

# 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)

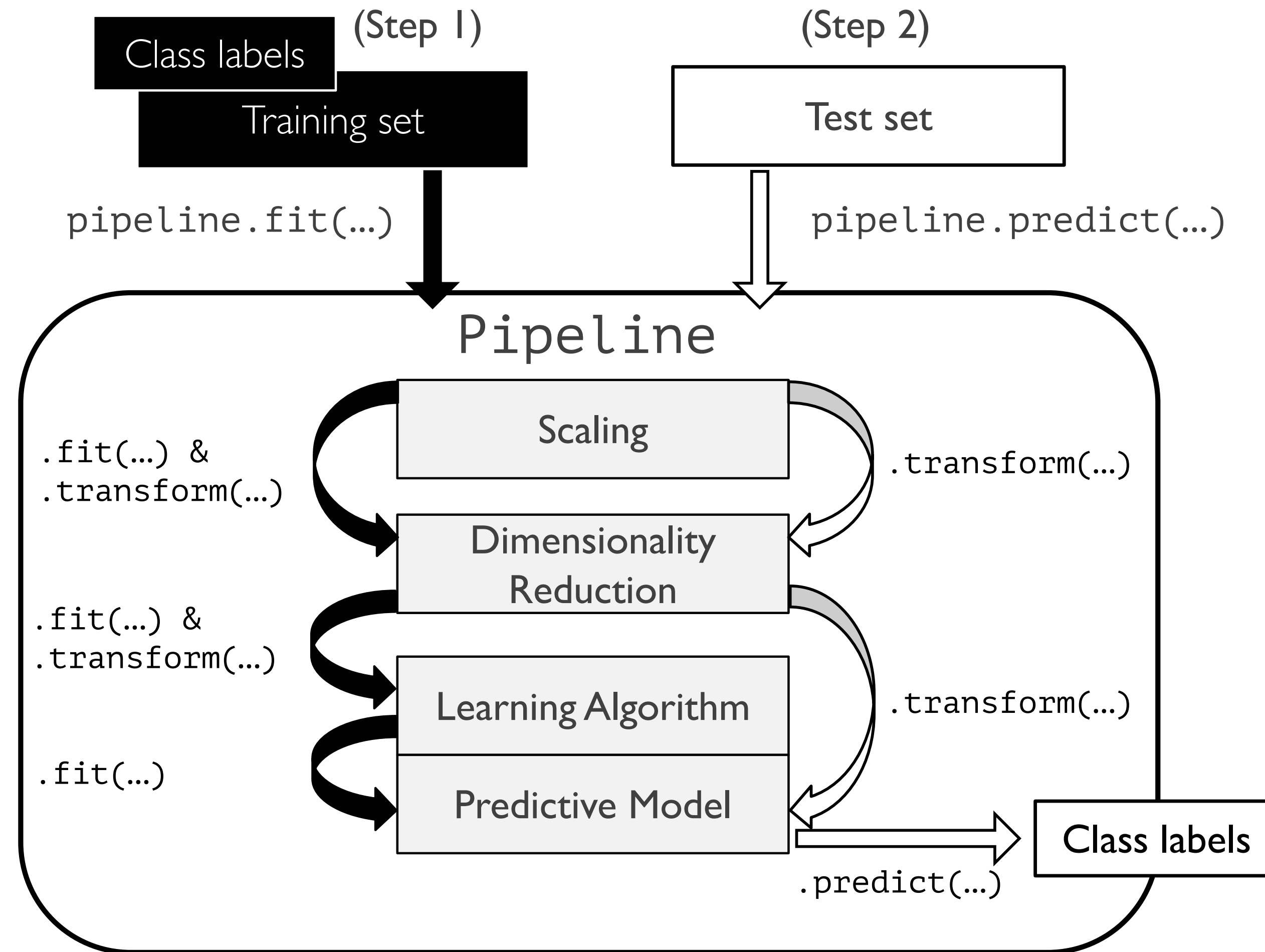
...

## 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

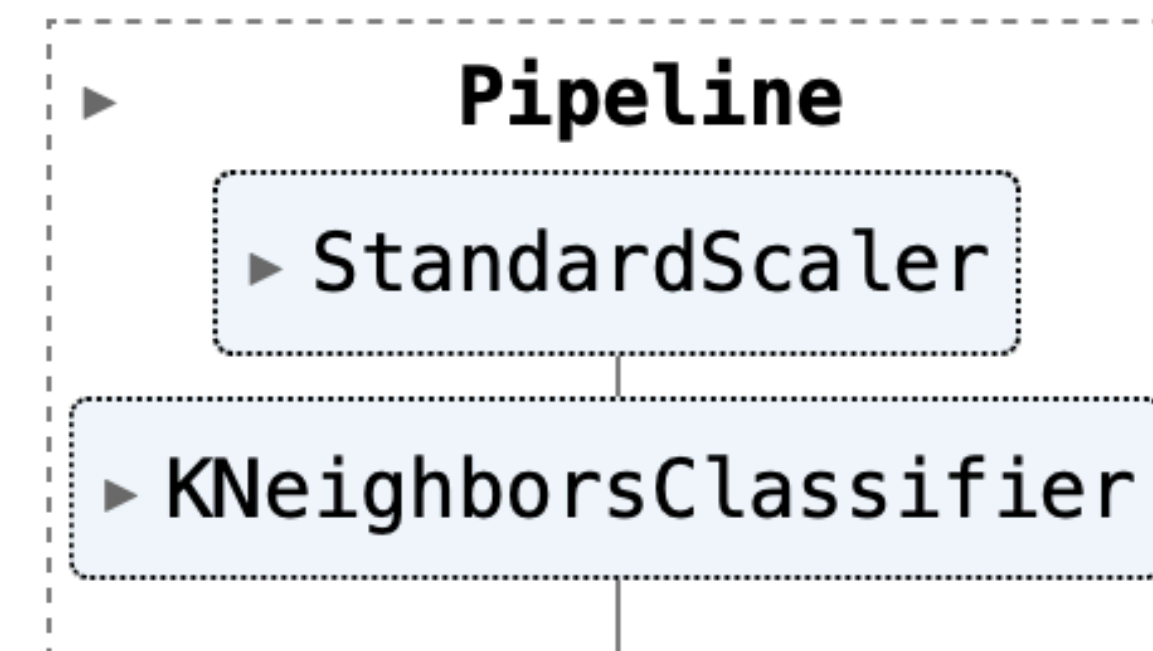
pipe = make_pipeline(
    StandardScaler(),
    KNeighborsClassifier(n_neighbors=3)
)
```

# Scikit-learn pipelines

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

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

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



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

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

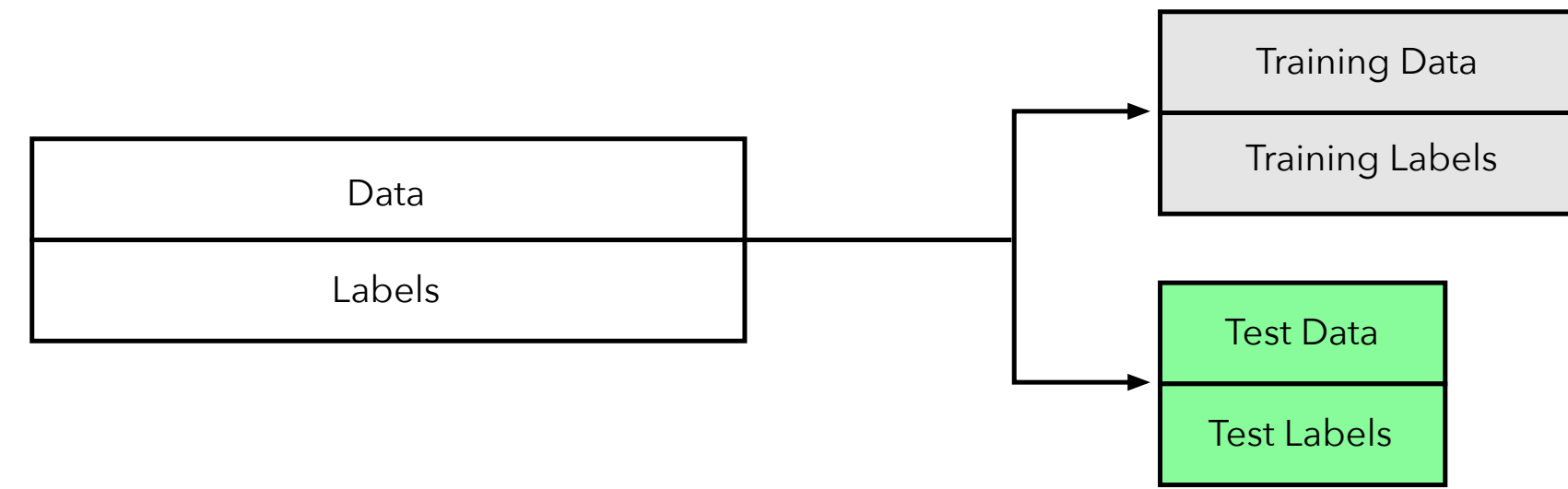
```
pipe.score(X_test, y_test)
```

```
0.92
```

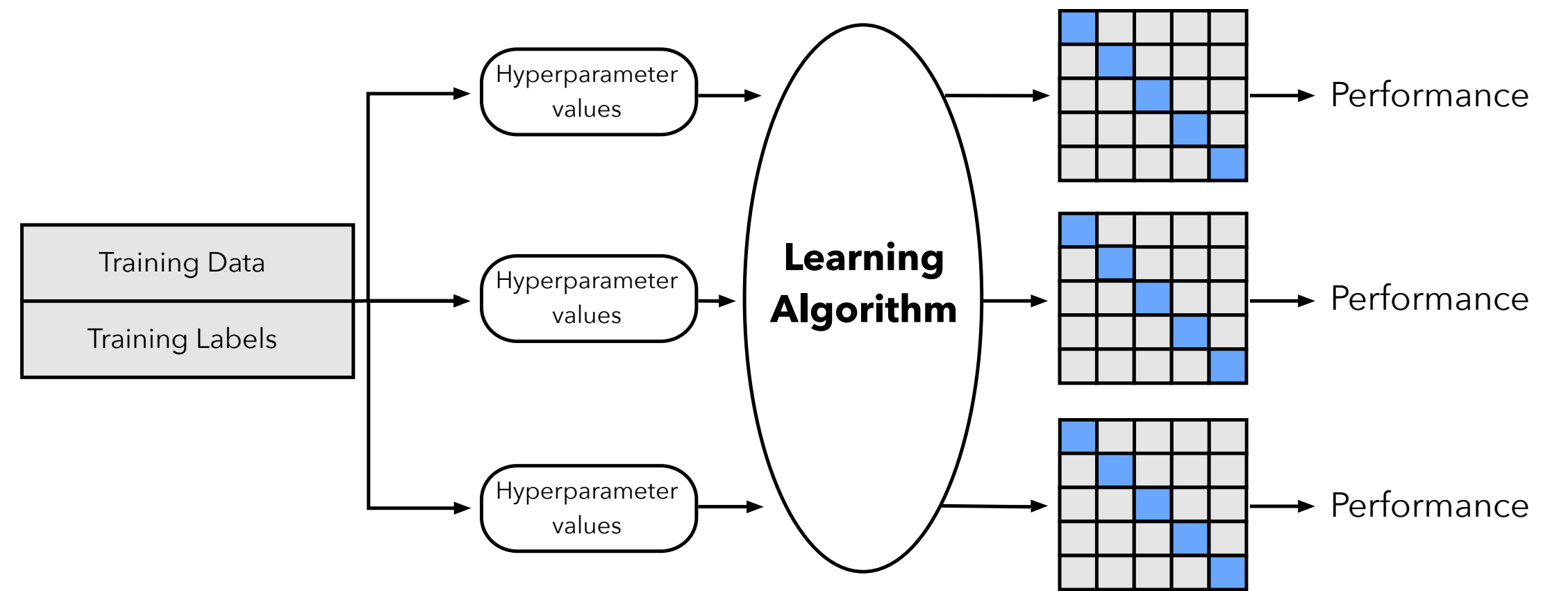
# Model evaluation

# Model selection with k-fold cross-validation

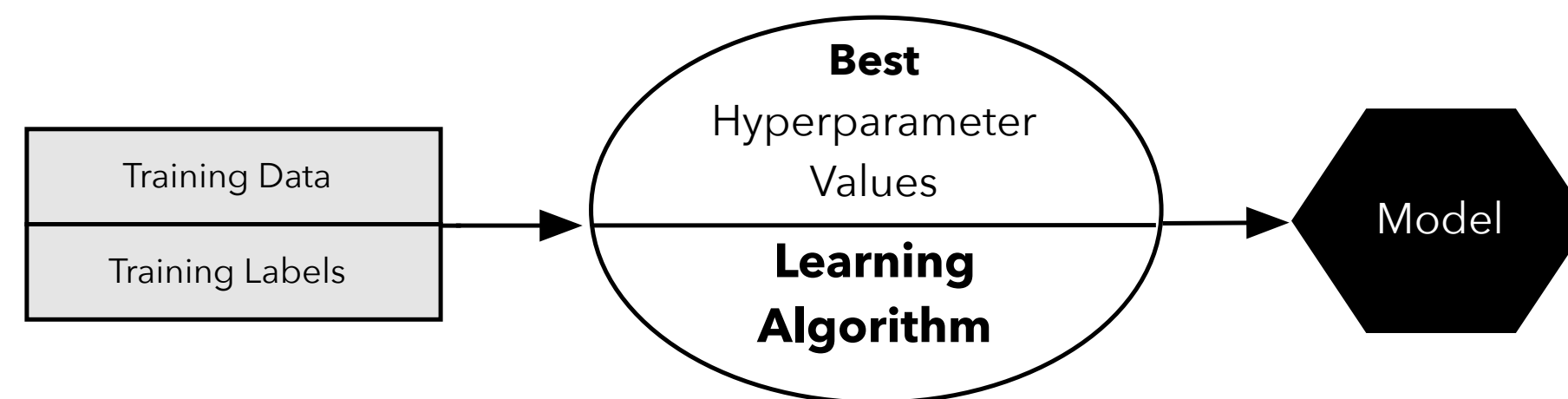
1



2



3



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

Sebastian Raschka

<https://arxiv.org/abs/1811.12808>

## Contents

<b>1</b>	<b>Introduction: Essential Model Evaluation Terms and Techniques</b>	<b>4</b>
1.1	Performance Estimation: Generalization Performance vs. Model Selection . . . . .	4
1.2	Assumptions and Terminology . . . . .	5
1.3	Resubstitution Validation and the Holdout Method . . . . .	7
1.4	Stratification . . . . .	7
1.5	Holdout Validation . . . . .	8
1.6	Pessimistic Bias . . . . .	10
1.7	Confidence Intervals via Normal Approximation . . . . .	10
<b>2</b>	<b>Bootstrapping and Uncertainties</b>	<b>11</b>
2.1	Overview . . . . .	11
2.2	Resampling . . . . .	12
2.3	Repeated Holdout Validation . . . . .	15
2.4	The Bootstrap Method and Empirical Confidence Intervals . . . . .	15
<b>3</b>	<b>Cross-validation and Hyperparameter Optimization</b>	<b>20</b>
3.1	Overview . . . . .	20
3.2	About Hyperparameters and Model Selection . . . . .	21
3.3	The Three-Way Holdout Method for Hyperparameter Tuning . . . . .	22
3.4	Introduction to $k$ -fold Cross-Validation . . . . .	24
3.5	Special Cases: 2-Fold and Leave-One-Out Cross-Validation . . . . .	26
3.6	$k$ -fold Cross-Validation and the Bias-Variance Trade-off . . . . .	28
3.7	Model Selection via $k$ -fold Cross-Validation . . . . .	30
3.8	A Note About Model Selection and Large Datasets . . . . .	30
3.9	A Note About Feature Selection During Model Selection . . . . .	30
3.10	The Law of Parsimony . . . . .	32
3.11	Summary . . . . .	33
<b>4</b>	<b>Algorithm Comparison</b>	<b>34</b>
4.1	Overview . . . . .	34
4.2	Testing the Difference of Proportions . . . . .	34
4.3	Comparing Two Models with the McNemar Test . . . . .	35
4.4	Exact $p$ -Values via the Binomial Test . . . . .	37
4.5	Multiple Hypotheses Testing . . . . .	38
4.6	Cochran's $Q$ Test for Comparing the Performance of Multiple Classifiers . . . . .	39
4.7	The $F$ -test for Comparing Multiple Classifiers . . . . .	41
4.8	Comparing Algorithms . . . . .	42
4.9	Resampled Paired $t$ -Test . . . . .	43
4.10	$k$ -fold Cross-validated Paired $t$ -Test . . . . .	44



# 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
```

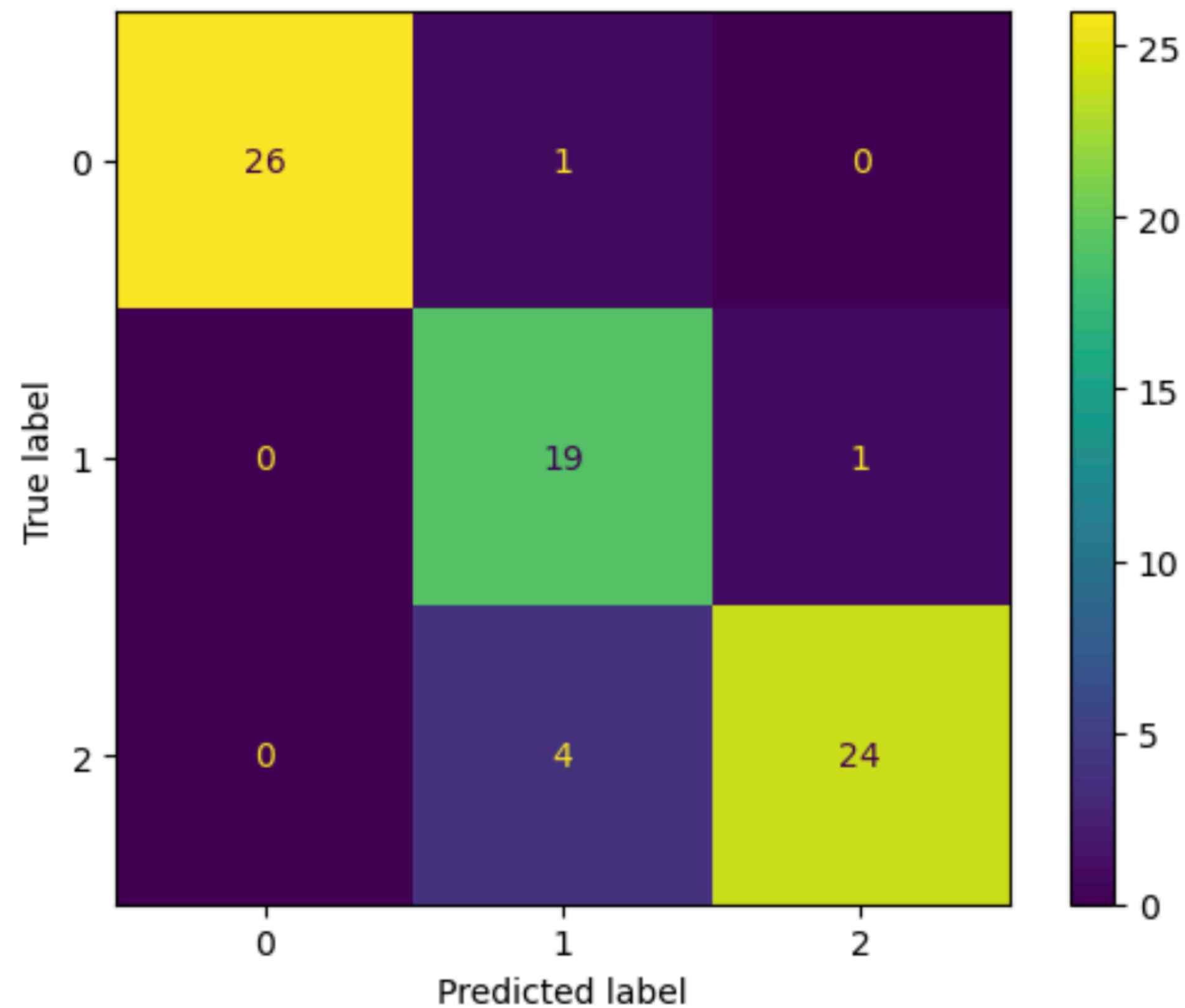
# Confusion matrices to look at failure cases

```
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

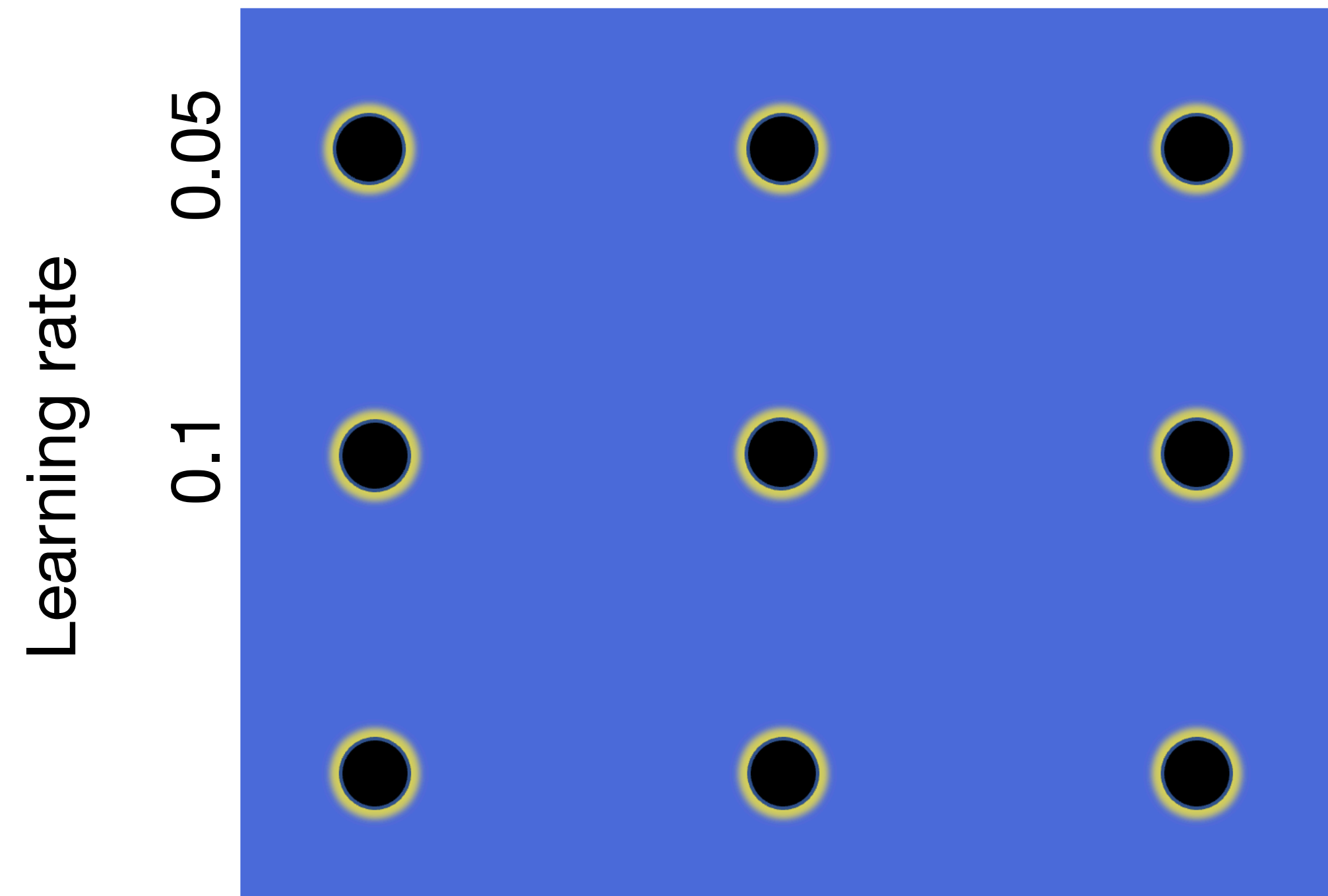
Note: Often, changing e.g., the batch size also requires tuning other parameters from scratch again (like the learning rate)

# Grid search: a brute-force search

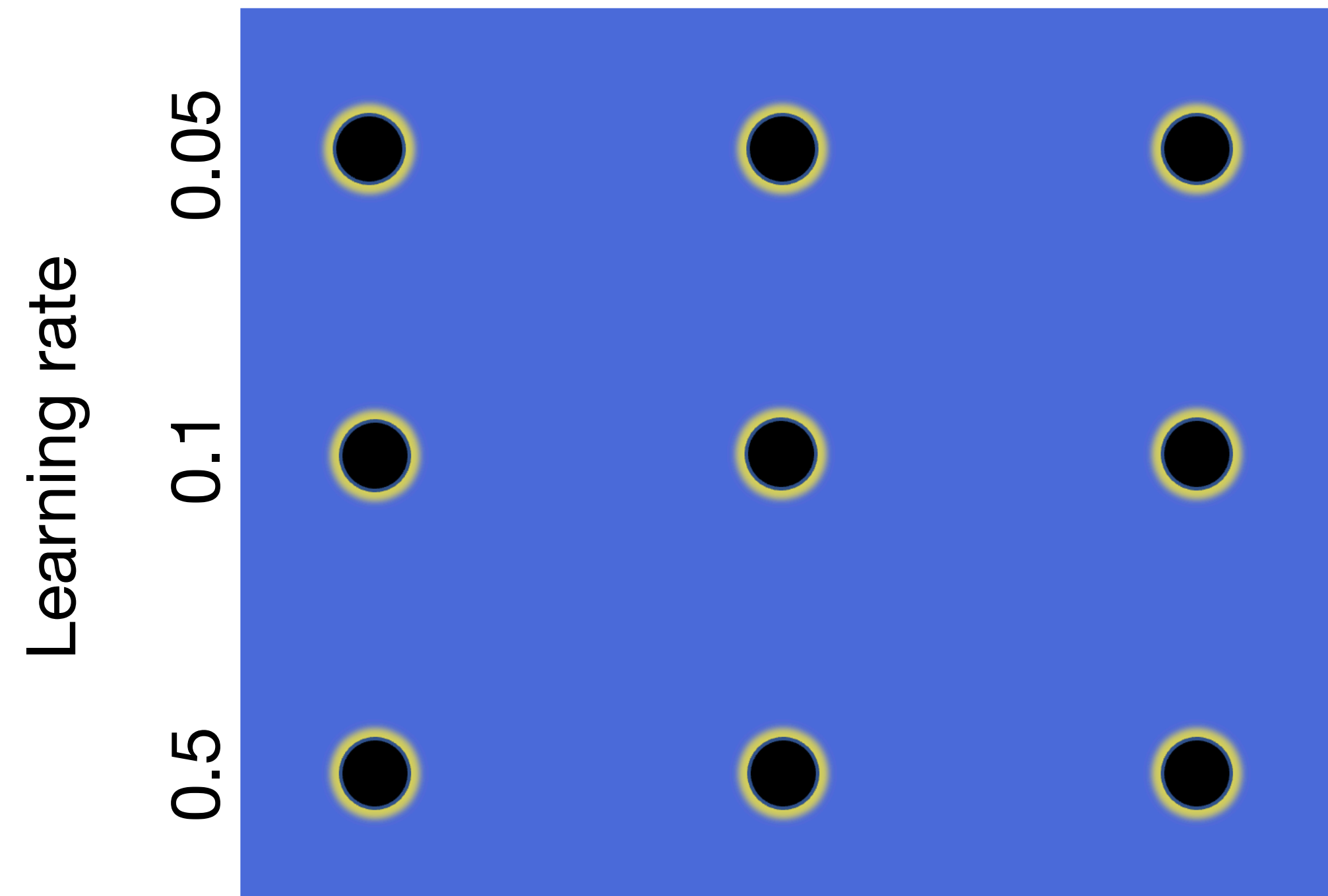
Suppose we only consider the following two hyperparameters:

- 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



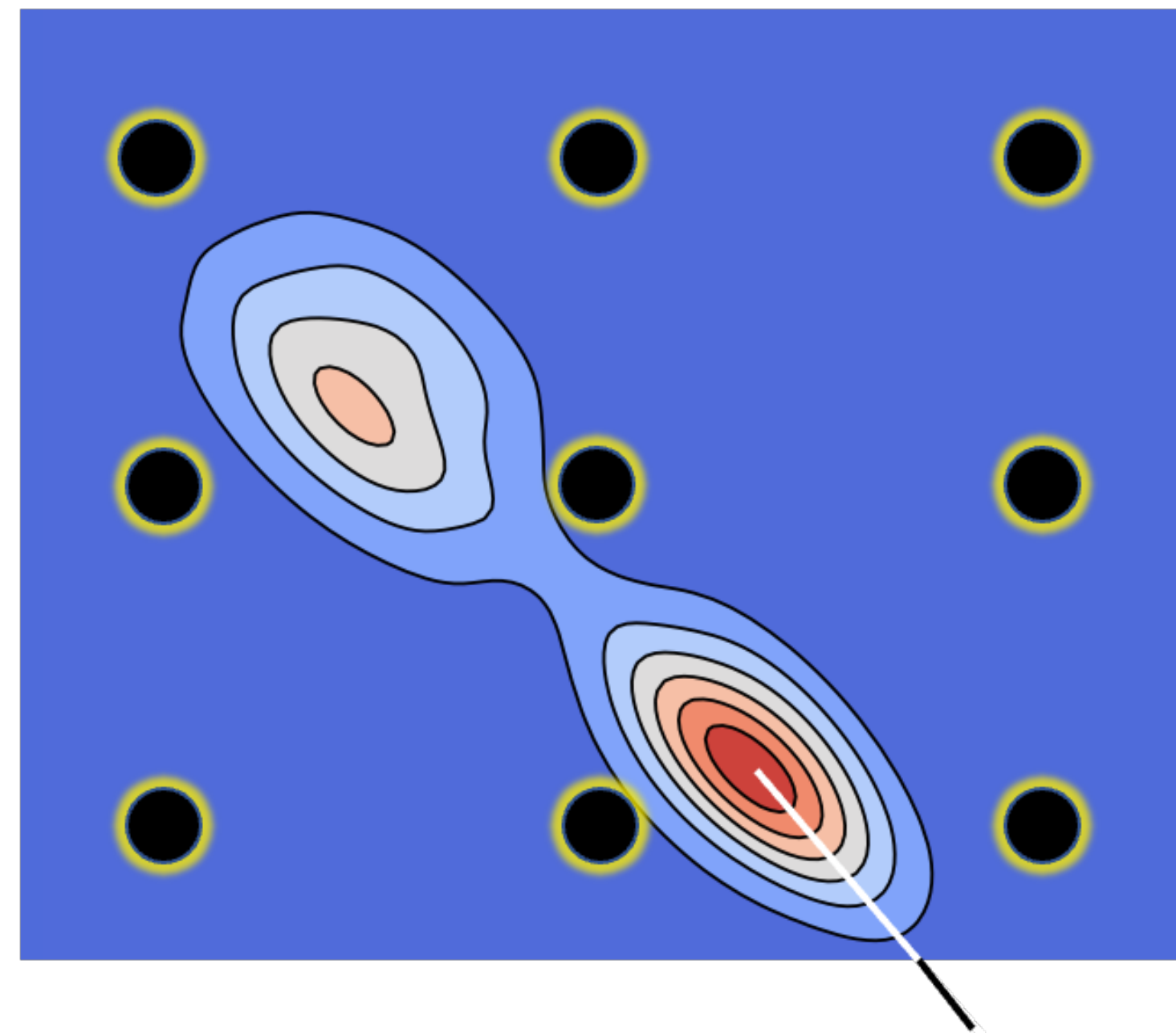


# Randomized search

Draw parameters from distributions

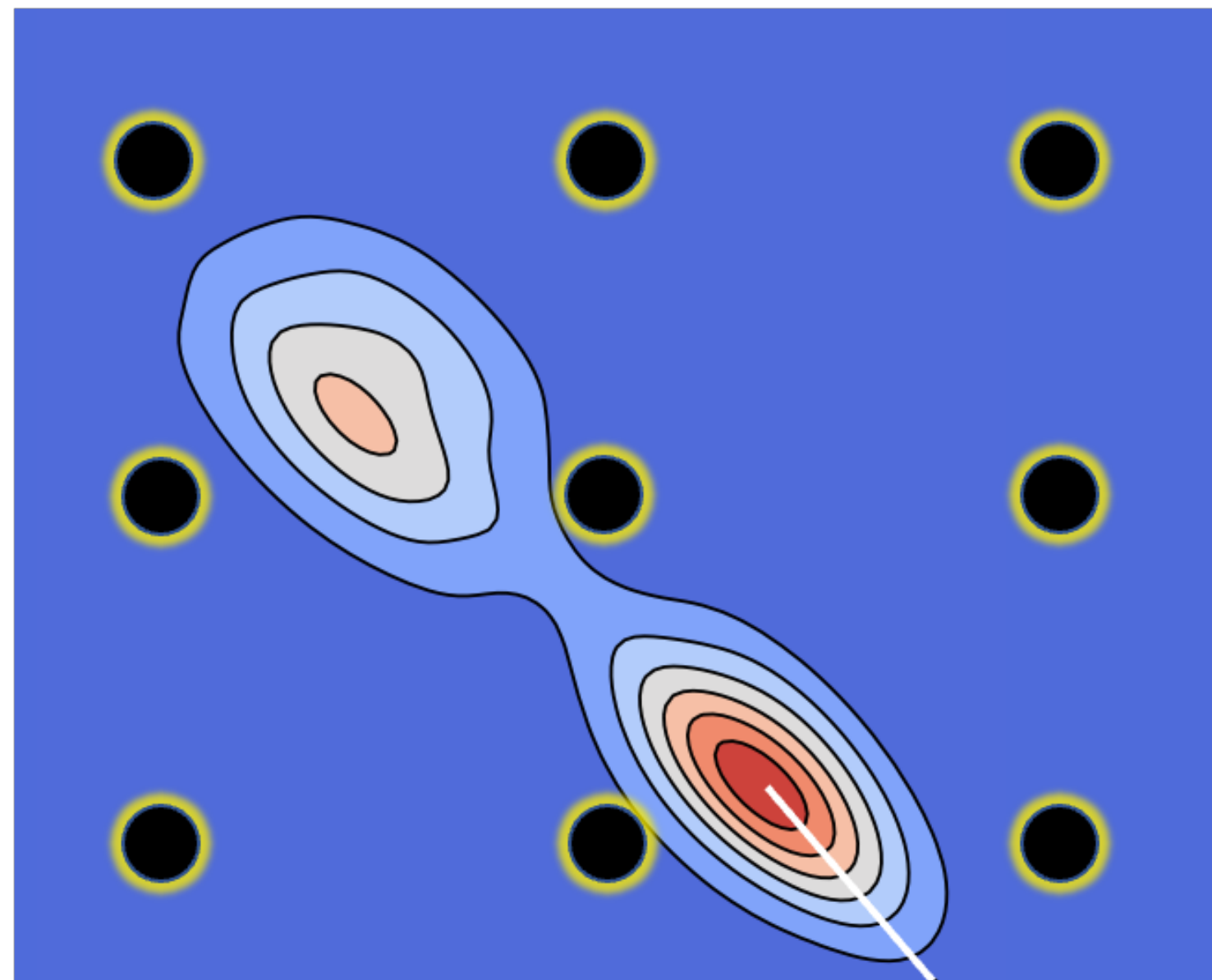
Explore a wider range of parameter settings

# Grid search



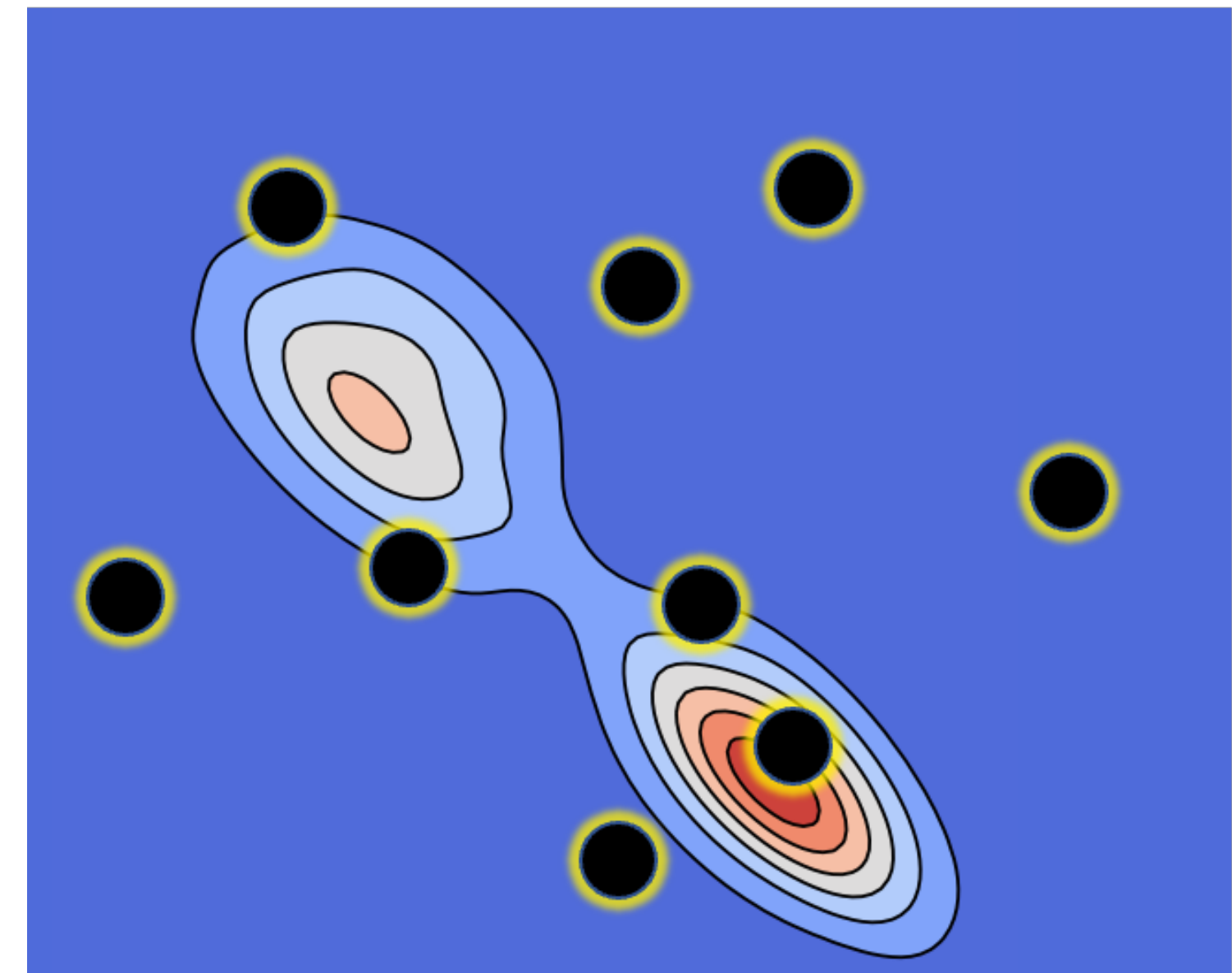
Optimal hyperparameter combination  
(never sampled)

## 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 0.00284251592929916
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

```
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

```
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>