18 - Advanced Splitting

Data Science Practicum 2021/22, Lesson 18

Marko Tkalčič

Univerza na Primorskem

Table of Contents

Train-test-validation splitting

- Predictor/regressors are trained
  - using the train set
  - the weights/parameters are the outcome of the training process
- Most of the classifier/regressors have **hyperparameters**
  - Parameters whose values are used to control the learning process
  - Are set before the training
  - The prediction quality depends on these parameters
  - Examples:
    - max depth of a random forest (model hyperparameter)
    - learning rate (algorithm hyperparameter)
    - topology and size of a neural network

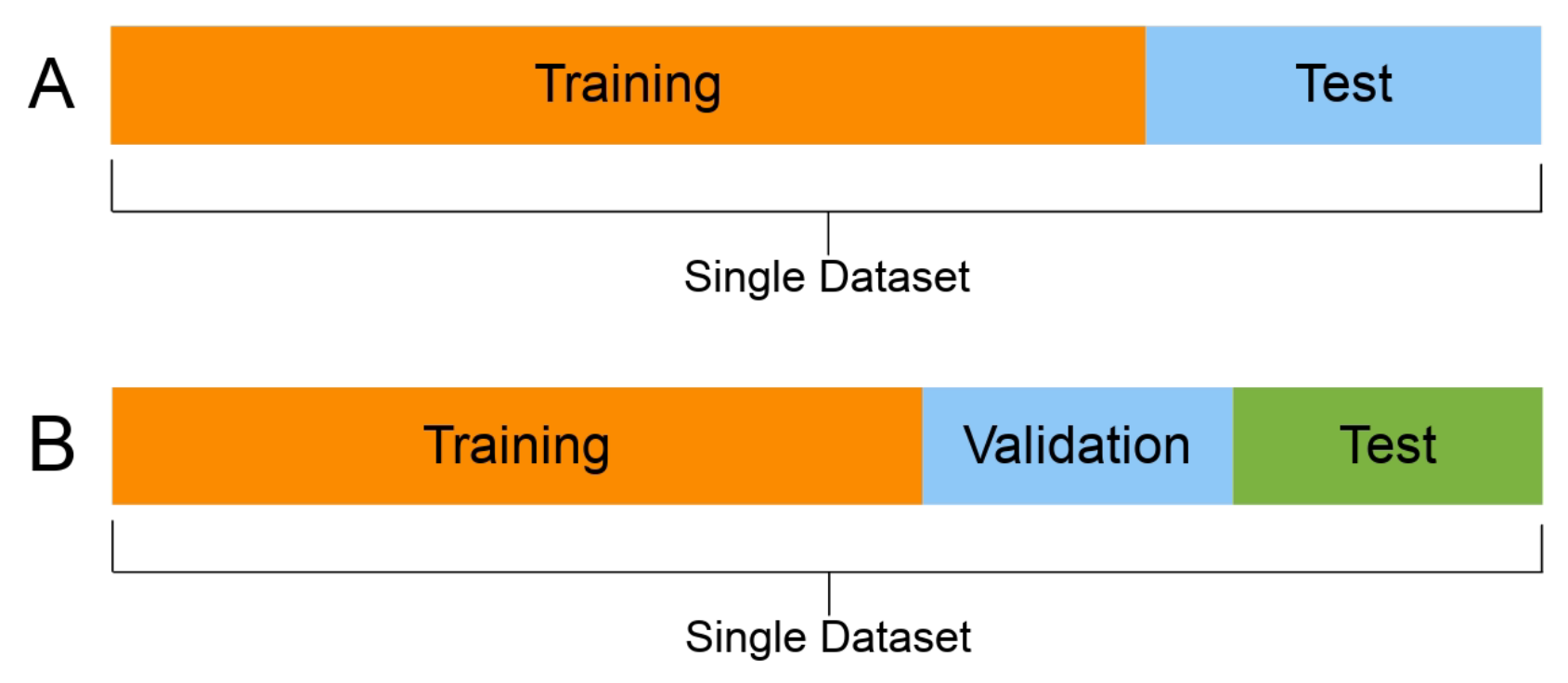
Train-test-validation splitting

- Predictor/regressors are trained
  - using the train set
  - the weights/parameters are the outcome of the training process
- Most of the classifier/regressors have **hyperparameters**
  - Parameters whose values are used to control the learning process
  - Are set before the training
  - The prediction quality depends on these parameters
  - Examples:
    - max depth of a random forest (model hyperparameter)
    - learning rate (algorithm hyperparameter)
    - topology and size of a neural network
- We need to find the optimal hyperparameters
- How?

Train-test-validation splitting

- Predictor/regressors are trained
  - using the train set
  - the weights/parameters are the outcome of the training process
- Most of the classifier/regressors have **hyperparameters**
  - Parameters whose values are used to control the learning process
  - Are set before the training
  - The prediction quality depends on these parameters
  - Examples:
    - max depth of a random forest (model hyperparameter)
    - learning rate (algorithm hyperparameter)
    - topology and size of a neural network
- We need to find the optimal hyperparameters
- How?
- We introduce an additional data subset only for validating the hyperparameters: the validation set

Validation set




```
X_train, X_test, y_train, y_test
    = train_test_split(X, y, test_size=0.2, random_state=1)

X_train, X_val, y_train, y_val
    = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
```

```
train, validate, test = \
            np.split(df.sample(frac=1, random_state=42),
                     [int(.6*len(df)), int(.8*len(df))])
```

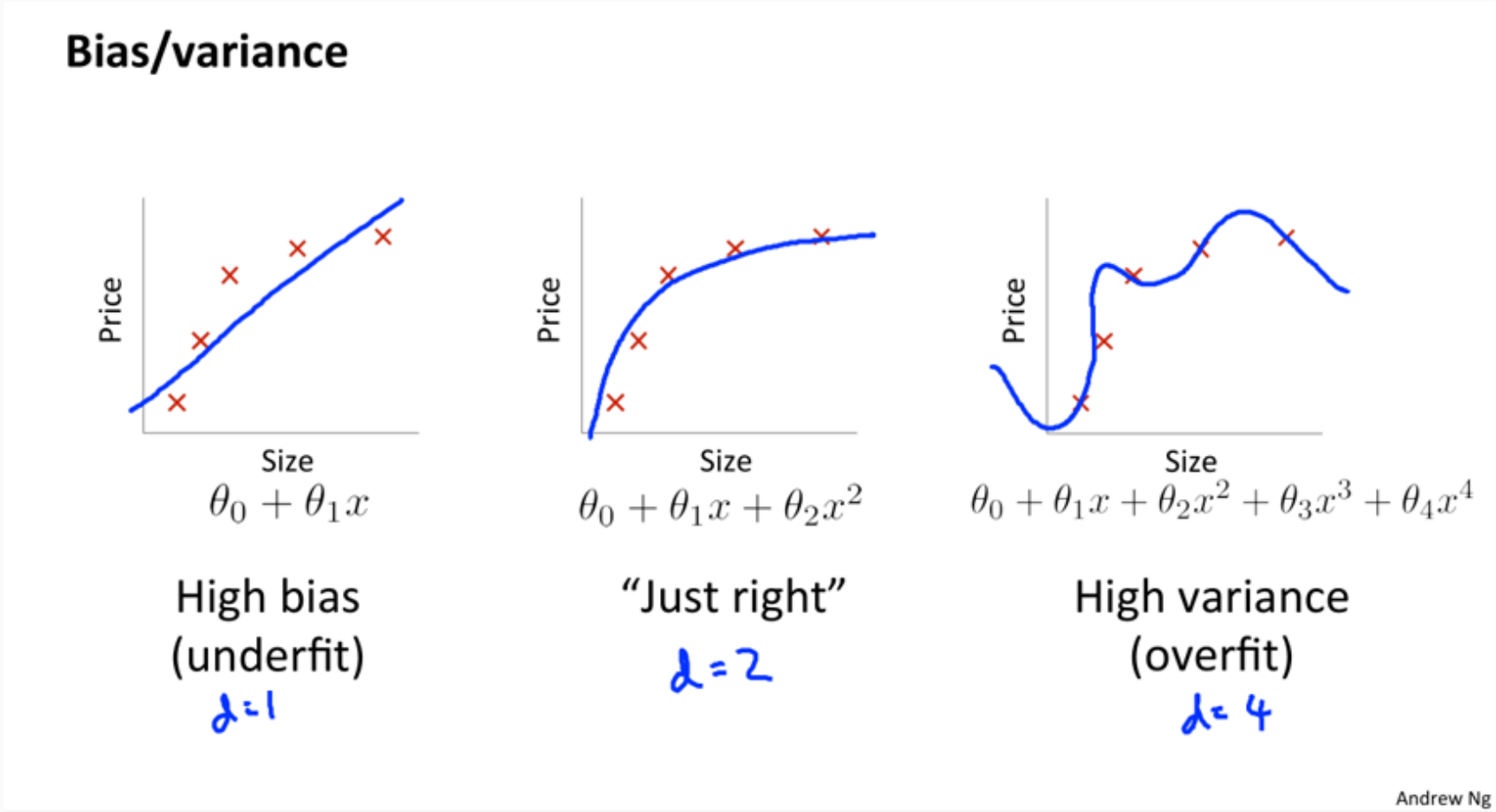
Table of Contents

Hyperparameters

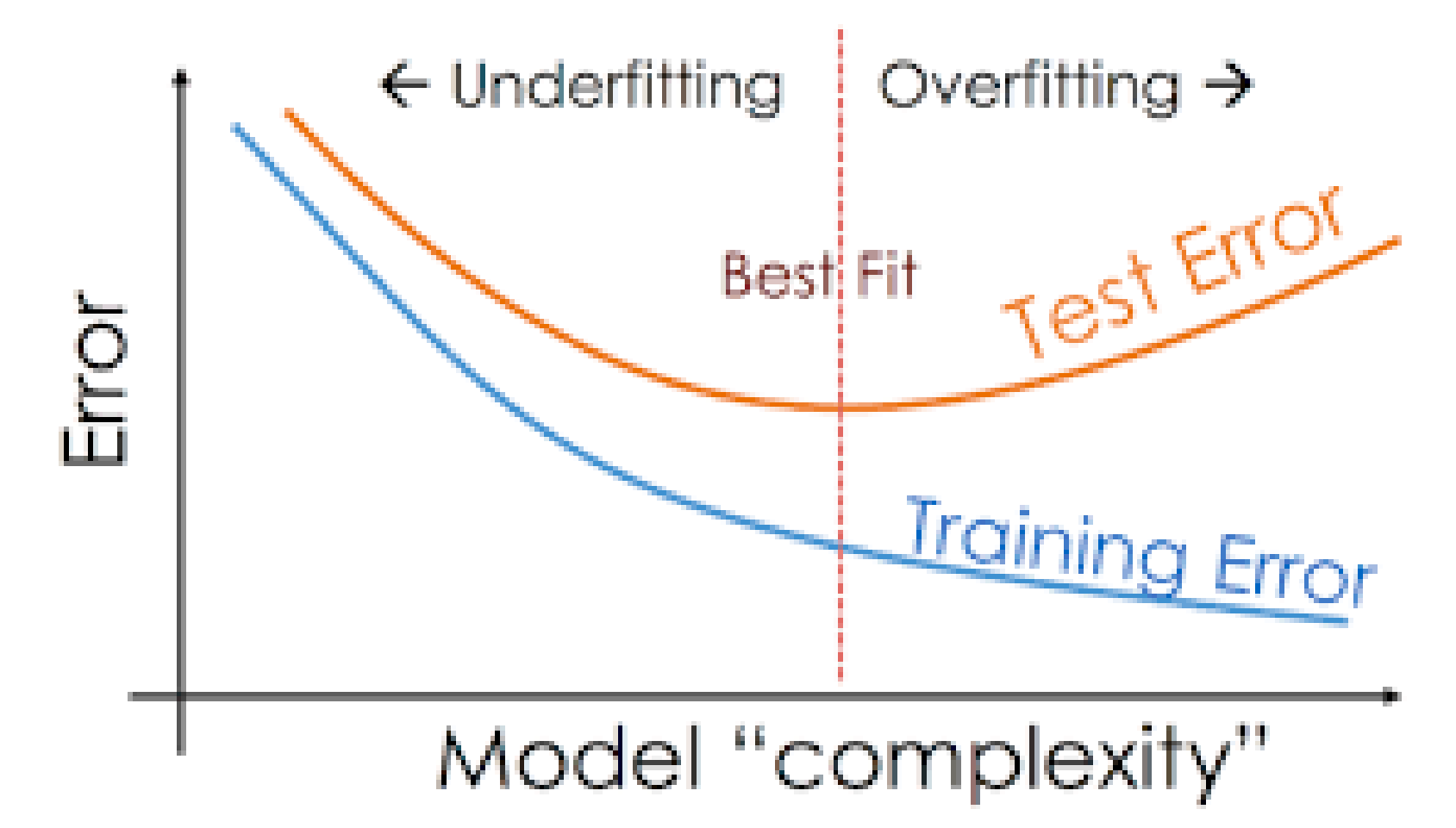
- The hyperparameter search often boils down to:
  - *How Complex should my model be?*
    - polynomial degree
    - network depth
    - forest depth

Hyperparameters

- The hyperparameter search often boils down to:
  - *How Complex should my model be?*
    - polynomial degree
    - network depth
    - forest depth



Bias-variance



Exercise

- Make classification dataset 1000 samples, 10 fetures
- Split into train, test, and validation 80:10:10
- RandomForestClassifier
- Find the best value for the max_depth hyperparameter (iterate over it and plot the recall_score)

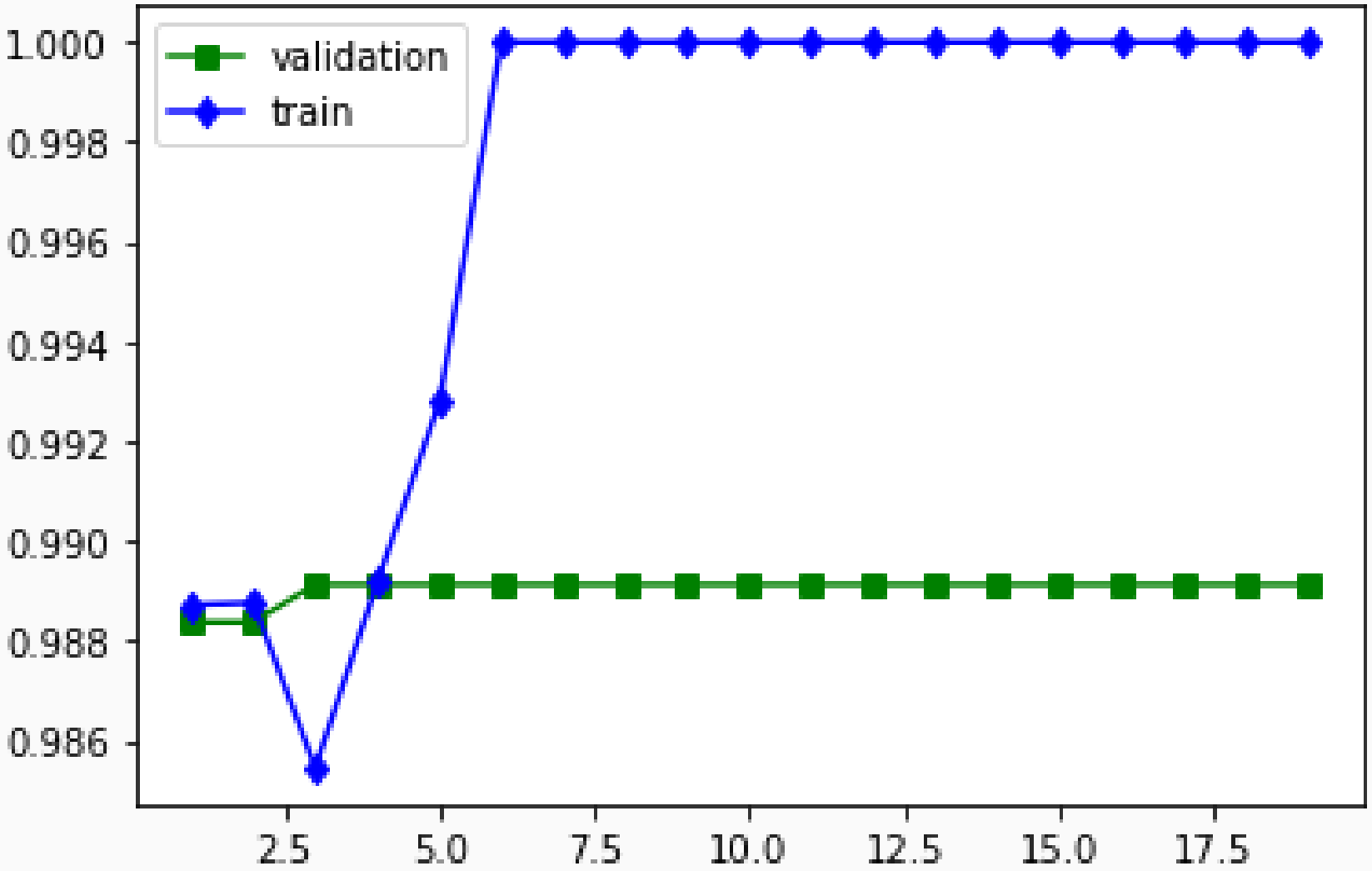
Exercise

- Make classification dataset 1000 samples, 10 fetures
- Split into train, test, and validation 80:10:10
- RandomForestClassifier
- Find the best value for the max_depth hyperparameter (iterate over it and plot the recall_score)

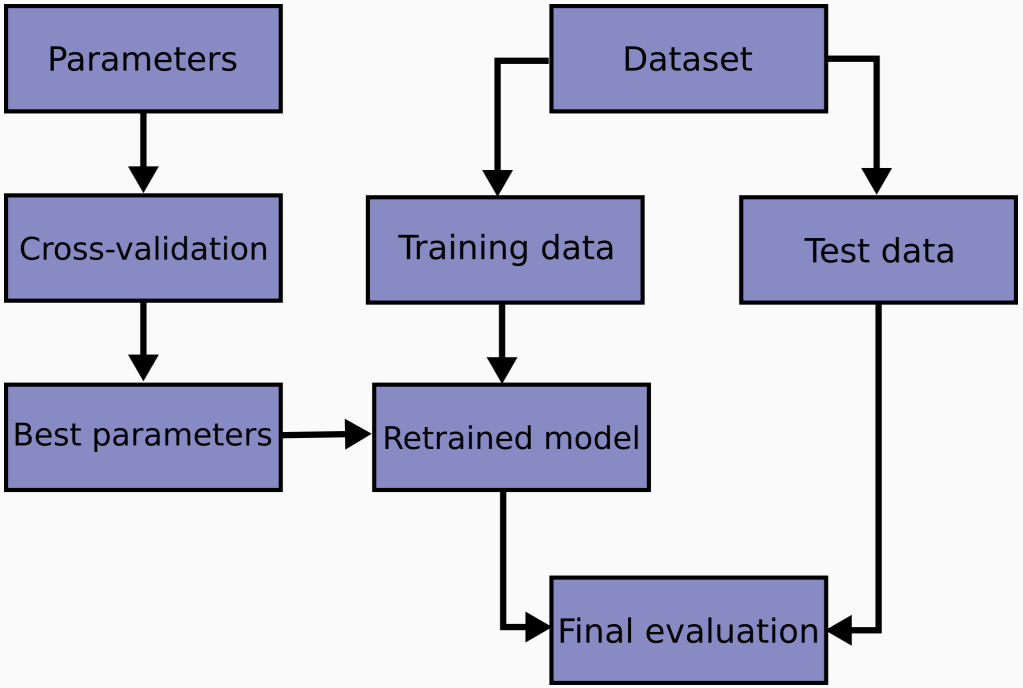
```
from sklearn.metrics import precision_score, recall_score
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

X, y = make_classification(n_samples=1000, n_features=10, random_state=7)
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state=1)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=1)
mse_valid = []
mse_train = []
tss = range(1,20)
for n in tss:
    model =  RandomForestClassifier(max_depth=n, random_state=2)
    model.fit(X_train,y_train)
    mse_valid.append(recall_score(model.predict(X_val),y_val))
    mse_train.append(recall_score(model.predict(X_train),y_train))
plt.plot(tss,mse_valid,"-sg", label="validation")
plt.plot(tss,mse_train,"-db", label="train")
plt.legend()
plt.show()
```

Exercise


validation
train
1.000
0.998
0.996
0.994
0.992
0.990
0.988
0.986
2.5
5.0
7.5
10.0
12.5
15.0
17.5

Grid Search



- Exhaustive search over a set of hyperparameters

```
param_grid = [
  {'C': [1, 10, 100, 1000], 'kernel': ['linear']},
  {'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']},
 ]
```

Grid Search

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC

digits = datasets.load_digits()
n_samples = len(digits.images)
X = digits.images.reshape((n_samples, -1))
y = digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]
scores = ['precision', 'recall']
for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    clf = GridSearchCV(
        SVC(), tuned_parameters, scoring='%s_macro' % score
    )
    clf.fit(X_train, y_train)
    print("Best parameters set found on development set:")
    print(clf.best_params_)
```

```
# Tuning hyper-parameters for precision
Best parameters set found on development set:
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
# Tuning hyper-parameters for recall
Best parameters set found on development set:
{'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

Exercise

- Make a classification dataset with 100 samples and 5 features
- RandomForestClassifier
- Use GridSearchCV to find the best hyperparameters in the set:
- n_estimators = [50, 100, 200]
- max_depth = [10, 20, 30]

Exercise

- Make a classification dataset with 100 samples and 5 features
- RandomForestClassifier
- Use GridSearchCV to find the best hyperparameters in the set:
- n_estimators = [50, 100, 200]
- max_depth = [10, 20, 30]

```
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import sklearn

X, y = make_classification(n_samples=100, n_features=5, random_state=7)
#print(sklearn.metrics.SCORERS.keys())
n_estimators = [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
m_depth = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]
n_estimators = [50, 100, 200, 400]
m_depth = [10, 20, 30]
params = [{"n_estimators": n_estimators, "max_depth" : m_depth}]
clf = GridSearchCV(
    RandomForestClassifier(), params, scoring="precision"
)
clf.fit(X,y)
print(clf.best_params_)
```

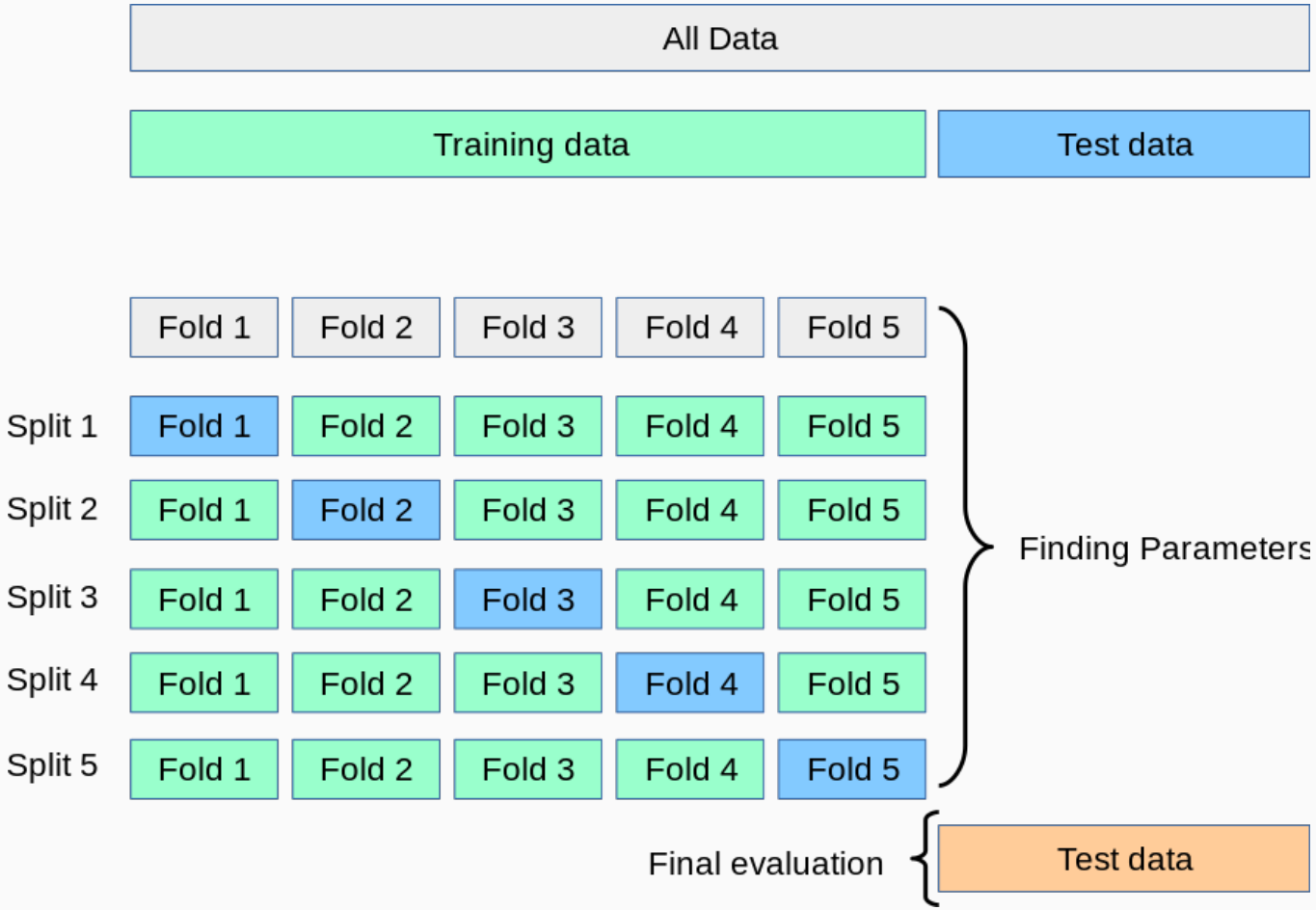
```
{'max_depth': 10, 'n_estimators': 100}
```

Validation and Learning Curves

https://scikit-learn.org/stable/modules/learning_curve.html

Table of Contents

Cross validation



- Can be used for hyperparameter tuning or for testing only

Cross validation in Python

- `cross_val_score`
- `cross_validate` (allows multiple metrics, returns more info)
- `cross_val_predict` (returns also the predicted values in the test sets)

Cross validation in Python

- `cross_val_score`
- `cross_validate` (allows multiple metrics, returns more info)
- `cross_val_predict` (returns also the predicted values in the test sets)

```
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

X, y = make_classification(n_samples=1000, n_features=10, random_state=7)
clf = SVC(kernel='linear', C=1)
scores = cross_val_score(clf, X, y, cv=5)
scores
```

```
array([0.92 , 0.92 , 0.935, 0.94 , 0.945])
```

Cross validation Iterators

- Cross validation iterators in sklearn
  - utilities to generate indices that can be used to generate dataset splits according to different cross validation strategies.
  - K-fold
  - Repeated K-fold
  - Leave One Out
  - Stratified k-fold
  - ...

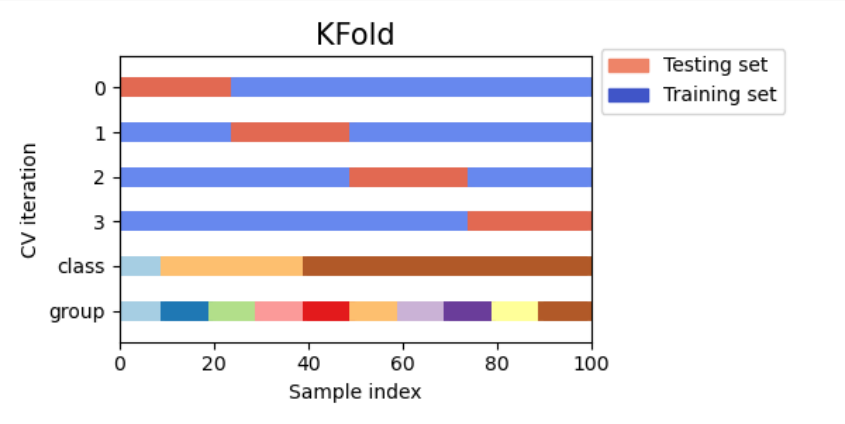
K Fold

- divides all the samples in k groups of samples, called folds (if $k = n$, this is equivalent to the Leave One Out strategy), of equal sizes (if possible). The prediction function is learned using $k - 1$ folds, and the fold left out is used for test.

```
import numpy as np
from sklearn.model_selection import KFold

X = np.array(["a", "b", "c", "d", "e", "f", "g", "h"])
kf = KFold(n_splits=4)
for train, test in kf.split(X):
    X[train]
    print("%s %s" % (train, test))
```

```
[2 3 4 5 6 7] [0 1]
[0 1 4 5 6 7] [2 3]
[0 1 2 3 6 7] [4 5]
[0 1 2 3 4 5] [6 7]
```

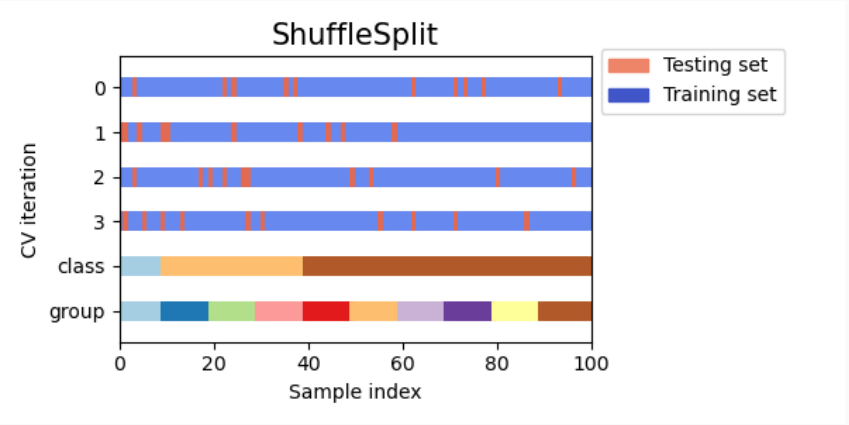


Shuffle and Split

```
import numpy as np
from sklearn.model_selection import ShuffleSplit

X = np.array(["a", "b", "c", "d", "e", "f", "g", "h"])
kf = ShuffleSplit(n_splits=4)
for train, test in kf.split(X):
    X[train]
    print("%s %s" % (train, test))
```

```
[2 5 6 3 0 7 4] [1]
[6 5 3 4 2 1 7] [0]
[1 3 4 0 6 5 2] [7]
[2 1 3 4 6 5 7] [0]
```



Stratified k-fold

- class imbalance
- relative class frequencies is approximately preserved in each train and validation fold

```
# Stratified k fold
from sklearn.model_selection import StratifiedKFold, KFold
import numpy as np
X, y = np.ones((50, 1)), np.hstack(([0] * 45, [1] * 5))

skf = StratifiedKFold(n_splits=3)
for train, test in skf.split(X, y):
    print('Stratified: train - {}   |   test - {}'.format(
        np.bincount(y[train]), np.bincount(y[test])))

kf = KFold(n_splits=3)
for train, test in kf.split(X, y):
    print('KFold: train - {}   |   test - {}'.format(
        np.bincount(y[train]), np.bincount(y[test])))
```

```
Stratified: train - [30  3]   |   test - [15  2]
Stratified: train - [30  3]   |   test - [15  2]
Stratified: train - [30  4]   |   test - [15  1]
KFold: train - [28  5]   |   test - [17]
KFold: train - [28  5]   |   test - [17]
KFold: train - [34]   |   test - [11  5]
```

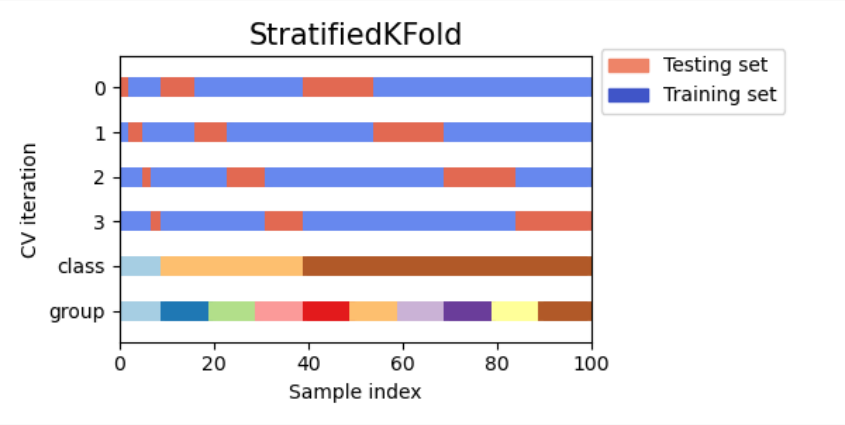
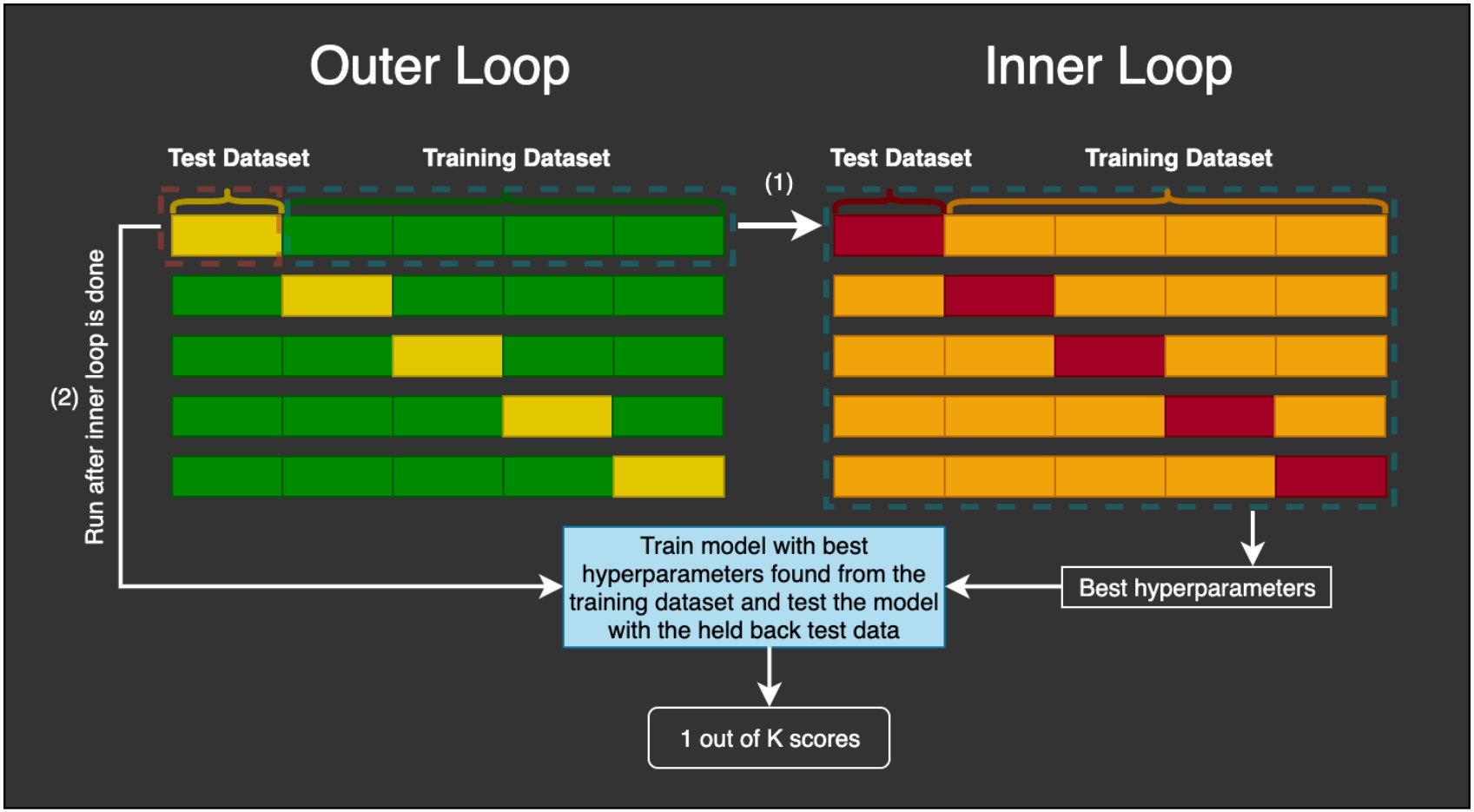


Table of Contents

Nested Cross Validation



- https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/
- https://mlfromscratch.com/nested-cross-validation-python-code/#/

Table of Contents

Assignment (optional)

- Load the Iris dataset
- Pick a classifier of your choice
- Pick three evaluation metrics of your choice. In the code comment provide a short rationale.
- Pick at least two hyper-parameters of the classifier.
- For each hyper-parameter choose at least 5 values according to your subjective assessment.
- Perform grid search using GridSearchCV for each of the three metrics
- Visualize the performance of each hyper-parameter pair using a 2D visualization (e.g. a heatmap)

Table of Contents

References

Part of the material has been taken from the following sources. The usage of the referenced copyrighted work is in line with fair use since it is for nonprofit educational purposes.

- https://scikit-learn.org/stable/modules/model_evaluation.html
- https://en.wikipedia.org/wiki/Precision_and_recall
- https://en.wikipedia.org/wiki/Hyperparameter_(machine_learning)
- https://stackoverflow.com/questions/38250710/how-to-split-data-into-3-sets-train-validation-and-test
- https://datascience.stackexchange.com/questions/15135/train-test-validation-set-splitting-in-sklearn
- https://algotrading101.com/learn/train-test-split/
- https://scipy-lectures.org/packages/scikit-learn/auto_examples/plot_bias_variance.html
- https://mlfromscratch.com/nested-cross-validation-python-code/#/