiency.
- Works better when some parameters are more important than others.
- In general, I recommend using `RandomizedSearchCV` rather than `GridSearchCV`.

1.4.4 Advantages of `RandomizedSearchCV`



Source: Bergstra and Bengio, Random Search for Hyper-Parameter Optimization, JMLR 2012.

- The yellow on the left shows how your scores are going to change when you vary the unimportant hyperparameter.
- The green on the top shows how your scores are going to change when you vary the important hyperparameter.
- You don't know in advance which hyperparameters are important for your problem.
- In the left figure, 6 of the 9 searches are useless because they are only varying the unimportant parameter.
- In the right figure, all 9 searches are useful.

1.5 Fancier methods (optional)

- Both `GridSearchCV` and `RandomizedSearchCV` do each trial independently.
- What if you could learn from your experience, e.g. learn that `max_depth=3` is bad?
  - That could save time because you wouldn't try combinations involving `max_depth=3` in the future.
- We can do this with `scikit-optimize`, which is a completely different package from `scikit-learn`
- It uses a technique called "model-based optimization" and we'll specifically use "Bayesian optimization".
  - In short, it uses machine learning to predict what hyperparameters will be good.
  - Machine learning on machine learning!
- This is an active research area and there are sophisticated packages for this.

Here are some examples - hyperopt-sklearn - auto-sklearn - SigOptSearchCV - TPOT - hyperopt - hyperband - SMAC - MOE - pybo - spearmint - BayesOpt

```
[ ]:
```

1.5.1 Questions for class discussion (hyperparameter optimization)

- Suppose you have 10 hyperparameters, each with 4 possible values. If you run `GridSearchCV` with this parameter grid, how many cross-validation experiments it would carry out?
- `GridSearchCV` exhaustively searches the grid and so it's guaranteed to give you the optimal hyperparameters for the given problem.
- It is possible to get different hyperparameters in different runs of `RandomizedSearchCV`.
- Suppose you have 10 hyperparameters and each takes 4 values. If you run `RandomizedSearchCV` with this parameter grid, how many cross-validation experiments it would carry out?

1.6 Optimization bias/Overfitting of the validation set

1.6.1 Overfitting of the validation error

- Why do we need to evaluate the model on the test set in the end?
- Why not just use cross-validation on the whole dataset?
- While carrying out hyperparameter optimization, we usually try over many possibilities.

- If our dataset is small and if your validation set is hit too many times, we suffer from **optimization bias** or **overfitting the validation set**.

1.6.2 Optimization bias of parameter learning

- Overfitting of the training error
- An example:
  - During training, we could search over tons of different decision trees.
  - So we can get "lucky" and find one with low training error by chance.

1.6.3 Optimization bias of hyper-parameter learning

- Overfitting of the validation error
- An example:
  - Here, we might optimize the validation error over 1000 values of `max_depth`.
  - One of the 1000 trees might have low validation error by chance.

1.6.4 Example 1: Optimization bias (optional)

Consider a multiple-choice (a,b,c,d) "test" with **10 questions**: - If you choose answers randomly, expected grade is 25% (no bias). - If you fill out two tests randomly and pick the best, expected grade is 33%. - Optimization bias of ~8%. - If you take the best among 10 random tests, expected grade is ~47%. - If you take the best among 100, expected grade is ~62%. - If you take the best among 1000, expected grade is ~73%. - If you take the best among 10000, expected grade is ~82%. - You have so many "chances" that you expect to do well.

But on new questions the "random choice" accuracy is still 25%.

[57]:
```
# (optional) Code attribution: Rodolfo Lourenzutti
number_tests = [1, 2, 10, 100, 1000, 10000]
for ntests in number_tests:
    y = np.zeros(10000)
    for i in range(10000):
        y[i] = np.max(np.random.binomial(10.0, 0.25, ntests))
    print(
        "The expected grade among the best of %d tests is : %0.2f"
        % (ntests, np.mean(y) / 10.0)
    )
```

```
The expected grade among the best of 1 tests is : 0.25
The expected grade among the best of 2 tests is : 0.33
The expected grade among the best of 10 tests is : 0.47
The expected grade among the best of 100 tests is : 0.62
The expected grade among the best of 1000 tests is : 0.73
The expected grade among the best of 10000 tests is : 0.83
```

1.6.5 Example 2: Optimization bias (optional)

- If we instead used a **100-question** test then:
  - Expected grade from best over 1 randomly-filled test is 25%.
  - Expected grade from best over 2 randomly-filled test is ~27%.
  - Expected grade from best over 10 randomly-filled test is ~32%.
  - Expected grade from best over 100 randomly-filled test is ~36%.
  - Expected grade from best over 1000 randomly-filled test is ~40%.
  - Expected grade from best over 10000 randomly-filled test is ~43%.
- The optimization bias **grows with the number of things we try**.
  - "Complexity" of the set of models we search over.
- But, optimization bias **shrinks quickly with the number of examples**.
  - But it's still non-zero and growing if you over-use your validation set!

[58]:
```
# (optional) Code attribution: Rodolfo Lourenzutti
number_tests = [1, 2, 10, 100, 1000, 10000]
for ntests in number_tests:
    y = np.zeros(10000)
    for i in range(10000):
        y[i] = np.max(np.random.binomial(100.0, 0.25, ntests))
    print(
        "The expected grade among the best of %d tests is : %0.2f"
        % (ntests, np.mean(y) / 100.0)
    )
```

```
The expected grade among the best of 1 tests is : 0.25
The expected grade among the best of 2 tests is : 0.28
The expected grade among the best of 10 tests is : 0.32
The expected grade among the best of 100 tests is : 0.36
The expected grade among the best of 1000 tests is : 0.40
The expected grade among the best of 10000 tests is : 0.43
```

1.6.6 Optimization bias on the Spotify dataset

Notice the unrealistic setting `test_size=0.99`. We do that here for demonstration only.

[59]:
```
X_train_tiny, X_test_big, y_train_tiny, y_test_big = train_test_split(
    X_spotify, y_spotify, test_size=0.99, random_state=42
)
```

[60]:
```
X_train_tiny.shape
```

[60]:
```
(20, 13)
```

[61]:
```
X_train_tiny.head()
```

[61]:
```
      acousticness  danceability  duration_ms  energy  instrumentalness  key  \
130       0.055100         0.547       251093   0.643          0.000000    1
1687      0.000353         0.420       210240   0.929          0.000747    7
871       0.314000         0.430       193427   0.734          0.000286    9
1123      0.082100         0.725       246653   0.711          0.000000   10
1396      0.286000         0.616       236960   0.387          0.000000    9

      liveness  loudness  mode  speechiness    tempo  time_signature  valence
130     0.2670    -8.904     1       0.2270  143.064             4.0   0.1870
1687    0.1220    -3.899     0       0.1210  127.204             4.0   0.3180
871     0.0808   -10.043     0       0.1020  133.992             4.0   0.0537
1123    0.0931    -4.544     1       0.0335   93.003             4.0   0.4760
1396    0.2770    -6.079     0       0.0335   81.856             4.0   0.4700
```

[62]:
```
pipe = make_pipeline(StandardScaler(), SVC())
```

[63]:
```
from sklearn.model_selection import RandomizedSearchCV

param_grid = {
    "svc__gamma": 10.0 ** np.arange(-20, 10),
    "svc__C": 10.0 ** np.arange(-20, 10),
}

print("Grid size:", np.prod([len(v) for v in param_grid.values()]))
param_grid
```

```
Grid size: 900
```

[63]:
```
{'svc__gamma': array([1.e-20, 1.e-19, 1.e-18, 1.e-17, 1.e-16, 1.e-15, 1.e-14,
1.e-13,
        1.e-12, 1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05,
        1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03,
        1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+08, 1.e+09]),
 'svc__C': array([1.e-20, 1.e-19, 1.e-18, 1.e-17, 1.e-16, 1.e-15, 1.e-14,
1.e-13,
        1.e-12, 1.e-11, 1.e-10, 1.e-09, 1.e-08, 1.e-07, 1.e-06, 1.e-05,
        1.e-04, 1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03,
        1.e+04, 1.e+05, 1.e+06, 1.e+07, 1.e+08, 1.e+09])}
```

[64]:
```
random_search = RandomizedSearchCV(
    pipe, param_distributions=param_grid, n_jobs=-1, n_iter=900, cv=5,␣
 ↪random_state=123
)
random_search.fit(X_train_tiny, y_train_tiny);
```

[65]:
```
pd.DataFrame(random_search.cv_results_)[relevant].set_index("rank_test_score").
 ↪sort_index().T
```

[65]:
```
rank_test_score             1            1             3         3    \
mean_test_score           0.8          0.8          0.75      0.75
param_svc__gamma          0.0          0.0         0.001     0.001
param_svc__C     1000000000.0  100000000.0  1000000000.0   10000.0
mean_fit_time         0.01006     0.009334      0.010692  0.018828

rank_test_score         3           3          3         3            3    \
mean_test_score      0.75        0.75       0.75      0.75         0.75
param_svc__gamma    0.001         0.0      0.001     0.001        0.001
param_svc__C     1000000.0  10000000.0  100000.0    1000.0   10000000.0
mean_fit_time     0.011188     0.00936  0.009404  0.012803     0.010117

rank_test_score           3   …       20        20        20        20   \
mean_test_score        0.75   …     0.65      0.65      0.65      0.65
param_svc__gamma      0.001   …      0.0       0.0       0.0       0.0
param_svc__C     100000000.0  …      0.0       0.0       0.0       0.0
mean_fit_time      0.009571   …  0.010388  0.010264  0.009756  0.010275

rank_test_score        20        20        20        20            20   \
mean_test_score      0.65      0.65      0.65      0.65          0.65
param_svc__gamma      0.0       0.0       0.0  100000.0  1000000000.0
param_svc__C          0.0       0.0       0.0       0.0  1000000000.0
mean_fit_time    0.012348  0.009535  0.010359  0.013648       0.00904

rank_test_score           900
mean_test_score          0.55
param_svc__gamma          0.0
param_svc__C     1000000000.0
mean_fit_time        0.010052

[4 rows x 900 columns]
```

Given the results: one might claim that we found a model that performs with 0.8 accuracy on our dataset.

- Do we really **believe that 0.80** is a good estimate of our test data?
- Do we really believe that `gamma`=0.0 and C=1_000_000_000 are the best hyperparameters?

- Let's find out the **test score** with this best model.

```
[66]: random_search.score(X_test, y_test)
```

```
[66]: 0.6163366336633663
```

- The results above are **overly optimistic**.
  - because our training data is very small and so our validation splits in cross validation would be small.
  - because of the small dataset and the fact that we **hit the small validation set 900 times** and it's possible that we got lucky on the validation set!

- As we suspected, the best cross-validation score is not a good estimate of our test data; it is overly optimistic.
- We can trust this test score because the test set is of good size.

```
[67]: X_test_big.shape
```

```
[67]: (1997, 13)
```

1.6.7 Overfitting of the validation data

The following plot demonstrates what happens during overfitting of the validation data.

Source

- Thus, not only can we not trust the cv scores, we also cannot trust cv's ability to choose the best hyperparameters.

1.6.8 Why do we need a test set?

- This is why we need a test set.
- The frustrating part is that if our dataset is small then our test set is also small .
- But we don't have a lot of better alternatives, unfortunately, if we have a **small dataset**.

1.6.9 When test score is much lower than CV score

- What to do if your test score is much lower than your cross-validation score:
  - Try simpler models and **use the test set a couple of times**; it's not the end of the world.
  - **Communicate** this clearly when you report the results.

1.6.10 Large datasets solve many of these problems

- With infinite amounts of training data, overfitting would not be a problem and you could have your test score = your train score.
  - Overfitting happens because you only see a bit of data and you learn patterns that are overly specific to your sample.

  - If you saw "all" the data, then the notion of "overly specific" would not apply.
- So, more data will make your test score better and robust.

1.6.11 Questions for you

1.6.12 Would you trust the model?

- You have a dataset and you give me half of it. I build a model using all the data you have given me and I tell you that the model accuracy is 0.99. Would it classify the rest of the data with similar accuracy?

1. Probably
2. **Probably not** No validation?

1.6.13 Would you trust the model?

- You have a dataset and you give me half of it. I build a model using 80% of the data given to me and report the accuracy of 0.95 on the remaining 20% of the data. Would it classify the rest of the data with similar accuracy?

1. **Probably**
2. Probably not

1.6.14 Would you trust the model?

- You have a dataset and you give me 1/10th of it. The dataset given to me is rather small and so I split it into 96% train and 4% validation split. I carry out hyperparameter optimization using a single 4% validation split and report validation accuracy of 0.97. Would it classify the rest of the data with similar accuracy?

1. Probably
2. **Probably not** Overfitting of the validation error happened due to small sample size. Not even n-fold.

1.7 Final comments and summary

Automated hyperparameter optimization

- Advantages
  - reduce human effort
  - less prone to error and improve reproducibility
  - data-driven approaches may be effective
- Disadvantages
  - may be hard to incorporate intuition
  - be careful about overfitting on the validation set

Often, especially on typical datasets, we get back `scikit-learn`'s default hyperparameter values. This means that the defaults are well chosen by `scikit-learn` developers!

- The problem of finding the best values for the important hyperparameters is tricky because
  - You may have a lot of them (e.g. deep learning).

    - You may have multiple hyperparameters which may interact with each other in unexpected ways.

- The **best settings depend on the specific data/problem**.

1.8 Optional readings and resources

- Preventing “overfitting” of cross-validation data by Andrew Ng