Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

2024 FS CAS PML - Supervised Learning
4 Validierung (und mehr)
4.3 Grid Search und Random Search

Werner Dähler 2024

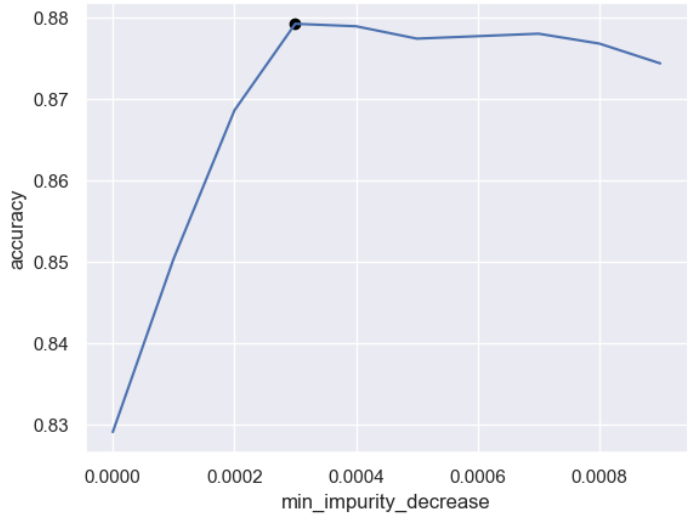
4 Validierung und mehr - AGENDA

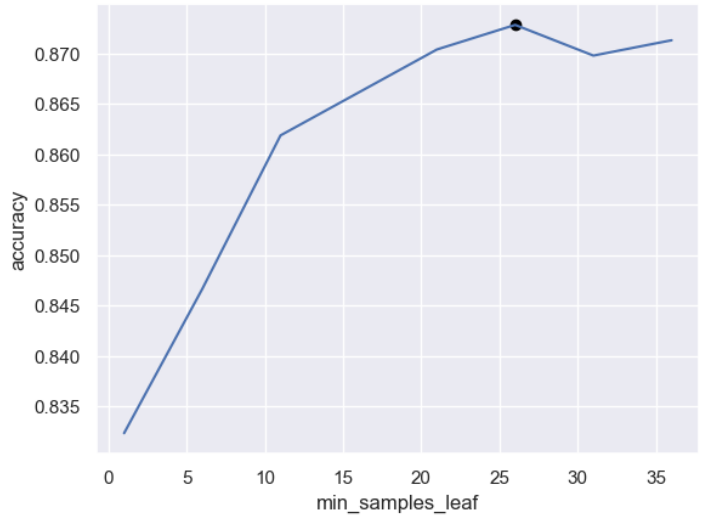
41. Sampling und Resampling
42. Validierungstechniken
43. Grid Search und Random Search
44. Performance Metriken
45. Unbalancierte Daten

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.1 Rekapitulation: Parameter Tuning

- schon mehrmals angewendet zum Parameter Tuning, d.h. mit einem Loop über verschiedene Werte eines Hyperparameters
- dieses Vorgehen kann auch gleich mit Kreuzvalidierung kombiniert werden
- im Kapitel 2.2.1.5 (z.B. bei Klassifikation - DecisionTreeClassifier) wurde Parameter Tuning mit einem selber codierten Loop durchgeführt, z.B. für die Hyperparamete
  - min_impurity_decrease im Bereich 0 bis 0.001 in Schritten von 0.0001 (links)
  - min_samples_leaf von 1 bis 35 (rechts)





4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.1 GridSearchCV - Mit einem Parameter

- mit sklearn.model_selection.GridSearchCV steht eine Klasse zur Verfügung, welche folgende Methoden kombiniert:
  - Parameter Tuning für einen oder mehrere Parameter gleichzeitig
  - Kreuzvalidierung
- die Methode nimmt als Parameter param_grid ein Dictionary (oder eine Liste von Dictionaries) entgegen, worin die anzuwendenden Werte für die Hyperparameter hinterlegt sind, z.B. für min_impurity_decrease bei DecisionTreeClassifier:

```
params = {'min_impurity_decrease': np.arange(0, 0.001, 0.0001)}
print(type(params))
print(params)
```

```
<class 'dict'>
{'min_impurity_decrease': array([0.    , 0.0001, 0.0002, 0.0003, 0.0004, 0.0005,
0.0006, 0.0007,
       0.0008, 0.0009])}
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.1 GridSearchCV - Mit einem Parameter

- ausgeführt

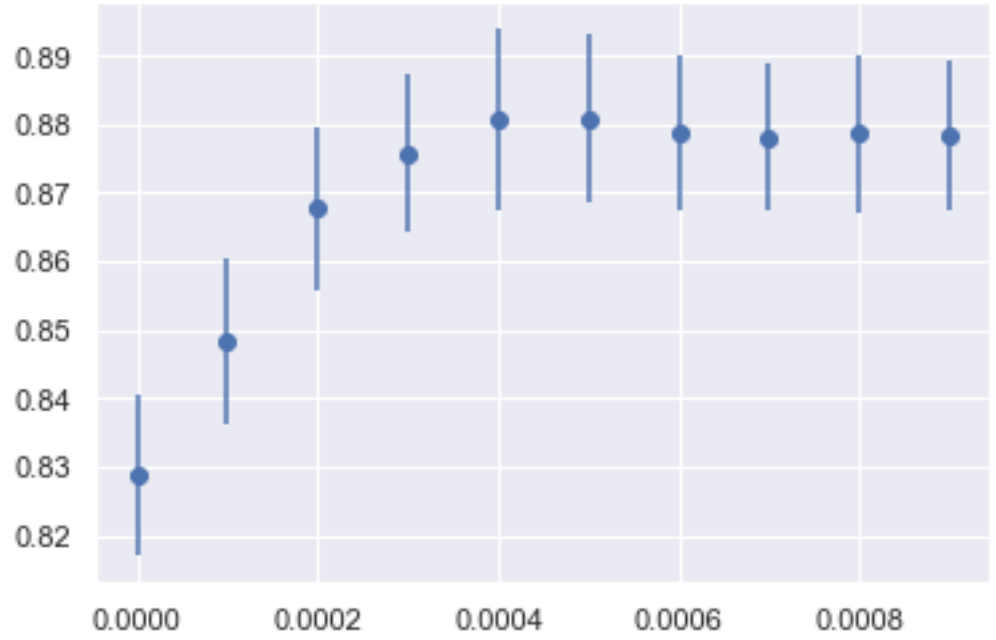
```
from sklearn.model_selection import GridSearchCV
params = {'min_impurity_decrease': np.arange(0, 0.001, 0.0001)}
model = DecisionTreeClassifier(random_state=1234)
gscv = GridSearchCV(model, param_grid=params, cv=10)
gscv.fit(X_train, y_train)
cv_df = pd.DataFrame(gscv.cv_results_)
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.1 GridSearchCV - Mit einem Parameter

- visualisierung mit errorbar()

```
plt.errorbar(
    'param_min_impurity_decrease',
    'mean_test_score',
    'std_test_score',
    data = cv_df, linestyle='None',
    marker='o')
plt.show()
```



- dieses Verfahren erlaubt eine Beurteilung der Performance unabhängig von random_state
- es gibt daher Hinweise zur *Stabilität* des Learners

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.2 GridSearchCV - Mit mehr als einem Parameter

- GridSearchCV erlaubt auch das Kombinieren von mehreren Parametern gleichzeitig
- dadurch wird ein zwei- bis mehrdimensionales Raster von Kombinationen von Parameterwerten angewendet
- im untenstehenden Beispiel werden zwei Parameter gleichzeitig verwendet
  - min_impurity_decrease: [0.0002, 0.0004, 0.0006, 0.0008]
  - min_samples_leaf: [5, 10, 15, 20]
- das ergibt zusammen 16 verschiedene Kombinationen von Parameterwerten
- die Definition des Grid erfolgt analog:

```
:
params = {'min_impurity_decrease': np.arange(0.0002, 0.001, 0.0002),
          'min_samples_leaf': range(5, 25, 5)}
:
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.2 GridSearchCV - Mit mehr als einem Parameter

- sichten der ermittelten Resultate, z.B. als Pivot-Tabelle

```
pvt = pd.pivot_table(pd.DataFrame(gscv.cv_results_),
    values='mean_test_score',
    index='param_min_impurity_decrease',
    columns='param_min_samples_leaf')
print(pvt)
```

```
param_min_samples_leaf                5         10        15        20
param_min_impurity_decrease
0.0002                         0.865557  0.872359  0.874964  0.871635
0.0004                         0.881910  0.879740  0.879884  0.876990
0.0006                         0.879740  0.879305  0.878871  0.878437
0.0008                         0.878582  0.878437  0.877858  0.877424
```

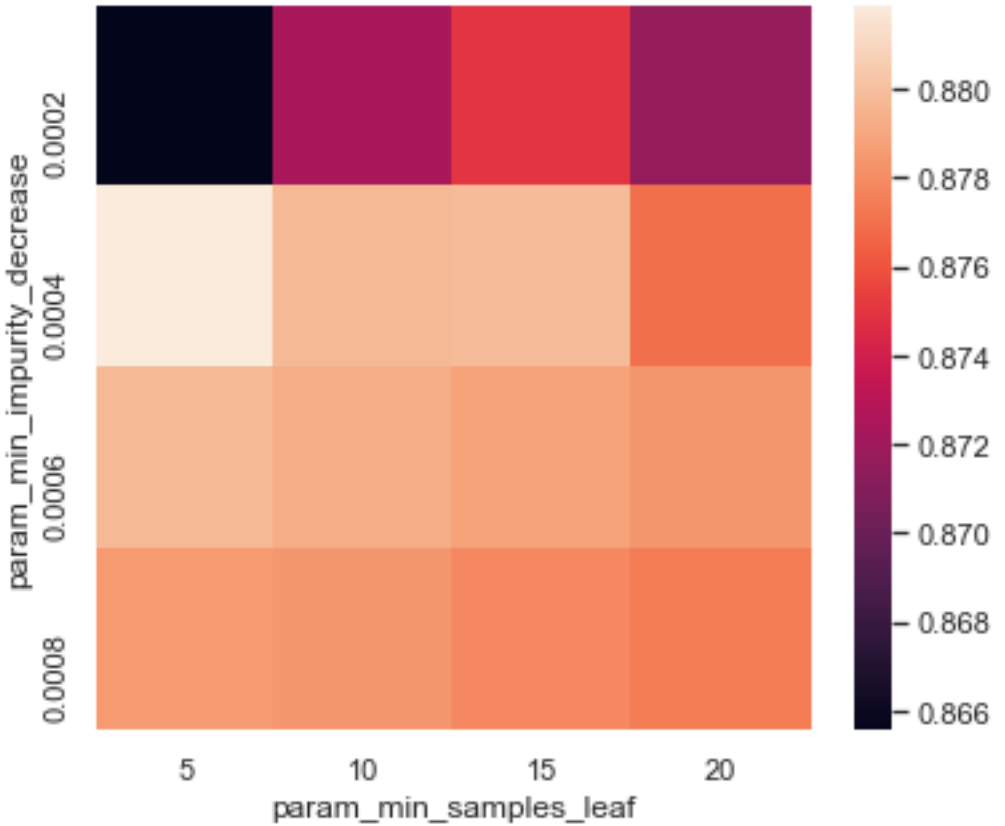
4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.2 GridSearchCV - Mit mehr als einem Parameter

- zum Visualisieren einer derartigen Datenstruktur bietet sich z.B. eine Heatmap an

```
sns.heatmap(pvt)
plt.show()
```

- vgl. dabei die Legende rechts:
  - kleinste Werte: dunkel
  - grösste Werte: hell
    (auch wenn sich diese Werte nur zwischen 0.866 und 0.880 bewegen)



4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.2 GridSearchCV - Mit mehr als einem Parameter

- ausserdem stehen zur Diagnose in der Klasse verschiedene Attribute zur Verfügung, z.B.

```
print('best_params_ :', gscv.best_params_)
print('best_score_  :', gscv.best_score_)
print('refit_time_  :', gscv.refit_time_)
```

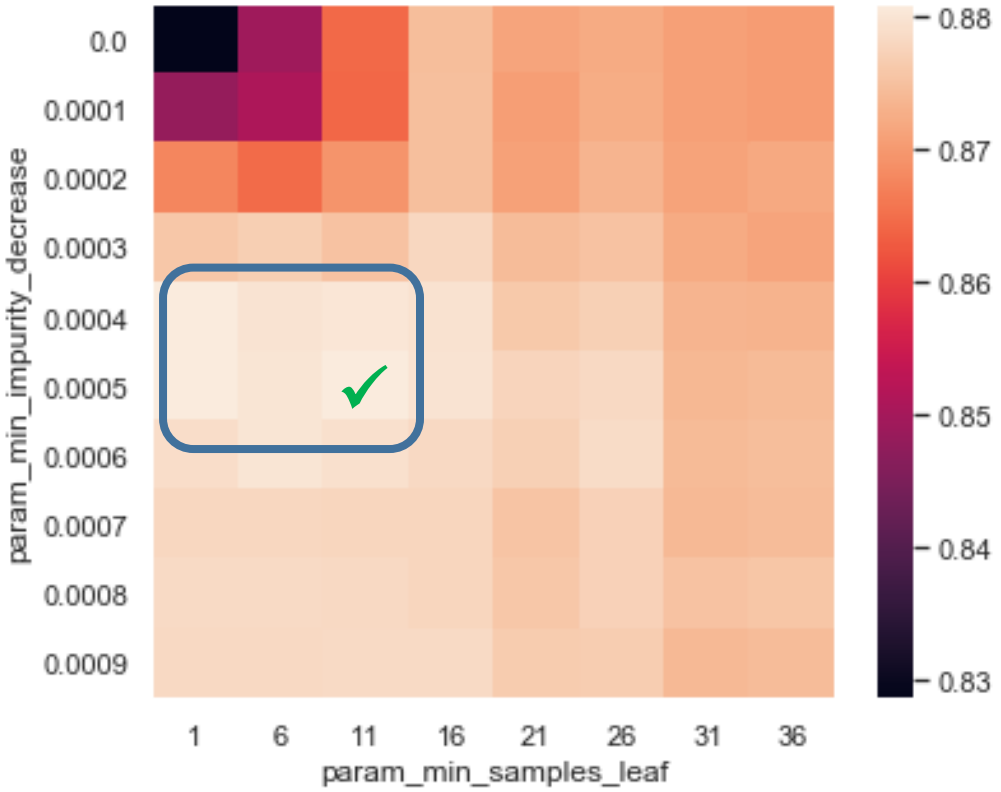
```
best_params_ : {'min_impurity_decrease': 0.0004, 'min_samples_leaf': 5}
best_score_  : 0.881910274963820 5
refit_time_  : 0.046860218048095 7
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.2.2 GridSearchCV - Mit mehr als einem Parameter

- eine etwas feinere Gliederung der Werte der Tuning Parameter finde sich im [ipynb]:
  - min_samples_leaf: 1, 6, 11, 16, 21, 26, 31, 36
  - min_impurity_decrease: 0-0.001
    in Schritten von 0.0001
- Zeitbedarf insgesamt: ca. 50"
- als best_params werden folgende Werte ausgegeben:

```
best_params_ : {
    'min_impurity_decrease': 0.0005,
    'min_samples_leaf': 11}
best_score_  : 0.8808972503617947
```



4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

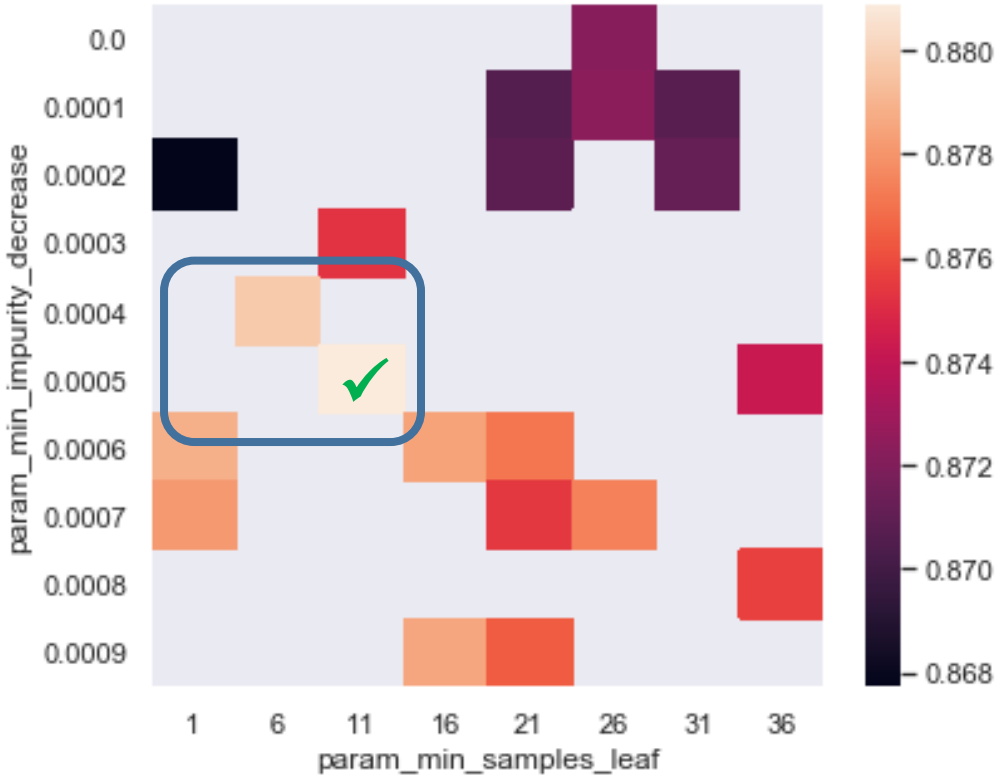
4.3.3 RandomizedSearchCV

- das Ganze kann auf 3, 4 oder mehr Parameter ausgedehnt werden
- aber problematisch, da die Anzahl Tests das kartesische Produkt der Anzahl Levels jeden Parameters entspricht
- z.B. 4 Parameter mit je 10 Werten ergibt 10'000 unterschiedliche Kombinationen, die alle durchgetestet werden müssten
  - und dann kommt noch die Anzahl Folds der Kreuzvalidierung dazu

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.3 RandomizedSearchCV

- Abhilfe: RandomizedSearchCV:
  funktioniert analog GridSearchCV, mit dem Unterschied, dass nur noch eine Zufallsstichprobe aus allen Kombinationen der Parameterwerte untersucht wird
- Parameter
  - param_distribution: nimmt das vorbereitete Parameter Grid entgegen
  - n_iter: Anzahl Kombinationen
- die etwas andere Einfärbung gegenüber der GridSearch Heatmap erklärt sich durch den anderen Wertebereich
  (vgl. [ipynb])



4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.3 RandomizedSearchCV

- sowohl GridSearchCV wie RandomizedSearchCV können auch auf 3 oder mehr Parameter ausgeweitet werden
- eine Visualisierung z.B. mit Heatmap ist dann aber nicht mehr möglich
- trotzdem können mit den Attributen
  - best_params_
  - best_score_
    die Ergebnisse in einem Report gesichtet werden
- Nachtrag: definieren eines Range für nicht ganzzahlige Tuning Parameter
  - im letzten Beispiel wird ein Grid mit fixen Werten definiert
  - für Tuning-Parameter mit nicht ganzzahligen Werten (z.B. min_impurity_decrease) kann anstelle einer Liste auch ein Range mit einer Modellverteilung definiert werden, aus welchem für jede Iteration ein zufälliger Wert generiert wird (vgl: "It is highly recommended to use continuous distributions for continuous parameters.")
  - Beispielcode:

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.3 RandomizedSearchCV

- z.B. wie folgt (vgl. [ipynb]):

```
from scipy.stats.distributions import uniform
params = {'max_depth': range(1, 101),
          'min_samples_leaf': range(1, 21),
          'min_impurity_decrease': uniform(0, 0.001)}
```

- nach erfolgter Kreuzvalidierung mit RandomizedSearchCV können die wichtigsten Ergebnisse wie folgt abgerufen werden:

```
print('best_params_ :', rscv.best_params_)
print('best_score_ :', rscv.best_score_)
```

```
best_params_ : {'max_depth': 74, 'min_impurity_decrease': 0.000507281928841736,
'min_samples_leaf': 11}
best_score_     : 0.8780028943560059
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearchCV

4.3.3 RandomizedSearchCV

- sowie alle Details (soweit Bedarf dazu besteht)

```
print(rscv.cv_results_)
```

```
{'mean_fit_time': array([0.05086336, 0.04974766, 0.04516296, 0.04639444,
0.04700551,
       0.04655466, 0.04498959, 0.04897432, 0.047753  , 0.04603143,
       0.04679971, 0.05562248, 0.04719   , 0.04307485, 0.05609059,
       0.05744905, 0.04845166, 0.04661279, 0.04991403, 0.0451561 ]),
'std_fit_time': array([0.00447742, 0.006115  , 0.00374077, 0.00443711,
0.00413382,
       0.00390924, 0.00338776, 0.00315332, 0.00408781, 0.00433044,
       0.00383186, 0.0038746 , 0.00517052, 0.00302239, 0.00460161,
       0.00470857, 0.00422683, 0.00399542, 0.0011293 , 0.00406338]),
'mean_score_time': array([0.01034484, 0.00633292, 0.00688591, 0.00864611,
0.00867858,
:
```

4.3 Validierung und mehr - GridSearchCV und RandomizedSearch

Workshop 14

Gruppen zu 2 bis 4, Zeit: 30'

- untersuchen Sie Kombinationen von Parameterwerten bei RandomForestClassifier mit RandomizedSearchCV(
- Vorschlag:
  - n_estimators in [50, 100, 150, 200]
  - max_features in [3, 5, 7, 9]
  - criterion in ['gini', 'entropy']
  - min_samples_leaf in [1, 2, 3, 4]
- wenden Sie 5-fach Kreuzvalidierung an
- setzen Sie die Anzahl der zu untersuchenden Kombinationen auf 12
- arbeiten Sie ohne setzen von random_state, damit anschliessend die Ergebnisse verglichen werden können