# GRIDSEARCHCV - OPTYMALIZACJA HIPERPARAMETRÓW

## Wprowadzenie

**GridSearchCV** to narzędzie z biblioteki scikit-learn, które automatycznie znajduje najlepsze parametry dla modelu machine learning. Zamiast ręcznie testować różne kombinacje parametrów, GridSearchCV robi to za Ciebie!

**Jak to działa?**

Wyobraź sobie, że masz model drzewa decyzyjnego i chcesz znaleźć najlepsze wartości dla:

* max\_depth (głębokość drzewa)
* min\_samples\_split (minimalna liczba próbek do podziału)

**Bez GridSearchCV:**

# Musisz ręcznie testować każdą kombinację:

for depth in [2, 3, 4, 5]:

for split in [2, 5, 10]:

model = DecisionTreeClassifier(max\_depth=depth, min\_samples\_split=split)

model.fit(X\_train, y\_train)

score = model.score(X\_test, y\_test)

# zapisz wyniki...

**Z GridSearchCV:**

# GridSearchCV robi to wszystko automatycznie!

param\_grid = {

'max\_depth': [2, 3, 4, 5],

'min\_samples\_split': [2, 5, 10]

}

grid\_search = GridSearchCV(model, param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

print(grid\_search.best\_params\_) # Najlepsze parametry!

## Kluczowe pojęcia

**1. Hiperparametry**

**Parametry, które ustawiamy PRZED treningiem modelu.**

Przykłady:

* Dla drzewa decyzyjnego: max\_depth, min\_samples\_split
* Dla KNN: n\_neighbors, metric
* Dla regresji: alpha, penalty

**2. Siatka parametrów (Parameter Grid)**

**Wszystkie kombinacje parametrów do przetestowania.**

param\_grid = {

'max\_depth': [2, 3, 4, 5, 6], # 5 wartości

'min\_samples\_split': [2, 5, 10, 15] # 4 wartości

}

# Łącznie: 5 × 4 = 20 kombinacji

**3. Cross-Validation (CV)**

**Metoda oceny modelu na różnych fragmentach danych.**

Dla cv=5 (5-fold cross-validation):

1. Dane dzielone na 5 części
2. Model trenowany 5 razy
3. Za każdym razem inna część jest zbiorem testowym
4. Wynik końcowy = średnia z 5 prób

**Dlaczego to ważne?**

* Pojedynczy podział train/test może być przypadkowy
* CV daje bardziej wiarygodną ocenę modelu

**4. Scoring (Metryka oceny)**

**Jak mierzymy, który model jest lepszy?**

Przykłady:

* 'accuracy' - procent poprawnych predykcji
* 'f1' - średnia harmoniczna precision i recall
* 'roc\_auc' - pole pod krzywą ROC

## Implementacja - Kompletny kod

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.tree import DecisionTreeClassifier

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

# 1. WCZYTANIE DANYCH

iris = load\_iris()

X, y = iris.data, iris.target

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# 2. DEFINICJA PARAMETRÓW

param\_grid = {

'max\_depth': [2, 3, 4, 5, 6],

'min\_samples\_split': [2, 5, 10, 15]

}

# 3. GRIDSEARCHCV

model = DecisionTreeClassifier(random\_state=42)

grid\_search = GridSearchCV(

estimator=model,

param\_grid=param\_grid,

cv=5,

scoring='accuracy',

return\_train\_score=True # ważne dla wizualizacji!

)

print("Przeszukuję parametry...")

grid\_search.fit(X\_train, y\_train)

# 4. PRZYGOTOWANIE DANYCH DO WIZUALIZACJI

results = pd.DataFrame(grid\_search.cv\_results\_)

# ==============================================================================

# WIZUALIZACJA 1: HEATMAPA WYNIKÓW

# ==============================================================================

plt.figure(figsize=(10, 6))

# Pivot table dla heatmapy

pivot\_table = results.pivot\_table(

values='mean\_test\_score',

index='param\_max\_depth',

columns='param\_min\_samples\_split'

)

# Rysowanie heatmapy

im = plt.imshow(pivot\_table, cmap='RdYlGn', aspect='auto')

plt.colorbar(im, label='Accuracy')

# Ustawienia osi

plt.xticks(range(len(pivot\_table.columns)), pivot\_table.columns)

plt.yticks(range(len(pivot\_table.index)), pivot\_table.index)

plt.xlabel('min\_samples\_split', fontsize=12)

plt.ylabel('max\_depth', fontsize=12)

plt.title('GridSearchCV - Heatmapa wyników\n(zielone = lepsze, czerwone = gorsze)', fontsize=14)

# Dodanie wartości w komórkach

for i in range(len(pivot\_table.index)):

for j in range(len(pivot\_table.columns)):

text = plt.text(j, i, f'{pivot\_table.iloc[i, j]:.3f}',

ha="center", va="center", color="black", fontsize=9)

plt.tight\_layout()

plt.show()

# ==============================================================================

# WIZUALIZACJA 2: WYKRES SŁUPKOWY TOP 10 KOMBINACJI

# ==============================================================================

plt.figure(figsize=(12, 6))

# Sortowanie wyników

results\_sorted = results.sort\_values('mean\_test\_score', ascending=False).head(10)

# Tworzenie etykiet

labels = [f"depth={row['param\_max\_depth']}\nsplit={row['param\_min\_samples\_split']}"

for \_, row in results\_sorted.iterrows()]

# Wykres słupkowy

bars = plt.bar(range(len(results\_sorted)), results\_sorted['mean\_test\_score'],

color='skyblue', edgecolor='navy')

# Podświetlenie najlepszego wyniku

bars[0].set\_color('gold')

bars[0].set\_edgecolor('orange')

bars[0].set\_linewidth(3)

plt.xticks(range(len(results\_sorted)), labels, rotation=45, ha='right')

plt.ylabel('Accuracy (CV)', fontsize=12)

plt.title('GridSearchCV - Top 10 kombinacji parametrów', fontsize=14)

plt.ylim([results\_sorted['mean\_test\_score'].min() - 0.01,

results\_sorted['mean\_test\_score'].max() + 0.01])

plt.axhline(y=results\_sorted['mean\_test\_score'].iloc[0],

color='red', linestyle='--', alpha=0.5, label='Najlepszy wynik')

plt.legend()

plt.grid(axis='y', alpha=0.3)

plt.tight\_layout()

plt.show()

# ==============================================================================

# WIZUALIZACJA 3: WPŁYW POSZCZEGÓLNYCH PARAMETRÓW

# ==============================================================================

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Wykres 1: Wpływ max\_depth

depth\_scores = results.groupby('param\_max\_depth')['mean\_test\_score'].mean()

axes[0].plot(depth\_scores.index, depth\_scores.values, 'o-', linewidth=2, markersize=8, color='blue')

axes[0].set\_xlabel('max\_depth', fontsize=12)

axes[0].set\_ylabel('Średnia accuracy', fontsize=12)

axes[0].set\_title('Wpływ max\_depth na wyniki', fontsize=13)

axes[0].grid(True, alpha=0.3)

axes[0].fill\_between(depth\_scores.index, depth\_scores.values, alpha=0.3)

# Wykres 2: Wpływ min\_samples\_split

split\_scores = results.groupby('param\_min\_samples\_split')['mean\_test\_score'].mean()

axes[1].plot(split\_scores.index, split\_scores.values, 'o-', linewidth=2, markersize=8, color='green')

axes[1].set\_xlabel('min\_samples\_split', fontsize=12)

axes[1].set\_ylabel('Średnia accuracy', fontsize=12)

axes[1].set\_title('Wpływ min\_samples\_split na wyniki', fontsize=13)

axes[1].grid(True, alpha=0.3)

axes[1].fill\_between(split\_scores.index, split\_scores.values, alpha=0.3, color='green')

plt.tight\_layout()

plt.show()

# ==============================================================================

# PODSUMOWANIE

# ==============================================================================

print("\n" + "="\*60)

print("NAJLEPSZE PARAMETRY:", grid\_search.best\_params\_)

print(f"NAJLEPSZY WYNIK (CV): {grid\_search.best\_score\_:.4f}")

print(f"WYNIK NA ZBIORZE TESTOWYM: {grid\_search.score(X\_test, y\_test):.4f}")

print(f"PRZETESTOWANO {len(results)} kombinacji")

print("="\*60)

## Interpretacja wyników

**Przykładowe wyniki:**

NAJLEPSZE PARAMETRY: {'max\_depth': 4, 'min\_samples\_split': 10}

NAJLEPSZY WYNIK (CV): 0.9429

WYNIK NA ZBIORZE TESTOWYM: 0.9556

PRZETESTOWANO 20 kombinacji

**Co to oznacza?**

1. **Najlepsze parametry:** max\_depth=4, min\_samples\_split=10
   * GridSearchCV przetestował 20 kombinacji
   * Ta kombinacja dała najwyższą dokładność
2. **Wynik CV (0.9429):** Średnia z 5-fold cross-validation
   * Model poprawnie klasyfikuje ~94% próbek na zbiorze treningowym
3. **Wynik testowy (0.9556):** Dokładność na zbiorze testowym
   * Model dobrze generalizuje na nowych danych
   * Wyższy niż CV - dobry znak!

## Interpretacja wizualizacji

### Wizualizacja 1: Heatmapa wyników

**Co pokazuje:**

* Każda komórka = jedna kombinacja parametrów
* **Kolor:**
  + **Zielony** = wysoka dokładność (lepsze)
  + **Żółty** = średnia dokładność
  + **Czerwony** = niska dokładność (gorsze)
* **Liczby w komórkach** = dokładna wartość accuracy

**Jak czytać:**

* **Najlepsze wyniki (ciemnozielone):** max\_depth=4,5,6 i min\_samples\_split=10
* **Najgorsze wyniki (czerwone):** max\_depth=2 (za płytkie drzewo)
* **Wzorzec:** Drzewa zbyt płytkie (max\_depth=2) mają niską dokładność

**Wnioski:**

* Zwiększenie głębokości drzewa z 2 do 3-4 znacznie poprawia wyniki
* min\_samples\_split=10 daje najlepsze rezultaty (zapobiega overfittingowi)
* Dla max\_depth≥4 wyniki są stabilne (~0.93-0.94)

### Wizualizacja 2: Top 10 kombinacji parametrów

**Co pokazuje:**

* 10 najlepszych kombinacji parametrów posortowane malejąco
* **Złoty słupek** = absolutnie najlepsza kombinacja
* **Czerwona linia przerywana** = poziom najlepszego wyniku (do porównania)

**Jak czytać:**

* **1. miejsce (złoty):** depth=4, split=10 - accuracy ~0.943
* **2-3. miejsce (jasnoniebieski):** depth=5/4, split=10 - accuracy ~0.943
* **Reszta kombinacji:** Również dobre wyniki (~0.933)

**Wnioski:**

* Różnice między najlepszymi kombinacjami są małe (0.943 vs 0.933)
* Kilka kombinacji daje podobnie dobre wyniki
* split=10 dominuje w top 3 - jest to stabilna wartość

### Wizualizacja 3: Wpływ poszczególnych parametrów

**Lewy wykres - Wpływ max\_depth:**

**Co pokazuje:**

* Jak zmienia się średnia dokładność wraz ze wzrostem głębokości drzewa
* Uśrednione dla wszystkich wartości min\_samples\_split

**Jak czytać:**

* **max\_depth=2:** Dokładność ~0.89 (najniższa)
* **max\_depth=3:** Skok do ~0.91
* **max\_depth=4-6:** Stabilna dokładność ~0.93

**Wnioski:**

* Drzewa zbyt płytkie (depth=2) są za słabe (underfitting)
* Optymalna głębokość: 3-4
* Dalsze zwiększanie głębokości nie poprawia wyników znacząco

**Prawy wykres - Wpływ min\_samples\_split:**

**Co pokazuje:**

* Jak zmienia się dokładność wraz ze wzrostem minimalnej liczby próbek do podziału
* Uśrednione dla wszystkich wartości max\_depth

**Jak czytać:**

* **split=2:** ~0.925 (nieco niższy)
* **split=5:** ~0.926
* **split=10:** ~0.929 (najwyższy)
* **split=15:** ~0.927 (lekki spadek)

**Wnioski:**

* min\_samples\_split=10 daje najlepsze wyniki
* Zbyt mała wartość (2) może prowadzić do overfittingu
* Zbyt duża wartość (15) może ograniczać możliwości modelu

## Kluczowe parametry GridSearchCV

**estimator**

**Model do optymalizacji**

model = DecisionTreeClassifier(random\_state=42)

**param\_grid**

**Słownik z parametrami do przetestowania**

param\_grid = {

'max\_depth': [2, 3, 4, 5, 6],

'min\_samples\_split': [2, 5, 10, 15]

}

**cv**

**Liczba foldów w cross-validation**

* cv=5 (domyślnie) - 5-fold CV
* Im większa wartość, tym dłużej trwa, ale bardziej wiarygodne wyniki

**scoring**

**Metryka do optymalizacji**

* 'accuracy' - procent poprawnych predykcji
* 'f1' - dla niezbalansowanych klas
* 'roc\_auc' - dla problemów binarnych

**return\_train\_score**

**Czy zwracać wyniki na zbiorze treningowym?**

* True - przydatne do wykrywania overfittingu
* False (domyślnie) - szybsze obliczenia

### Dostęp do wyników

Po wywołaniu grid\_search.fit(), mamy dostęp do:

**Najlepsze parametry**

print(grid\_search.best\_params\_)

# {'max\_depth': 4, 'min\_samples\_split': 10}

**Najlepszy wynik**

print(grid\_search.best\_score\_)

# 0.9429

**Najlepszy model**

best\_model = grid\_search.best\_estimator\_

# Możesz od razu używać tego modelu do predykcji!

**Wszystkie wyniki**

results = pd.DataFrame(grid\_search.cv\_results\_)

# DataFrame ze wszystkimi kombinacjami i ich wynikami

**Predykcja**

# GridSearchCV automatycznie używa najlepszego modelu

y\_pred = grid\_search.predict(X\_test)

## Zalety i wady GridSearchCV

**Zalety:**

1. **Automatyzacja** - nie musisz ręcznie testować każdej kombinacji
2. **Cross-validation** - bardziej wiarygodne wyniki niż pojedynczy train/test split
3. **Prosty w użyciu** - kilka linijek kodu
4. **Zwraca najlepszy model** - gotowy do użycia
5. **Kompletne informacje** - masz dostęp do wszystkich wyników

**Wady:**

1. **Wolny** - testuje WSZYSTKIE kombinacje
   * Dla wielu parametrów może trwać godzinami
   * Przykład: 10 parametrów × 10 wartości = 10 miliardów kombinacji!
2. **Wymaga dużo pamięci** - przechowuje wszystkie wyniki
3. **Może przegapić optima** - jeśli najlepsza wartość jest poza siatką

## Kiedy używać GridSearchCV?

**Używaj gdy:**

* Masz **niewielką liczbę parametrów** do testowania (2-4 parametry)
* Masz **mały lub średni dataset** (do 100k próbek)
* Chcesz **dokładne wyniki** - przetestować każdą kombinację
* Masz **czas** na oczekiwanie (minuty/godziny)

**Nie używaj gdy:**

* Masz **dużo parametrów** (>5 parametrów) → użyj **RandomizedSearchCV**
* Masz **ogromny dataset** (miliony próbek) → za wolne
* Chcesz **szybkie wyniki** → użyj **RandomizedSearchCV**
* Testujesz **ciągłe wartości** (np. learning\_rate od 0.0001 do 1.0) → użyj **RandomizedSearchCV**

## Praktyczne wskazówki

**1. Zacznij od szerokiego zakresu**

# Pierwsza iteracja - szeroki zakres

param\_grid = {

'max\_depth': [2, 5, 10, 20, None],

'min\_samples\_split': [2, 10, 50, 100]

}

**2. Zawęź zakres wokół najlepszych wartości**

# Druga iteracja - precyzyjny zakres

param\_grid = {

'max\_depth': [3, 4, 5, 6, 7],

'min\_samples\_split': [8, 9, 10, 11, 12]

}

**3. Monitoruj czas wykonania**

import time

start = time.time()

grid\_search.fit(X\_train, y\_train)

print(f"Czas: {time.time() - start:.2f} sekund")

**4. Zapisz wyniki**

# Zapisz najlepszy model

import joblib

joblib.dump(grid\_search.best\_estimator\_, 'best\_model.pkl')

# Zapisz wszystkie wyniki

results.to\_csv('grid\_search\_results.csv', index=False)

**5. Równoległe przetwarzanie**

# Przyspiesz obliczenia używając wielu rdzeni

grid\_search = GridSearchCV(

model, param\_grid, cv=5,

n\_jobs=-1 # Użyj wszystkich dostępnych rdzeni

)

## Porównanie: GridSearchCV vs RandomizedSearchCV

| **Cecha** | **GridSearchCV** | **RandomizedSearchCV** |
| --- | --- | --- |
| **Metoda** | Testuje WSZYSTKIE kombinacje | Testuje LOSOWĄ próbkę |
| **Szybkość** | Wolniejszy | Szybszy |
| **Dokładność** | Gwarantuje znalezienie optimum w siatce | Może przegapić optimum |
| **Parametry** | param\_grid (lista wartości) | param\_distributions (rozkłady) |
| **Liczba testów** | Automatyczna (wszystkie) | Określasz n\_iter |
| **Kiedy użyć** | Niewiele parametrów | Dużo parametrów |

**Przykład:**

* GridSearchCV z 5 parametrami × 10 wartości = **100,000 kombinacji**
* RandomizedSearchCV z n\_iter=100 = tylko **100 testów**