# RANDOMIZEDSEARCHCV - SZYBKA OPTYMALIZACJA HIPERPARAMETRÓW

## Wprowadzenie

**RandomizedSearchCV** to alternatywa dla GridSearchCV, która zamiast testować WSZYSTKIE kombinacje parametrów, testuje tylko **losową próbkę**. To sprawia, że jest znacznie szybsza, szczególnie gdy masz wiele parametrów do optymalizacji!

**Różnica między GridSearchCV a RandomizedSearchCV**

**GridSearchCV:**

# Testuje WSZYSTKIE kombinacje

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 (WSZYSTKIE będą przetestowane)

**RandomizedSearchCV:**

# Testuje tylko LOSOWĄ próbkę

param\_distributions = {

'max\_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10], # 9 wartości

'min\_samples\_split': [2, 3, 5, 7, 10, 15, 20], # 7 wartości

'min\_samples\_leaf': [1, 2, 3, 4, 5] # 5 wartości

}

# Możliwych kombinacji: 9 × 7 × 5 = 315

# RandomizedSearchCV testuje tylko 30 losowych! (n\_iter=30)

**Kiedy używać RandomizedSearchCV?**

**Używaj RandomizedSearchCV gdy:**

* Masz **wiele parametrów** (>4 parametry)
* Masz **duży zakres wartości** do przetestowania
* Chcesz **szybkich wyników** (minuty zamiast godzin)
* Testujesz **ciągłe wartości** (np. learning\_rate)
* Masz **ograniczony czas** obliczeń

**Używaj GridSearchCV gdy:**

* Masz **niewiele parametrów** (2-3 parametry)
* Masz **mały zakres wartości**
* Chcesz **przetestować każdą kombinację**
* Masz **czas** i zasoby obliczeniowe

# Kluczowe pojęcia

**1. Losowe próbkowanie (Random Sampling)**

**Zamiast testować wszystkie kombinacje, losujemy próbkę.**

Przykład:

* Możliwych kombinacji: **315**
* Testujemy losowo: **30** (około 10%)
* Oszczędność czasu: **90%**!

**2. Parameter Distributions**

**Zamiast listy wartości, możemy użyć rozkładów statystycznych.**

from scipy.stats import randint, uniform

param\_distributions = {

'max\_depth': randint(2, 20), # Losowa liczba z zakresu 2-20

'min\_samples\_split': randint(2, 50), # Losowa liczba z zakresu 2-50

'learning\_rate': uniform(0.01, 0.5) # Losowa wartość z zakresu 0.01-0.51

}

Dla prostoty w naszym przykładzie używamy list, ale rozkłady dają większą elastyczność.

**3. n\_iter**

**Liczba losowych kombinacji do przetestowania.**

n\_iter=30 # Przetestuj 30 losowych kombinacji

**Jak wybrać n\_iter?**

* Więcej = dokładniejsze wyniki, ale dłużej trwa
* Mniej = szybsze, ale może przegapić optimum
* Typowo: **10-100** iteracji
* Reguła kciuka: **n\_iter ≈ 10-20% wszystkich kombinacji**

**4. random\_state**

**Zapewnia powtarzalność wyników.**

random\_state=42 # Za każdym razem te same losowe kombinacje

## Implementacja - Kompletny kod

from sklearn.datasets import load\_iris

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

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 (szeroki zakres!)

param\_distributions = {

'max\_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10],

'min\_samples\_split': [2, 3, 5, 7, 10, 15, 20],

'min\_samples\_leaf': [1, 2, 3, 4, 5]

}

# Możliwych kombinacji: 9 × 7 × 5 = 315

# 3. RANDOMIZEDSEARCHCV

model = DecisionTreeClassifier(random\_state=42)

random\_search = RandomizedSearchCV(

estimator=model,

param\_distributions=param\_distributions,

n\_iter=30, # testujemy tylko 30 losowych kombinacji!

cv=5,

scoring='accuracy',

random\_state=42,

return\_train\_score=True

)

print("Losowe przeszukiwanie parametrów...")

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

# 4. PRZYGOTOWANIE DANYCH

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

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

# WIZUALIZACJA 1: WYNIKI POSORTOWANE (WODOSPAD)

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

plt.figure(figsize=(14, 6))

# Sortowanie

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

x\_pos = range(len(results\_sorted))

# Wykres

colors = ['gold' if i == 0 else 'lightcoral' if i >= len(results\_sorted)-5

else 'skyblue' for i in range(len(results\_sorted))]

bars = plt.bar(x\_pos, results\_sorted['mean\_test\_score'], color=colors,

edgecolor='navy', linewidth=1)

# Oznaczenie najlepszego i najgorszych

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

bars[0].set\_linewidth(3)

plt.xlabel('Kombinacja parametrów (posortowane)', fontsize=12)

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

plt.title(f'RandomizedSearchCV - Wszystkie {len(results)} przetestowane kombinacje\n(złoty=najlepszy, czerwone=najgorsze)', fontsize=14)

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

linestyle='--', alpha=0.5, label=f'Najlepszy: {results\_sorted["mean\_test\_score"].iloc[0]:.4f}')

plt.axhline(y=results\_sorted['mean\_test\_score'].mean(), color='blue',

linestyle='--', alpha=0.5, label=f'Średnia: {results\_sorted["mean\_test\_score"].mean():.4f}')

plt.legend()

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

plt.tight\_layout()

plt.show()

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

# WIZUALIZACJA 2: TOP 10 KOMBINACJI Z WARTOŚCIAMI PARAMETRÓW

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

plt.figure(figsize=(14, 7))

top10 = results\_sorted.head(10)

# Tworzenie szczegółowych etykiet

labels = []

for idx, row in top10.iterrows():

label = f"depth={row['param\_max\_depth']}\n"

label += f"split={row['param\_min\_samples\_split']}\n"

label += f"leaf={row['param\_min\_samples\_leaf']}"

labels.append(label)

# Wykres

bars = plt.bar(range(len(top10)), top10['mean\_test\_score'],

color=plt.cm.RdYlGn(np.linspace(0.5, 0.9, len(top10))),

edgecolor='black', linewidth=1.5)

# Dodanie wartości na słupkach

for i, (bar, score) in enumerate(zip(bars, top10['mean\_test\_score'])):

plt.text(bar.get\_x() + bar.get\_width()/2, bar.get\_height() + 0.002,

f'{score:.4f}', ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.xticks(range(len(top10)), labels, rotation=0, ha='center', fontsize=9)

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

plt.title('RandomizedSearchCV - Top 10 najlepszych kombinacji', fontsize=14)

plt.ylim([top10['mean\_test\_score'].min() - 0.02,

top10['mean\_test\_score'].max() + 0.015])

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

plt.tight\_layout()

plt.show()

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

# WIZUALIZACJA 3: ROZKŁAD WYNIKÓW (HISTOGRAM + BOX PLOT)

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

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

# Histogram

axes[0].hist(results['mean\_test\_score'], bins=15, color='skyblue',

edgecolor='navy', alpha=0.7)

axes[0].axvline(results['mean\_test\_score'].mean(), color='red',

linestyle='--', linewidth=2, label=f'Średnia: {results["mean\_test\_score"].mean():.4f}')

axes[0].axvline(random\_search.best\_score\_, color='gold',

linestyle='--', linewidth=2, label=f'Najlepszy: {random\_search.best\_score\_:.4f}')

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

axes[0].set\_ylabel('Liczba kombinacji', fontsize=12)

axes[0].set\_title('Rozkład wyników (histogram)', fontsize=13)

axes[0].legend()

axes[0].grid(axis='y', alpha=0.3)

# Box plot

box = axes[1].boxplot([results['mean\_test\_score']],

vert=True, patch\_artist=True, widths=0.5)

box['boxes'][0].set\_facecolor('lightblue')

box['boxes'][0].set\_edgecolor('navy')

box['boxes'][0].set\_linewidth(2)

box['medians'][0].set\_color('red')

box['medians'][0].set\_linewidth(2)

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

axes[1].set\_title('Box plot wyników', fontsize=13)

axes[1].set\_xticklabels(['Wszystkie\nkombinacje'])

axes[1].grid(axis='y', alpha=0.3)

# Dodanie punktu dla najlepszego wyniku

axes[1].scatter([1], [random\_search.best\_score\_], color='gold',

s=200, zorder=5, edgecolor='orange', linewidth=2,

label='Najlepszy', marker='\*')

axes[1].legend()

plt.tight\_layout()

plt.show()

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

# WIZUALIZACJA 4: PORÓWNANIE Z PRZESTRZENIĄ PARAMETRÓW

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

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

# Wyciągnięcie wartości parametrów

depths = results['param\_max\_depth'].values

splits = results['param\_min\_samples\_split'].values

scores = results['mean\_test\_score'].values

# Scatter plot 3D

scatter = ax.scatter(depths, splits, scores,

c=scores, cmap='RdYlGn',

s=100, alpha=0.6, edgecolors='black')

# Oznaczenie najlepszego punktu

best\_idx = results['mean\_test\_score'].idxmax()

ax.scatter([results.loc[best\_idx, 'param\_max\_depth']],

[results.loc[best\_idx, 'param\_min\_samples\_split']],

[results.loc[best\_idx, 'mean\_test\_score']],

color='gold', s=300, marker='\*', edgecolors='orange',

linewidths=2, label='Najlepszy')

ax.set\_xlabel('max\_depth', fontsize=11)

ax.set\_ylabel('min\_samples\_split', fontsize=11)

ax.set\_zlabel('Accuracy', fontsize=11)

ax.set\_title('RandomizedSearchCV - Przestrzeń parametrów 3D\n(30 losowych punktów z 315 możliwych)', fontsize=13)

plt.colorbar(scatter, label='Accuracy', pad=0.1)

ax.legend()

plt.tight\_layout()

plt.show()

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

# PODSUMOWANIE

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

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

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

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

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

print(f"PRZETESTOWANO: {len(results)} z 315 możliwych kombinacji ({len(results)/315\*100:.1f}%)")

print(f"ŚREDNI WYNIK: {results['mean\_test\_score'].mean():.4f}")

print(f"ODCHYLENIE: {results['mean\_test\_score'].std():.4f}")

print("="\*60)

**Interpretacja wyników**

**Przykładowe wyniki:**

NAJLEPSZE PARAMETRY: {'min\_samples\_split': 2, 'min\_samples\_leaf': 1, 'max\_depth': 4}

NAJLEPSZY WYNIK (CV): 0.9524

WYNIK NA ZBIORZE TESTOWYM: 0.9556

PRZETESTOWANO: 30 z 315 możliwych kombinacji (9.5%)

ŚREDNI WYNIK: 0.9276

ODCHYLENIE: 0.0157

**Co to oznacza?**

1. **Najlepsze parametry:**
   * max\_depth=4, min\_samples\_split=2, min\_samples\_leaf=1
   * Znalezione losowo w 30 próbach spośród 315 możliwości!
2. **Wynik CV (0.9524):**
   * Najlepszy znaleziony wynik z 5-fold cross-validation
   * Accuracy ~95% na zbiorze treningowym
3. **Wynik testowy (0.9556):**
   * Model dobrze generalizuje na nowych danych
   * Nawet lepszy niż CV - dobry znak!
4. **Przetestowano tylko 9.5% kombinacji:**
   * Zamiast wszystkich 315 kombinacji, przetestowano tylko 30
   * **Oszczędność czasu: ~90%!**
5. **Średni wynik (0.9276):**
   * Większość losowych kombinacji daje około 93% dokładności
   * Pokazuje, że większość parametrów jest "w miarę dobrych"
6. **Odchylenie standardowe (0.0157):**
   * Niskie odchylenie = wyniki są stabilne
   * Losowe kombinacje nie różnią się drastycznie

## Interpretacja wizualizacji

### Wizualizacja 1: Wykres wodospadowy (wszystkie kombinacje posortowane)

**Co pokazuje:**

* Wszystkie 30 przetestowanych kombinacji posortowane malejąco
* **Kolory:**
  + **Złoty** = najlepsza kombinacja
  + **Niebieski** = przeciętne kombinacje
  + **Czerwony** = 5 najgorszych kombinacji
* **Linie przerywane:**
  + Czerwona linia = najlepszy wynik (dla porównania)
  + Niebieska linia = średni wynik ze wszystkich kombinacji

**Jak czytać:**

* **Najlepszy słupek (złoty):** Accuracy ~0.952
* **Najgorsze słupki (czerwone):** Accuracy ~0.905-0.910
* **Różnica między najlepszym a najgorszym:** ~0.047 (5 punktów procentowych)

**Wnioski:**

* Większość kombinacji daje podobne wyniki (0.92-0.95)
* Nawet "najgorsze" kombinacje mają przyzwoitą dokładność (~91%)
* Jest wyraźna różnica między top 5 a resztą
* Kształt "wodospadu" pokazuje stopniowy spadek jakości

**Porównanie z GridSearchCV:**

* W GridSearchCV zobaczylibyśmy WSZYSTKIE 315 kombinacji
* Tutaj widzimy tylko 30 losowych próbek
* Mimo to znaleźliśmy bardzo dobry wynik!

### Wizualizacja 2: Top 10 najlepszych kombinacji

**Co pokazuje:**

* 10 najlepszych kombinacji z dokładnymi wartościami parametrów
* **Kolory:** Gradient od zielonego (najlepszy) do żółtego
* **Liczby na słupkach:** Dokładne wartości accuracy dla każdej kombinacji

**Jak czytać:**

* **1. miejsce:** depth=4, split=2, leaf=1 → Accuracy 0.9524
* **2. miejsce:** depth=7, split=10, leaf=1 → Accuracy 0.9429
* **3. miejsce:** depth=4, split=10, leaf=1 → Accuracy 0.9429
* **Pozostałe:** Accuracy ~0.935-0.943

**Wnioski:**

* Najlepsza kombinacja wyraźnie wyprzedza pozostałe
* Miejsca 2-4 mają bardzo podobne wyniki (~0.943)
* min\_samples\_leaf=1 dominuje w top 10 (pojawia się w większości)
* max\_depth waha się między 4-7 (średnia głębokość)
* Różnice między top 10 są minimalne (0.952 vs 0.933 = 2%)

**Praktyczny wniosek:**

* Kilka kombinacji daje podobnie dobre wyniki
* Możesz wybrać prostszy model (np. depth=4) zamiast złożonego (depth=7)
* Większa prostota = mniejsze ryzyko overfittingu

### Wizualizacja 3: Rozkład wyników (histogram + box plot)

**Lewy wykres - Histogram:**

**Co pokazuje:**

* Rozkład wszystkich 30 wyników accuracy
* Oś X: Wartości accuracy
* Oś Y: Liczba kombinacji z danym wynikiem

**Jak czytać:**

* **Czerwona linia przerywana:** Średnia ~0.928
* **Złota linia przerywana:** Najlepszy wynik ~0.952
* **Kształt rozkładu:** Lekko lewoskośny (większość wyników blisko średniej)

**Wnioski:**

* Najwięcej kombinacji osiąga accuracy około 0.92-0.94
* Tylko kilka kombinacji ma wyniki poniżej 0.91
* Najlepszy wynik jest wyraźnie powyżej średniej
* Rozkład jest stosunkowo zwart (małe rozproszenie)

**Prawy wykres - Box plot:**

**Co pokazuje:**

* Statystyczne podsumowanie rozkładu wyników
* **Elementy:**
  + Pudełko (niebieski prostokąt) = zakres międzykwartylowy (50% środkowych wyników)
  + Czerwona linia w środku = mediana
  + Wąsy = zakres danych (bez outlierów)
  + Złota gwiazdka = najlepszy wynik

**Jak czytać:**

* **Mediana (czerwona linia):** ~0.930
* **Q1 (dolna krawędź pudełka):** ~0.920
* **Q3 (górna krawędź pudełka):** ~0.940
* **Zakres międzykwartylowy (IQR):** ~0.020
* **Najlepszy wynik (złota gwiazdka):** ~0.952

**Wnioski:**

* 50% wyników mieści się w zakresie 0.92-0.94
* Mediana jest blisko średniej (rozkład symetryczny)
* Brak outlierów - wszystkie wyniki są "rozsądne"
* Najlepszy wynik jest powyżej górnego wąsa (wyjątkowo dobry!)

**Porównanie:**

* Histogram pokazuje SZCZEGÓŁY rozkładu
* Box plot pokazuje STATYSTYKI rozkładu
* Oba potwierdzają: większość wyników jest dobra, najlepszy jest wyjątkowy

### Wizualizacja 4: Przestrzeń parametrów 3D

**Co pokazuje:**

* Przestrzenne rozmieszczenie 30 przetestowanych kombinacji
* **Osie:**
  + Oś X: max\_depth (głębokość drzewa)
  + Oś Y: min\_samples\_split (minimalna liczba próbek do podziału)
  + Oś Z: accuracy (dokładność modelu)
* **Kolory punktów:**
  + Zielony = wysoka accuracy
  + Żółty = średnia accuracy
  + Czerwony = niska accuracy
* **Złota gwiazdka:** Najlepsza znaleziona kombinacja

**Jak czytać:**

* Punkty są **LOSOWO** rozmieszczone w przestrzeni parametrów
* Nie wszystkie możliwe miejsca są pokryte (30 z 315)
* Wysokość punktu = jego dokładność
* Najlepszy punkt jest najwyżej i oznaczony gwiazdką

**Wnioski:**

* Punkty NIE tworzą regularnej siatki (jak w GridSearchCV)
* Losowe rozmieszczenie pokrywa różne regiony przestrzeni
* Najlepsze wyniki (zielone punkty wysoko) dla:
  + max\_depth około 4-7
  + min\_samples\_split niezależny (różne wartości dają dobre wyniki)
* Najgorsze wyniki (czerwone punkty nisko):
  + Różne kombinacje, bez wyraźnego wzorca

**Porównanie z GridSearchCV:**

* **GridSearchCV:** Regularna siatka punktów (wszystkie miejsca)
* **RandomizedSearchCV:** Losowe punkty (tylko niektóre miejsca)
* Mimo mniejszego pokrycia, znaleźliśmy bardzo dobry wynik!

**Co pokazuje losowość:**

* Niektóre regiony są gęsto próbkowane
* Inne regiony mogą być pominięte
* To ryzyko RandomizedSearchCV - możemy przegapić optimum

**Praktyczny wniosek:**

* Jeśli chcesz więcej pewności, zwiększ n\_iter
* Jeśli chcesz szybkości, zmniejsz n\_iter
* 30 iteracji to dobry kompromis dla tego problemu

## Kluczowe parametry RandomizedSearchCV

**estimator**

**Model do optymalizacji**

model = DecisionTreeClassifier(random\_state=42)

**param\_distributions**

**Słownik z rozkładami parametrów do losowania**

param\_distributions = {

'max\_depth': [2, 3, 4, 5, 6, 7, 8, 9, 10], # Lista wartości

'min\_samples\_split': [2, 3, 5, 7, 10, 15, 20] # Lista wartości

}

# lub

from scipy.stats import randint

param\_distributions = {

'max\_depth': randint(2, 20), # Rozkład jednorodny z zakresu 2-20

}

**n\_iter**

**Liczba losowych kombinacji do przetestowania**

n\_iter=30 # Przetestuj 30 losowych kombinacji

* **Więcej = dokładniej, ale wolniej**
* **Mniej = szybciej, ale może przegapić optimum**

**cv**

**Liczba foldów w cross-validation**

cv=5 # 5-fold cross-validation

**scoring**

**Metryka do optymalizacji**

scoring='accuracy' # Maksymalizuj accuracy

**random\_state**

**Ziarno losowości dla powtarzalności**

random\_state=42 # Zawsze te same losowe kombinacje

**return\_train\_score**

**Czy zwracać wyniki na zbiorze treningowym?**

return\_train\_score=True # Przydatne do wykrywania overfittingu

## Dostęp do wyników

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

**Najlepsze parametry**

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

# {'max\_depth': 4, 'min\_samples\_split': 2, 'min\_samples\_leaf': 1}

**Najlepszy wynik**

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

# 0.9524

**Najlepszy model**

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

# Gotowy model do predykcji!

**Wszystkie wyniki**

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

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

# Przetestowano 30 kombinacji

**Statystyki**

print(f"Średni wynik: {results['mean\_test\_score'].mean():.4f}")

print(f"Odchylenie: {results['mean\_test\_score'].std():.4f}")

print(f"Najlepszy: {results['mean\_test\_score'].max():.4f}")

print(f"Najgorszy: {results['mean\_test\_score'].min():.4f}")

## Zalety i wady RandomizedSearchCV

**Zalety:**

1. **Szybkość**
   * Testuje tylko próbkę, nie wszystkie kombinacje
   * Przykład: 30 z 315 = oszczędność 90% czasu!
2. **Skalowalność**
   * Działa dobrze z wieloma parametrami
   * Możesz testować bardzo szerokie zakresy
3. **Elastyczność**
   * Możesz użyć rozkładów statystycznych
   * Lepiej dla parametrów ciągłych (np. learning\_rate)
4. **Nieoczekiwane odkrycia**
   * Losowość może znaleźć dobre kombinacje, których nie przewidziałeś
   * Eksploruje różne regiony przestrzeni parametrów
5. **Kontrola czasu**
   * Możesz precyzyjnie ustawić liczbę iteracji
   * Przewidywalny czas wykonania

**Wady:**

1. **Brak gwarancji**
   * Może przegapić optimum
   * Nie testuje wszystkich możliwości
2. **Losowość**
   * Wyniki mogą się różnić (mimo random\_state dla n\_iter)
   * Może "nie trafić" w dobrą kombinację
3. **Trudniejsze dobieranie n\_iter**
   * Za mało iteracji = słabe wyniki
   * Za dużo iteracji = marnowanie czasu
4. **Mniej intuicyjne wyniki**
   * Trudniej zrozumieć wpływ parametrów
   * Brak regularnej siatki do analizy

## Porównanie: GridSearchCV vs RandomizedSearchCV

| **Aspekt** | **GridSearchCV** | **RandomizedSearchCV** |
| --- | --- | --- |
| **Metoda** | Testuje WSZYSTKIE kombinacje | Testuje LOSOWĄ próbkę |
| **Szybkość** | Wolniejszy (100%) | Szybszy (10-50%) |
| **Gwarancja** | Znajdzie optimum w siatce | Może przegapić optimum |
| **Parametry** | param\_grid (lista) | param\_distributions (lista lub rozkład) |
| **Kontrola** | Liczba automatyczna | Określasz n\_iter |
| **Dobre dla** | Mało parametrów (2-4) | Dużo parametrów (>4) |
| **Zakres** | Wąski, precyzyjny | Szeroki, eksploracyjny |
| **Czas (przykład)** | 315 kombinacji | 30 kombinacji |

**Przykład praktyczny:**

**Problem:** Optymalizacja 5 parametrów, każdy po 10 wartości

* **GridSearchCV:** 10^5 = **100,000 kombinacji** (dni/tygodnie)
* **RandomizedSearchCV:** n\_iter=100 = **100 kombinacji** (minuty/godziny)

**Redukcja czasu: 99.9%!**

## PODSUMOWANIE

**Czym jest RandomizedSearchCV?**

**Inteligentne narzędzie do szybkiej optymalizacji hiperparametrów poprzez testowanie losowej próbki kombinacji parametrów zamiast wszystkich możliwości.**

**Kluczowe różnice vs GridSearchCV:**

| **Co?** | **GridSearchCV** | **RandomizedSearchCV** |
| --- | --- | --- |
| Testuje | Wszystkie | Losową próbkę |
| Czas | Długi | Krótki |
| Parametr | param\_grid | param\_distributions + n\_iter |
| Optymum | Gwarantowane | Prawdopodobne |

**Kluczowe kroki:**

1. **Zdefiniuj rozkłady parametrów** (param\_distributions)
2. **Ustaw liczbę iteracji** (n\_iter)
3. **Wybierz metodę oceny** (scoring)
4. **Ustaw cross-validation** (cv)
5. **Wywołaj fit()** - RandomizedSearchCV losuje i testuje
6. **Użyj najlepszego modelu** (best\_estimator\_)

**Kiedy używać?**

**RandomizedSearchCV:**

* Dużo parametrów (>4)
* Szeroki zakres wartości
* Ograniczony czas
* Eksploracja przestrzeni parametrów

**GridSearchCV:**

* Mało parametrów (2-3)
* Wąski, precyzyjny zakres
* Dużo czasu
* Potrzeba gwarancji znalezienia optimum

**Złota zasada:**

**Zacznij od RandomizedSearchCV dla szerokiej eksploracji, potem użyj GridSearchCV dla precyzyjnej optymalizacji wokół znalezionych wartości!**

**Dla datasetu Iris:**

**Przetestowano:** 30 z 315 kombinacji (9.5%)  
**Najlepsze parametry:** max\_depth=4, min\_samples\_split=2, min\_samples\_leaf=1  
**Accuracy:** ~95% (5-fold CV)  
**Oszczędność czasu:** ~90%!  
**Wniosek:** Nawet z 10% kombinacji znaleźliśmy doskonały wynik!