Schedule

Block 2 (11:00 am - 12:30 pm)

(3) Data preprocessing in Python

(4) Evaluating & tuning machine learning classifiers

Lunch break (12:30 - 1:30 pm)

Classifier in scikit-learn (subset of the most popular ones)

Linear Models

Logistic Regression (from sklearn.linear_model import LogisticRegression)
Perceptron (from sklearn.linear_model import Perceptron)
SGD Classifier (from sklearn.linear_model import SGDClassifier)
SVC (from sklearn.svm import SVC)
…

Nearest Neighbors

KNeighbors Classifier (from sklearn.neighbors import KNeighborsClassifier)
…

Naive Bayes

GaussianNB (from sklearn.naive_bayes import GaussianNB)
…

Decision Trees

DecisionTreeClassifier (from sklearn.tree import DecisionTreeClassifier)Ensemble Methods
RandomForestClassifier (from sklearn.ensemble import RandomForestClassifier)
HistGradientBoostingClassifier (from sklearn.ensemble import GradientBoostingClassifier)
…

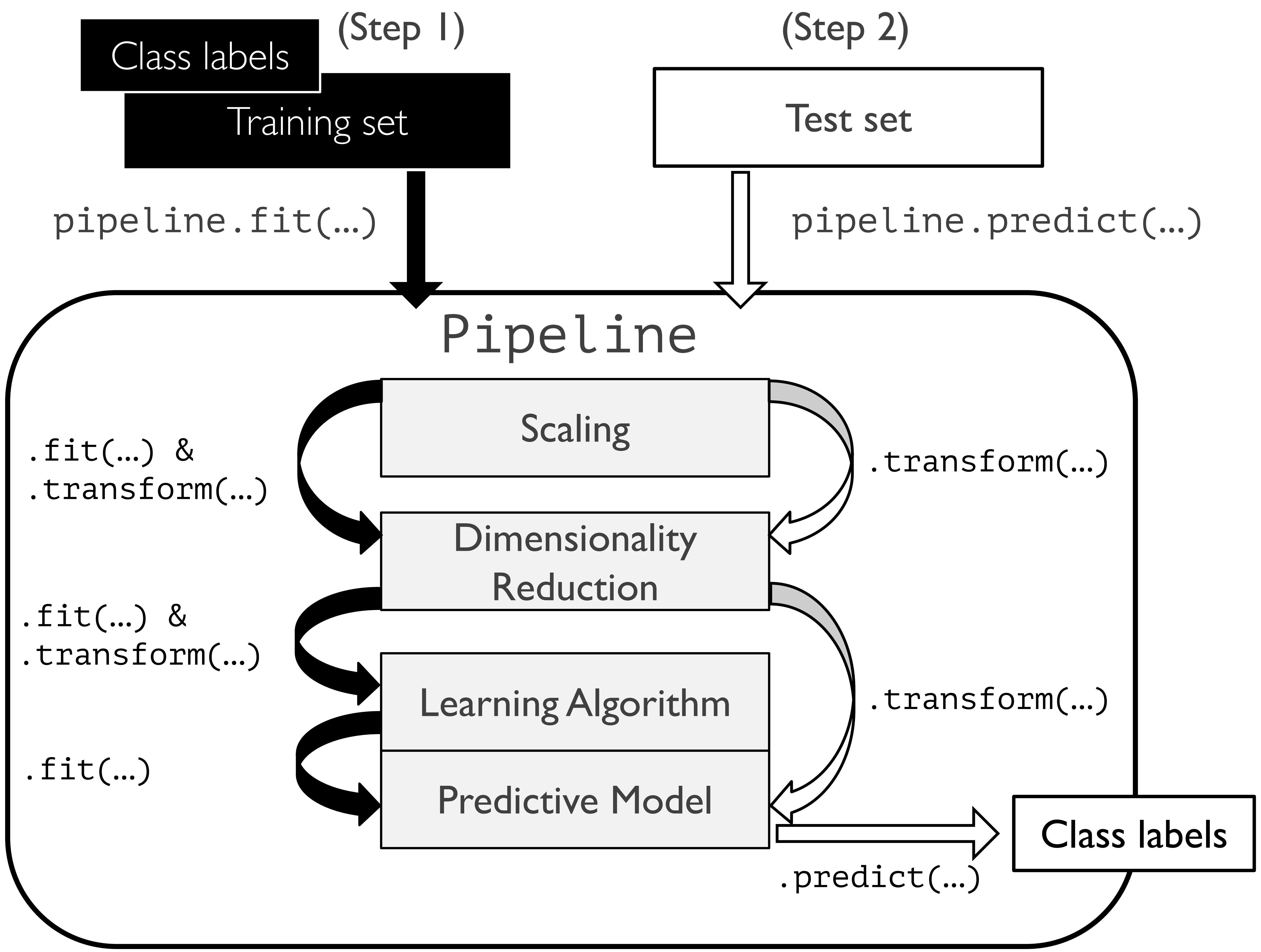
(More) Ensemble Methods

StackingClassifier (from sklearn.ensemble import StackingClassifier)
VotingClassifier (from sklearn.ensemble import VotingClassifier)
…

Neural Network Models

MLPClassifier (from sklearn.neural_network import MLPClassifier)
…

Scikit-learn pipelines



Scikit-learn pipelines

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier(n_neighbors=3)
)
```

Scikit-learn pipelines

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline


pipe = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier(n_neighbors=3)
)
```

```python
pipe.fit(X_train, y_train)
```

```
▸ Pipeline
  ▸ StandardScaler
  ▸ KNeighborsClassifier
```

```python
pipe.predict(X_test)[:10]
```

```
array([2, 2, 2, 1, 0, 1, 1, 0, 0, 1])
```

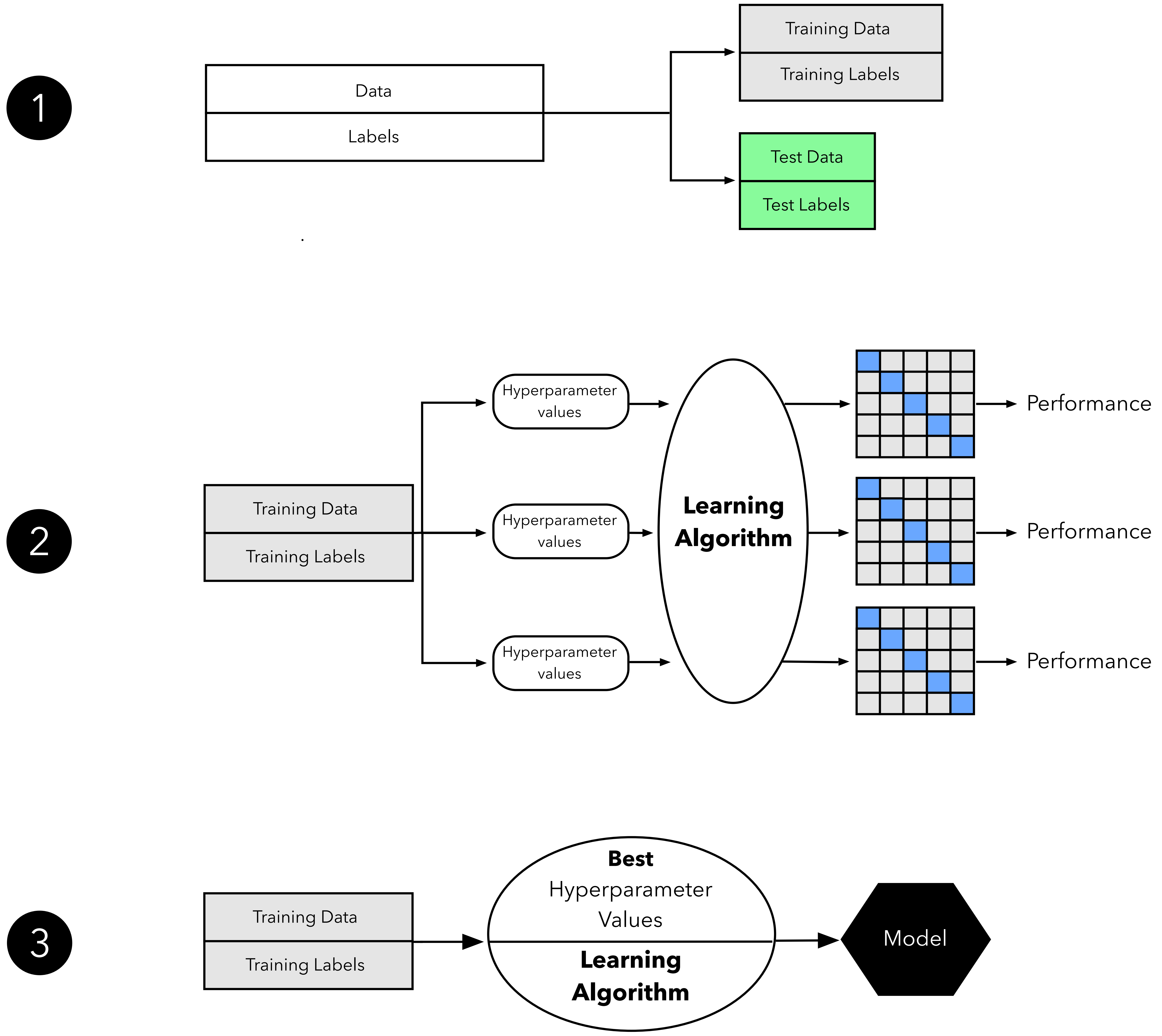
```python
pipe.score(X_test, y_test)
```

```
0.92
```

Model evaluation

Model selection

with k-fold cross-validation



Cornell University

Computer Science > Machine Learning

arXiv:1811.12808 (cs)

[Submitted on 13 Nov 2018 (v1), last revised 11 Nov 2020 (this version, v3)]

Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning

Sebastian Raschka

https://arxiv.org/abs/1811.12808

Contents

2

k-fold cross-validation score in scikit-learn

```
from sklearn.model_selection import cross_val_score


scores = cross_val_score(pipe, X=X_train, y=y_train, cv=5)
scores.mean()
```

```
0.9066666666666666
```

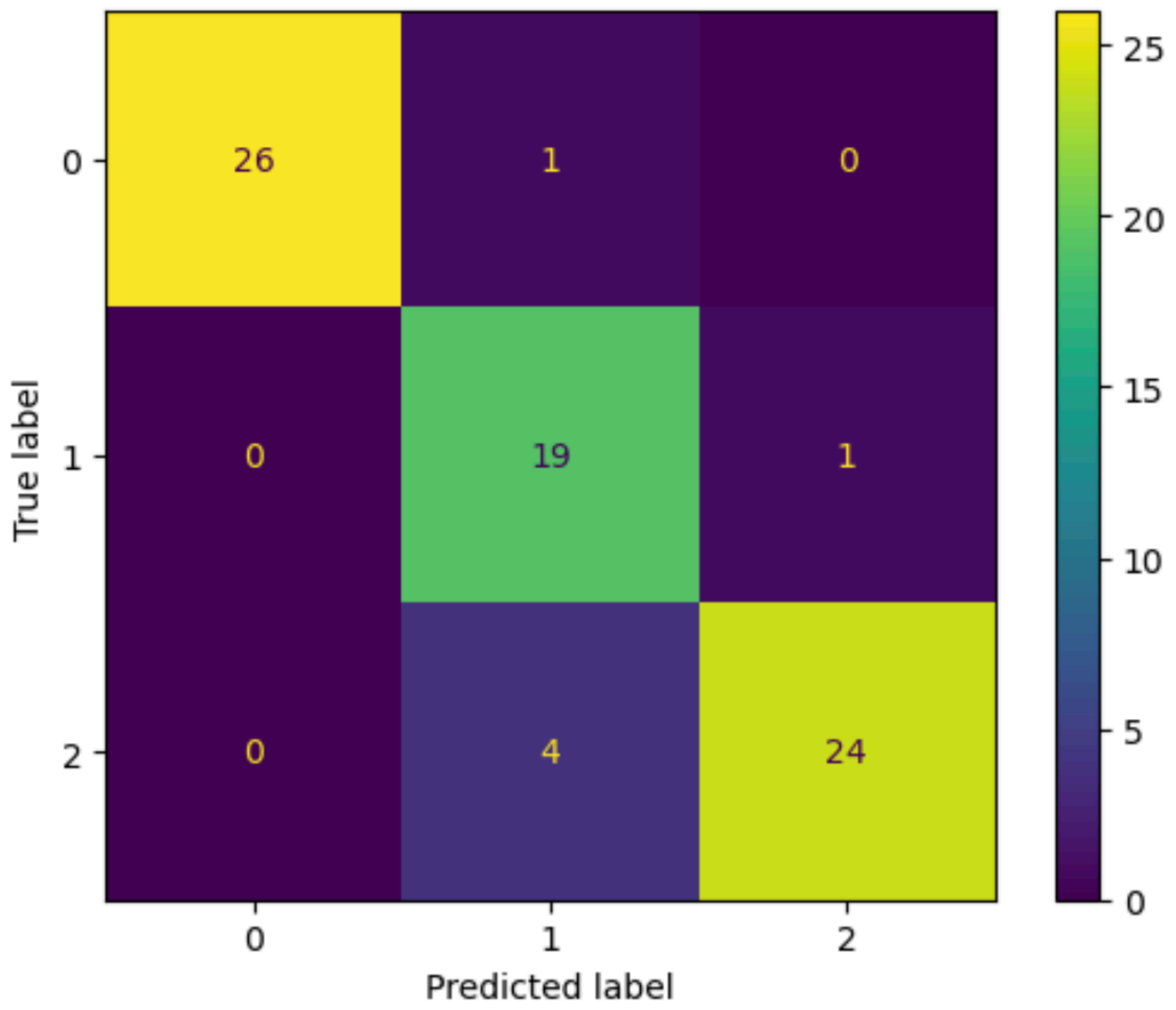
Confusion matrices to look at failure cases

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)

cmat = confusion_matrix(y_test, y_pred)
```

```
disp = ConfusionMatrixDisplay(confusion_matrix=cmat)
disp.plot();
```



Hyperparameter tuning

Manual search: trying one thing at a time

The idea: change one thing and see if it makes it better or worse

Manual search: trying one thing at a time

The idea: change one thing and see if it makes it better or worse

+ Can yield great insights

– It is very laborious

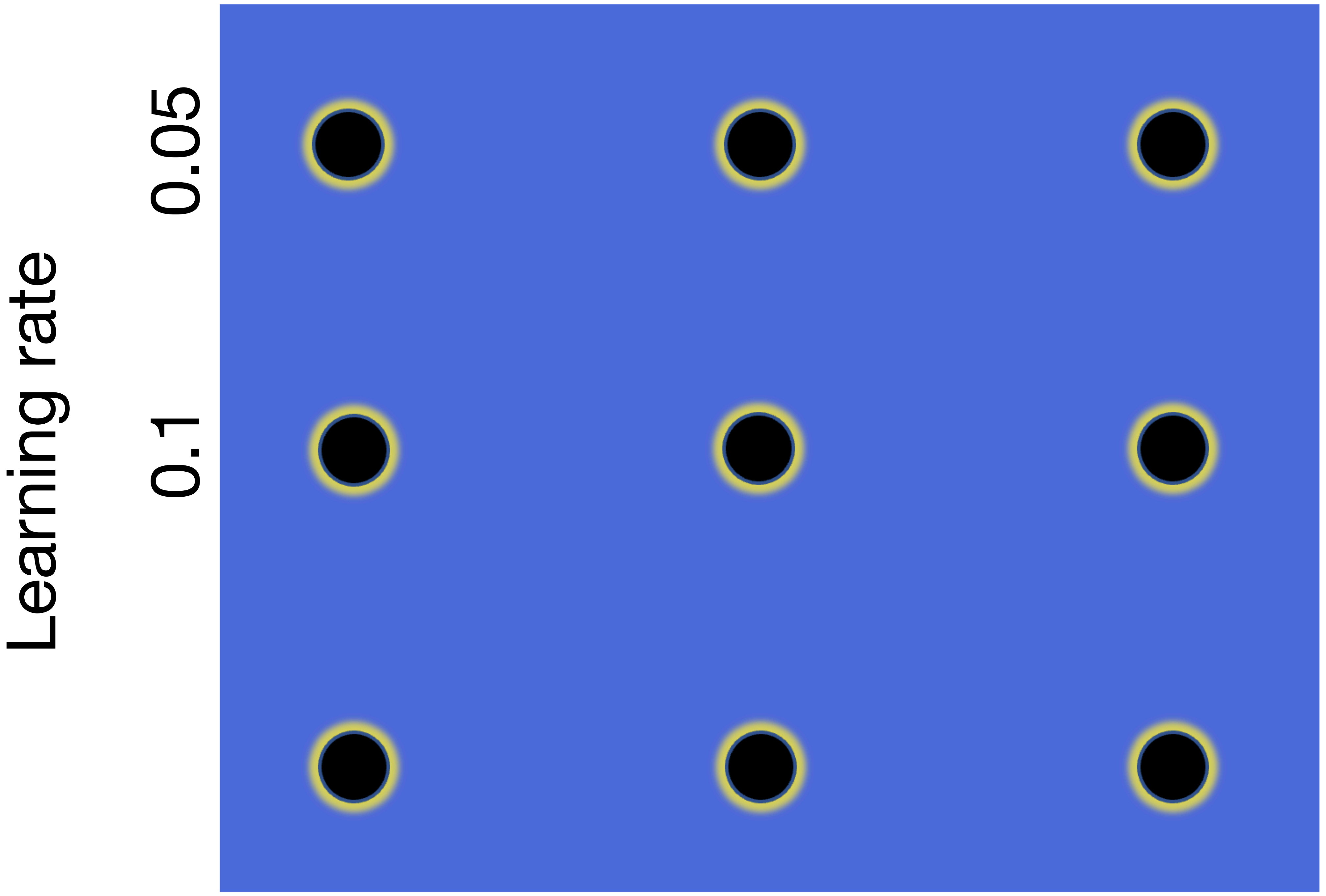
Note: Often, changing e.g., the batch size also requires tuning other parameters from scratch again (like the learning rate)

Grid search: a brute-force search

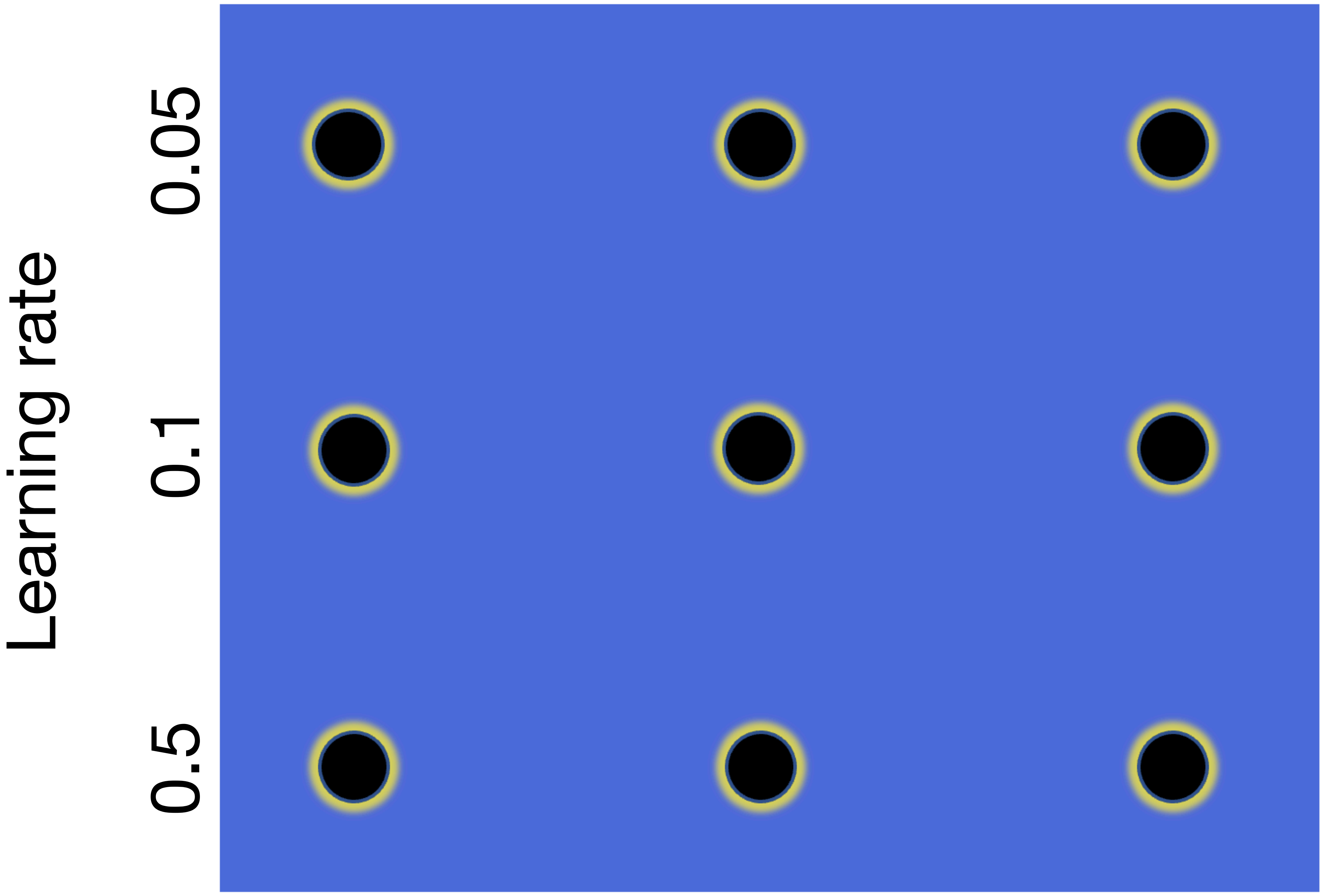
Suppose we only consider the following two hyperparameters:

- Learning rate: 0.05, 0.1, 0.5
- Hidden layer size: 50, 100, 150

Grid search: a brute-force search



Grid search: a brute-force search

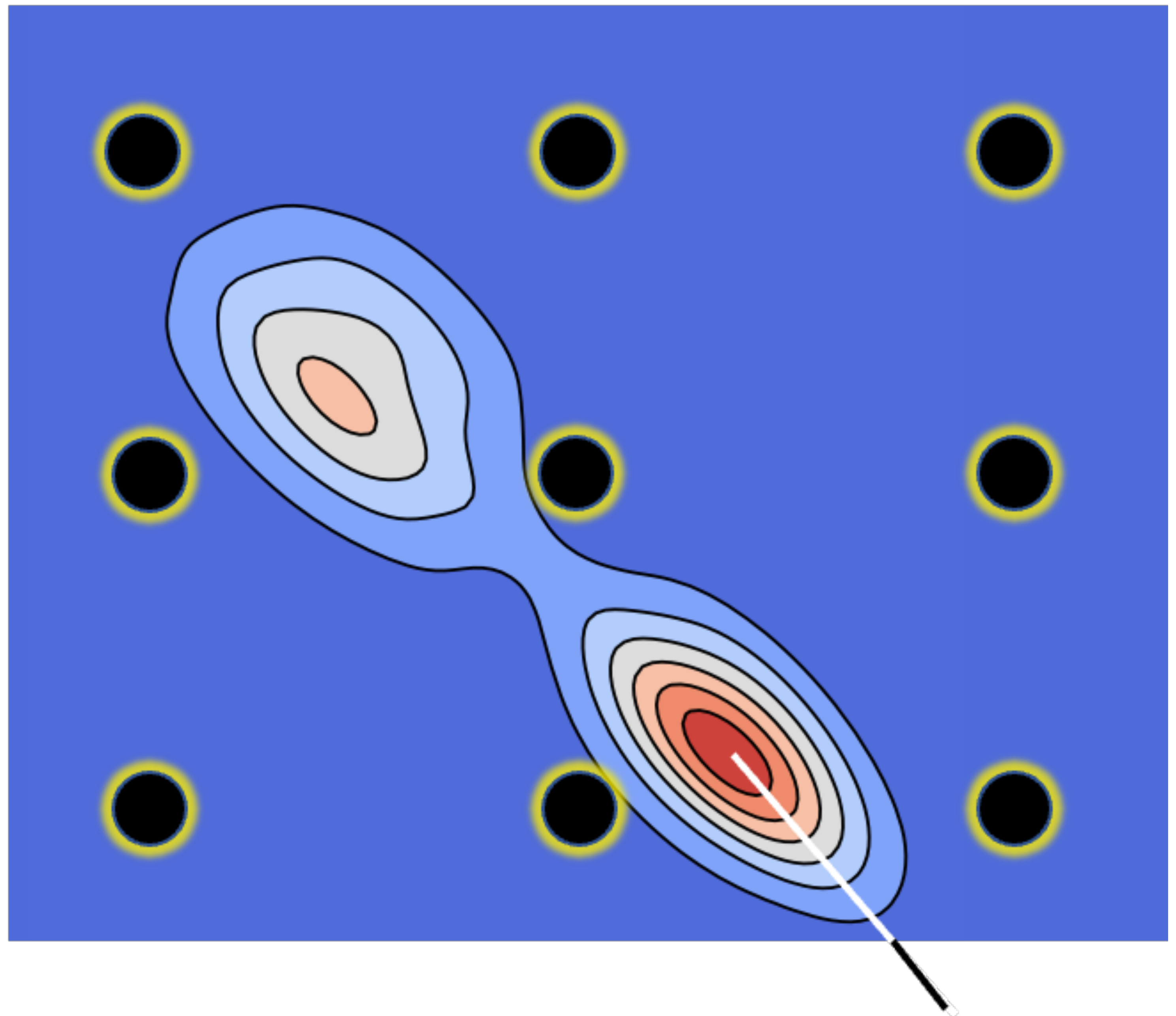


Randomized search

Draw parameters from distributions

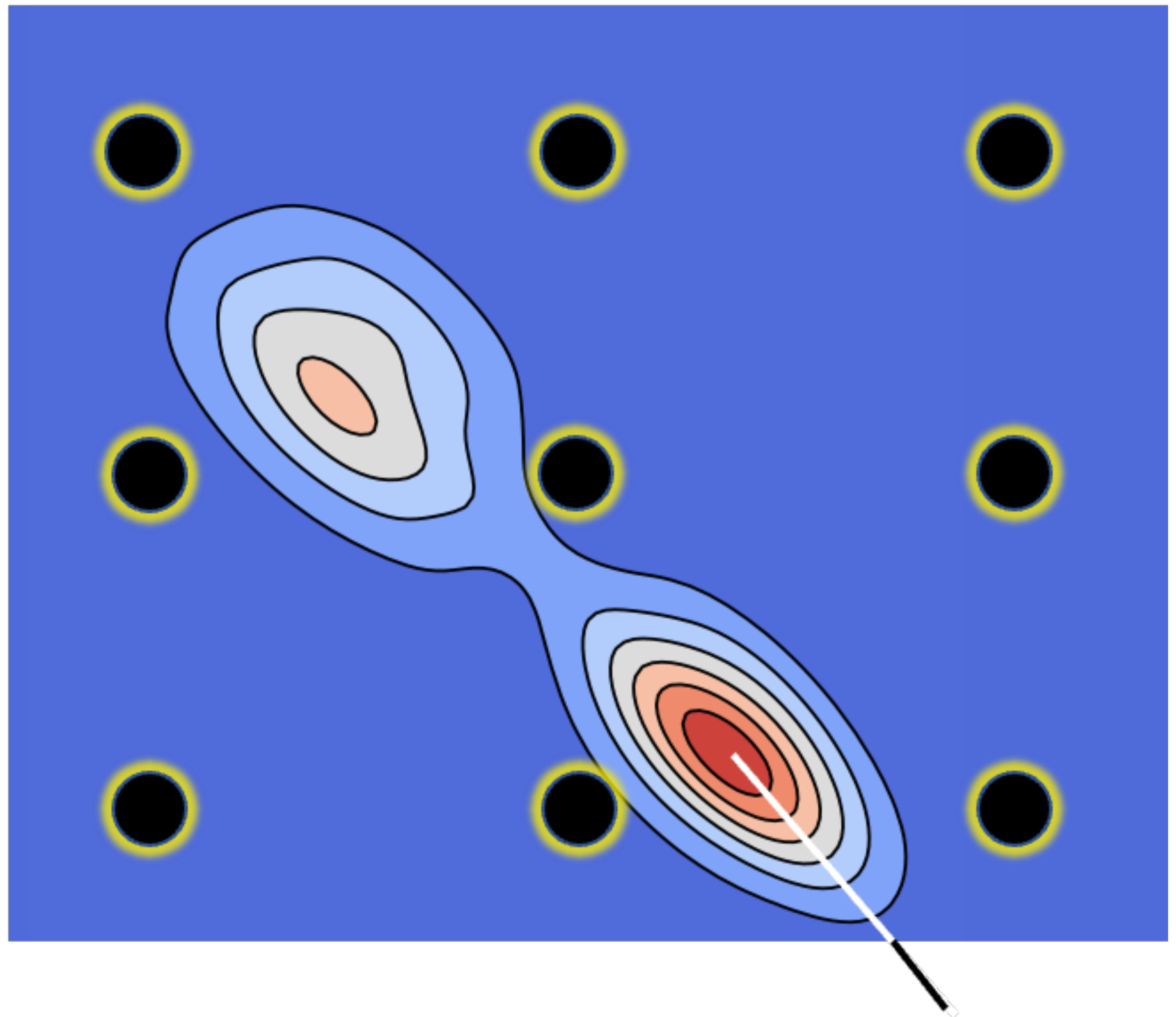
Explore a wider range of parameter settings

Grid search



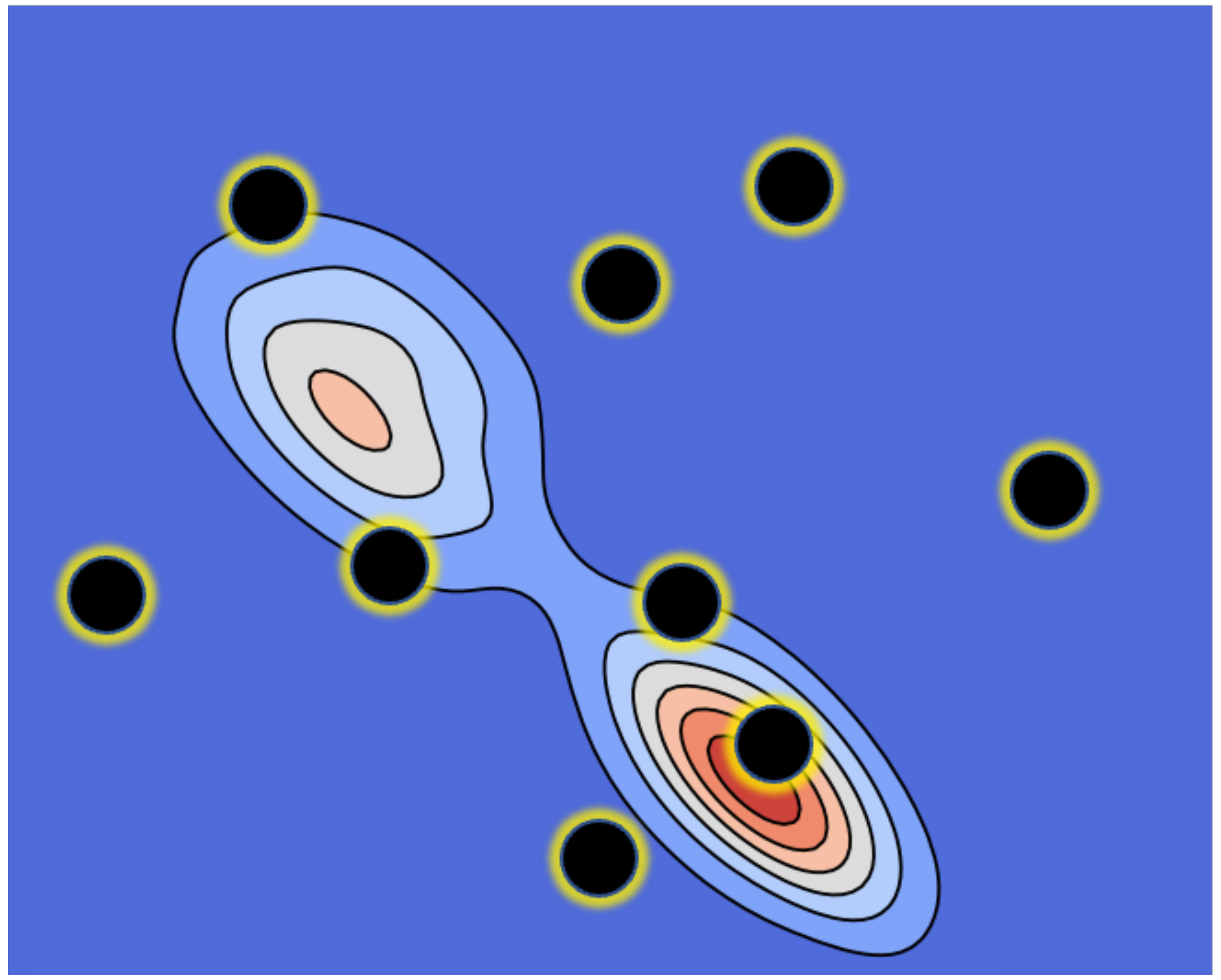
Optimal hyperparameter combination
(never sampled)

Grid search



Optimal hyperparameter combination
(never sampled)

Randomized search



Sampling hyperparameters

```
from sklearn.model_selection import ParameterSampler
from scipy.stats import loguniform


distributions = {
    "learning_rate": loguniform(0.001, 0.1),
    "activation": ['relu', 'swish', 'gelu']
}

sampler = ParameterSampler(distributions, n_iter=10, random_state=123)
```

Sampling hyperparameters

```
for param in sampler:
    print("my_script.py", end="")

    for k in param:
        print(f' --{k} {param [k]}', end="")

    print()
```

```
my_script.py --activation gelu --learning_rate 0.02666309997212923
my_script.py --activation gelu --learning_rate 0.0028425159292916
my_script.py --activation gelu --learning_rate 0.027434725570656345
my_script.py --activation gelu --learning_rate 0.007017992831138442
my_script.py --activation gelu --learning_rate 0.0066351194450833505
my_script.py --activation swish --learning_rate 0.009159332036121721
my_script.py --activation swish --learning_rate 0.006339209625904185
my_script.py --activation relu --learning_rate 0.028714378103928375
my_script.py --activation gelu --learning_rate 0.0013163027639428407
my_script.py --activation relu --learning_rate 0.015410076665458067
```

Randomized search

sklearn.model_selection.RandomizedSearchCV

class `sklearn.model_selection.`**`RandomizedSearchCV`**(*estimator*, *param_distributions*, ***, *n_iter=10*, *scoring=None*, *n_jobs=None*, *iid='deprecated'*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *random_state=None*, *error_score=nan*, *return_train_score=False*) [source]

n_iter : *int, default=10*

Number of parameter settings that are sampled. n_iter trades off runtime vs quality of the solution.

Randomized search

```
rcv = RandomizedSearchCV(
    pipe,
    param_distributions,
    n_iter=50,
    cv=5,
    random_state=123,
    verbose=1
)
```

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

Randomized search

```
rcv = RandomizedSearchCV(
    pipe,
    param_distributions,
    n_iter=50,
    cv=5,
    random_state=123,
    verbose=1
)
```

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

```
# Inherits the same methods we used previously:

rcv.fit(X_train, y_train)
rcv.score(X_test, y_test)
rcv.predict(X_test, y_test)
```

Randomized search

```python
rcv = RandomizedSearchCV(
    pipe,
    param_distributions,
    n_iter=50,
    cv=5,
    random_state=123,
    verbose=1
)
```

```python
rcv.fit(X_train, y_train)
```

```python
# Inherits the same methods we used previously:

rcv.fit(X_train, y_train)
rcv.score(X_test, y_test)
rcv.predict(X_test, y_test)
```

Randomized search

```
rcv = RandomizedSearchCV(
    pipe,
    param_distributions,
    n_iter=50,
    cv=5,
    random_state=123,
    verbose=1
)
```

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

```
# Inherits the same methods we used previously:

rcv.fit(X_train, y_train)
rcv.score(X_test, y_test)
rcv.predict(X_test, y_test)
```

Exercise:

Training a small neural network

🎮 https://github.com/rasbt/posit2023-python-ml