# MODELE KLASYFIKACJI W MACHINE LEARNING

## Wprowadzenie do Klasyfikacji

Klasyfikacja to zadanie uczenia nadzorowanego, w którym algorytm uczy się przypisywać obserwacje do predefiniowanych kategorii (klas) na podstawie ich cech. Przykłady zastosowań:

* Rozpoznawanie chorób na podstawie objawów
* Filtrowanie spamu w emailach
* Rozpoznawanie obrazów (koty vs psy)
* Przewidywanie, czy klient dokona zakupu

**Podstawowe pojęcia:**

* **Cechy (features)** - zmienne wejściowe opisujące obserwację (np. waga, wzrost)
* **Etykiety (labels)** - kategorie, do których przypisujemy obserwacje (np. gatunek rośliny)
* **Zbiór treningowy** - dane używane do uczenia modelu
* **Zbiór testowy** - dane używane do oceny jakości modelu

## 1. DRZEWA DECYZYJNE (Decision Trees)

**Jak działają?**

Drzewo decyzyjne to struktura przypominająca schemat blokowy, która podejmuje decyzje na podstawie serii pytań o cechy danych. Algorytm działa następująco:

1. **Wybór najlepszej cechy** - algorytm wybiera cechę, która najlepiej dzieli dane na grupy
2. **Podział danych** - dane dzielone są na podstawie wartości wybranej cechy
3. **Rekurencyjne powtarzanie** - proces powtarza się dla każdej podgrupy
4. **Zatrzymanie** - gdy wszystkie elementy w grupie należą do jednej klasy lub osiągnięto maksymalną głębokość

**Przykład decyzji:**

Czy długość płatka > 2.5cm?

├─ TAK → Czy szerokość płatka > 1.7cm?

│ ├─ TAK → Iris Virginica

│ └─ NIE → Iris Versicolor

└─ NIE → Iris Setosa

**Kluczowe parametry**

**criterion** - miara jakości podziału:

* gini - indeks Giniego (domyślnie), mierzy "nieczystość" węzła
* entropy - entropia informacyjna, mierzy niepewność

**max\_depth** - maksymalna głębokość drzewa:

* Kontroluje złożoność modelu
* Niższe wartości = prostsze drzewo = mniej przeuczenia
* Wyższe wartości = bardziej złożone drzewo = ryzyko overfittingu

**min\_samples\_split** - minimalna liczba próbek wymagana do podziału węzła:

* Wyższe wartości = bardziej konserwatywne dzielenie
* Pomaga zapobiegać przeuczeniu

**Zalety i wady**

**Zalety:**

* Bardzo intuicyjne i łatwe do zrozumienia
* Można je wizualizować i wyjaśnić decyzje
* Nie wymagają normalizacji ani standaryzacji danych
* Radzą sobie z danymi numerycznymi i kategorycznymi
* Automatycznie wykrywają interakcje między cechami

**Wady:**

* Podatne na przeuczenie (overfitting)
* Niestabilne - małe zmiany w danych mogą całkowicie zmienić strukturę drzewa
* Mogą tworzyć złożone drzewa, które słabo generalizują
* Preferują cechy z wieloma wartościami

**Implementacja w Pythonie**

from sklearn.tree import DecisionTreeClassifier

from sklearn.datasets import load\_iris

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

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import matplotlib.pyplot as plt

from sklearn import tree

# Wczytanie danych

iris = load\_iris()

X, y = iris.data, iris.target

# Podział na zbiór treningowy i testowy

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

# Utworzenie i trenowanie modelu

dt\_classifier = DecisionTreeClassifier(

criterion='gini', # lub 'entropy'

max\_depth=3, # ograniczenie głębokości

min\_samples\_split=2,

random\_state=42

)

dt\_classifier.fit(X\_train, y\_train)

# Predykcja

y\_pred = dt\_classifier.predict(X\_test)

# Ewaluacja

print(f"Dokładność: {accuracy\_score(y\_test, y\_pred):.4f}")

print("\nRaport klasyfikacji:")

print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))

# Wizualizacja drzewa

plt.figure(figsize=(15, 10))

tree.plot\_tree(dt\_classifier, feature\_names=iris.feature\_names,

class\_names=iris.target\_names, filled=True)

plt.show()

**Interpretacja wyników**

**Dokładność (Accuracy)** - procent poprawnie sklasyfikowanych obserwacji

**Raport klasyfikacji zawiera:**

* **Precision** (precyzja) - ile z przewidzianych przypadków danej klasy było prawdziwie tą klasą
* **Recall** (czułość) - ile przypadków danej klasy udało się wykryć
* **F1-score** - średnia harmoniczna precision i recall
* **Support** - liczba rzeczywistych wystąpień klasy w zbiorze testowym

**Wizualizacja drzewa:**

* Każdy prostokąt to węzeł decyzyjny
* Kolory wskazują dominującą klasę w węźle
* Im intensywniejszy kolor, tym bardziej "czysta" jest klasa w węźle
* Liczby pokazują rozkład próbek z każdej klasy

## 2. K-NEAREST NEIGHBORS (KNN)

**Jak działa?**

KNN to algorytm "leniwego uczenia" - nie buduje jawnego modelu, tylko zapamiętuje dane treningowe. Klasyfikacja nowej obserwacji przebiega następująco:

1. **Oblicz odległości** - mierzy się odległość nowej obserwacji do wszystkich punktów treningowych
2. **Znajdź k najbliższych sąsiadów** - wybiera się k punktów o najmniejszej odległości
3. **Głosowanie większościowe** - nowa obserwacja otrzymuje klasę, która jest najczęstsza wśród k sąsiadów

**Intuicja:** "Powiedz mi kim są twoi sąsiedzi, a powiem ci kim jesteś"

**Kluczowe decyzje**

**Wybór k:**

* **k = 1**: Każdy punkt klasyfikowany jak najbliższy sąsiad (ryzyko overfittingu, wrażliwy na szum)
* **k małe (3-5)**: Model bardziej złożony, lepiej dopasowany do szczegółów
* **k duże (>10)**: Model prostszy, bardziej wygładzony, bardziej odporne na szum
* **k nieparzyste**: Zapobiega remisowi w klasyfikacji binarnej

**Metryki odległości:**

* euclidean - odległość w linii prostej (najpopularniejsza)
* manhattan - suma różnic w każdym wymiarze (dobra dla przestrzeni miejskich)
* minkowski - uogólnienie euclidean i manhattan

**Wagi (weights):**

* uniform - wszyscy sąsiedzi mają równy głos
* distance - bliżsi sąsiedzi mają większy wpływ (lepsze dla niejednorodnych danych)

**Znaczenie standaryzacji**

**Dlaczego standaryzacja jest kluczowa dla KNN?**

KNN opiera się na odległościach, więc cechy o większych wartościach numerycznych będą dominować w obliczeniach, nawet jeśli nie są ważniejsze.

**Przykład bez standaryzacji:**

* Cecha A: waga w gramach (50-100)
* Cecha B: wzrost w metrach (0.5-1.0)

Odległość będzie zdominowana przez wagę, bo ma większe wartości!

**StandardScaler:**

* Przekształca każdą cechę tak, aby miała średnią = 0 i odchylenie standardowe = 1
* Wszystkie cechy mają teraz porównywalny zakres
* fit\_transform() na zbiorze treningowym - uczy się parametrów
* transform() na zbiorze testowym - stosuje te same parametry

**Implementacja w Pythonie**

from matplotlib import pyplot as plt

from sklearn.neighbors import KNeighborsClassifier

from sklearn.preprocessing import StandardScaler

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

from sklearn.metrics import accuracy\_score, confusion\_matrix

from sklearn.datasets import load\_iris

import numpy as np

# 1. ZAŁADOWANIE DANYCH (przykład z Iris dataset)

iris = load\_iris()

X = iris.data

y = iris.target

# 2. PODZIAŁ NA ZBIÓR TRENINGOWY I TESTOWY

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# 3. STANDARYZACJA DANYCH (ważne dla KNN!)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# 4. ZNALEZIENIE OPTYMALNEGO k

k\_range = range(1, 31)

scores = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train\_scaled, y\_train)

scores.append(knn.score(X\_test\_scaled, y\_test))

# 5. WIZUALIZACJA WYNIKÓW DLA RÓŻNYCH k

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

plt.plot(k\_range, scores, marker='o')

plt.xlabel('Wartość k')

plt.ylabel('Dokładność')

plt.title('Dokładność KNN dla różnych wartości k')

plt.grid(True)

plt.show()

# 6. TRENING Z OPTYMALNYM k

optimal\_k = k\_range[np.argmax(scores)]

print(f"Optymalne k: {optimal\_k}")

print(f"Najwyższa dokładność: {max(scores):.4f}")

knn\_classifier = KNeighborsClassifier(

n\_neighbors=optimal\_k,

weights='distance', # lub 'uniform'

metric='euclidean' # lub 'manhattan', 'minkowski'

)

knn\_classifier.fit(X\_train\_scaled, y\_train)

# 7. PREDYKCJA I EWALUACJA

y\_pred\_knn = knn\_classifier.predict(X\_test\_scaled)

print(f"\nDokładność KNN: {accuracy\_score(y\_test, y\_pred\_knn):.4f}")

print("\nMacierz pomyłek:")

print(confusion\_matrix(y\_test, y\_pred\_knn))

**Optymalizacja k**

**Wykres dokładności vs k:**

* Niska wartość k (1-3): Wysoka zmienność, możliwe przeuczenie
* Optymalne k: Najwyższa dokładność na zbiorze testowym
* Wysoka wartość k: Model zbyt prosty, możliwe niedouczenie

**Strategia wyboru k:**

1. Testuj zakres wartości (np. 1-30)
2. Wybierz k z najwyższą dokładnością na zbiorze walidacyjnym
3. Jeśli kilka wartości daje podobne wyniki, wybierz większe k (bardziej stabilne)

**Zalety i wady**

**Zalety:**

* Bardzo prosty i intuicyjny
* Nie wymaga trenowania (brak fazy uczenia)
* Naturalnie obsługuje klasyfikację wieloklasową
* Dobrze działa z nieliniowymi granicami decyzyjnymi

**Wady:**

* Bardzo wolny przy predykcji dla dużych zbiorów (musi obliczyć wszystkie odległości)
* Wymaga dużo pamięci (przechowuje wszystkie dane treningowe)
* Wrażliwy na skalę cech (konieczna standaryzacja)
* Wrażliwy na nieistotne cechy
* Słabo działa w wysokich wymiarach (curse of dimensionality)

## 3. NAIVE BAYES

**Jak działa?**

Naive Bayes to klasyfikator probabilistyczny oparty na twierdzeniu Bayesa:

**P(klasa|cechy) = P(cechy|klasa) × P(klasa) / P(cechy)**

**Założenie "naiwności":** Algorytm zakłada, że wszystkie cechy są niezależne od siebie (dlatego "naiwny"). To założenie rzadko jest prawdziwe, ale mimo to algorytm często działa dobrze!

**Proces klasyfikacji:**

1. Dla każdej klasy oblicz prawdopodobieństwo P(klasa|cechy)
2. Wybierz klasę o najwyższym prawdopodobieństwie

**Warianty Naive Bayes**

**GaussianNB** (użyty w kodzie):

* Dla cech ciągłych (numerycznych)
* Zakłada, że cechy mają rozkład normalny (Gaussa)
* Idealny dla danych takich jak Iris (wymiary płatków/działek)

**MultinomialNB:**

* Dla cech dyskretnych (zliczania, częstości)
* Świetny dla klasyfikacji tekstu (liczba wystąpień słów)

**BernoulliNB:**

* Dla cech binarnych (0/1, tak/nie)
* Użyteczny dla wykrywania obecności/braku cech

**Implementacja w Pythonie**

from sklearn.naive\_bayes import GaussianNB, MultinomialNB

from sklearn.metrics import roc\_curve, auc, accuracy\_score, classification\_report

from sklearn.preprocessing import label\_binarize

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

from sklearn.datasets import load\_iris

from matplotlib import pyplot as plt

import numpy as np

# 1. ZAŁADOWANIE DANYCH

iris = load\_iris()

X = iris.data

y = iris.target

# 2. PODZIAŁ NA ZBIÓR TRENINGOWY I TESTOWY

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# 3. GAUSSIAN NAIVE BAYES (dla cech ciągłych)

gnb\_classifier = GaussianNB()

gnb\_classifier.fit(X\_train, y\_train)

# 4. PREDYKCJA

y\_pred\_gnb = gnb\_classifier.predict(X\_test)

y\_proba\_gnb = gnb\_classifier.predict\_proba(X\_test)

print(f"Dokładność Gaussian Naive Bayes: {accuracy\_score(y\_test, y\_pred\_gnb):.4f}")

print("\nRaport klasyfikacji:")

print(classification\_report(y\_test, y\_pred\_gnb, target\_names=iris.target\_names))

# 5. KRZYWA ROC DLA KLASYFIKACJI WIELOKLASOWEJ

y\_test\_bin = label\_binarize(y\_test, classes=[0, 1, 2])

n\_classes = y\_test\_bin.shape[1]

plt.figure(figsize=(10, 8))

for i in range(n\_classes):

fpr, tpr, \_ = roc\_curve(y\_test\_bin[:, i], y\_proba\_gnb[:, i])

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'{iris.target\_names[i]} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label='Losowy klasyfikator')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Krzywa ROC - Naive Bayes')

plt.legend()

plt.grid(True)

plt.show()

**Krzywa ROC i AUC**

**ROC (Receiver Operating Characteristic):**

* Pokazuje kompromis między czułością (True Positive Rate) a False Positive Rate
* Oś X: False Positive Rate (ile zdrowych błędnie uznano za chorych)
* Oś Y: True Positive Rate (ile chorych poprawnie zidentyfikowano)

**AUC (Area Under Curve):**

* Pole pod krzywą ROC
* AUC = 1.0: Idealny klasyfikator
* AUC = 0.5: Klasyfikator losowy (linia przerywana)
* AUC > 0.8: Dobry klasyfikator
* AUC > 0.9: Doskonały klasyfikator

**Klasyfikacja wieloklasowa:** Dla problemów z więcej niż dwiema klasami (jak Iris: 3 gatunki), tworzymy oddzielną krzywą ROC dla każdej klasy metodą "one-vs-rest":

* Klasa A vs (B+C)
* Klasa B vs (A+C)
* Klasa C vs (A+B)

**Zalety i wady**

**Zalety:**

* Bardzo szybki w treningu i predykcji
* Wymaga małej ilości danych treningowych
* Dobrze działa z wysokowymiarowymi danymi
* Doskonały dla klasyfikacji tekstu i analizy sentymentu
* Daje oszacowania prawdopodobieństw

**Wady:**

* Założenie niezależności cech jest często nierealistyczne
* Może dawać skrajne prawdopodobieństwa (bardzo bliskie 0 lub 1)
* Słabo działa, gdy zależności między cechami są ważne
* Wrażliwy na cechy niepowiązane z klasyfikacją

## 4. PORÓWNANIE MODELI

**Implementacja w Pythonie (KOD 4)**

from sklearn.model\_selection import cross\_val\_score, train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

from sklearn.datasets import load\_iris

from matplotlib import pyplot as plt

import numpy as np

# 1. ZAŁADOWANIE DANYCH

iris = load\_iris()

X = iris.data

y = iris.target

# 2. PODZIAŁ NA ZBIÓR TRENINGOWY I TESTOWY (do znalezienia optimal\_k)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42

)

# 3. INICJALIZACJA SCALERA

scaler = StandardScaler()

# 4. ZNALEZIENIE OPTYMALNEGO k DLA KNN

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

k\_range = range(1, 31)

scores = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors=k)

knn.fit(X\_train\_scaled, y\_train)

scores.append(knn.score(X\_test\_scaled, y\_test))

optimal\_k = k\_range[np.argmax(scores)]

print(f"Optymalne k: {optimal\_k}\n")

# 5. PORÓWNANIE WSZYSTKICH TRZECH MODELI

models = {

'Decision Tree': DecisionTreeClassifier(max\_depth=3, random\_state=42),

'KNN': KNeighborsClassifier(n\_neighbors=optimal\_k),

'Naive Bayes': GaussianNB()

}

results = {}

print("Wyniki cross-validation:")

print("-" \* 50)

for name, model in models.items():

if name == 'KNN':

# KNN wymaga standaryzacji

X\_scaled = scaler.fit\_transform(X)

scores = cross\_val\_score(model, X\_scaled, y, cv=5)

else:

scores = cross\_val\_score(model, X, y, cv=5)

results[name] = scores

print(f"{name}: {scores.mean():.4f} (+/- {scores.std():.4f})")

# 6. WIZUALIZACJA PORÓWNANIA

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

plt.boxplot(results.values(), tick\_labels=results.keys())

plt.ylabel('Dokładność')

plt.title('Porównanie modeli klasyfikacji (5-fold CV)')

plt.grid(True)

plt.xticks(rotation=15)

plt.tight\_layout()

plt.show()

**Cross-Validation (Walidacja Krzyżowa)**

**Dlaczego cross-validation?** Pojedynczy podział na train/test może być przypadkowy. Cross-validation daje bardziej wiarygodną ocenę:

**5-fold CV:**

1. Dane dzielone na 5 równych części
2. Model trenowany 5 razy, każdy raz inna część jest testem
3. Wynik to średnia z 5 prób

**Interpretacja wyników:**

* **Średnia** - ogólna wydajność modelu
* **Odchylenie standardowe** - stabilność modelu
  + Małe odchylenie = model stabilny
  + Duże odchylenie = model niestabilny, wrażliwy na dane

**Boxplot - Wizualizacja rozkładu wyników**

**Elementy boxplota:**

* **Linia środkowa** - mediana (50. percentyl)
* **Pudełko** - zakres międzykwartylowy (25-75 percentyl)
* **Wąsy** - zasięg danych (bez outlierów)
* **Punkty** - wartości odstające (outliery)

**Co pokazuje dla modeli:**

* Wysokość pudełka: Jak dużo wyniki się różnią
* Pozycja mediany: Typowa wydajność modelu
* Symetryczność: Czy model stabilnie działa

**Jak wybrać najlepszy model?**

**Kryteria:**

1. **Dokładność** - najwyższa średnia dokładność
2. **Stabilność** - najmniejsze odchylenie standardowe
3. **Interpretowalność** - czy można wyjaśnić decyzje?
4. **Szybkość** - czas treningu i predykcji
5. **Złożoność** - łatwość implementacji i utrzymania

**Dla zbioru Iris (typowe wyniki):**

* **Decision Tree**: ~95-97% - dobra równowaga, łatwa interpretacja
* **KNN**: ~93-96% - prosty, ale wymaga standaryzacji
* **Naive Bayes**: ~94-96% - bardzo szybki, dobre prawdopodobieństwa

**Praktyczne wskazówki**

**Decision Trees - kiedy użyć:**

* Potrzebujesz interpretowalnego modelu
* Masz mieszane typy danych (numeryczne i kategoryczne)
* Chcesz wizualizować proces decyzyjny

**KNN - kiedy użyć:**

* Masz niewielki zbiór danych
* Granice decyzyjne są nieliniowe
* Nie potrzebujesz szybkich predykcji

**Naive Bayes - kiedy użyć:**

* Potrzebujesz bardzo szybkiego modelu
* Pracujesz z tekstem lub danymi o wysokiej wymiarowości
* Chcesz oszacowania prawdopodobieństw
* Masz małą ilość danych treningowych

## ZADANIA PRAKTYCZNE

**Zadanie 1: Eksperyment z parametrami Decision Tree**

Zmień max\_depth (2, 3, 5, 10, None) i porównaj:

* Jak zmienia się dokładność?
* Jak wygląda drzewo dla różnych głębokości?
* Który wariant jest najbardziej zrozumiały?

**Zadanie 2: Wpływ k w KNN**

Uruchom kod dla różnych zakresów k (1-10, 1-50):

* Jak zmienia się optymalne k?
* Czy większy zakres daje lepsze wyniki?
* Co się dzieje dla k=1 vs k=20?

**Zadanie 3: Porównanie metryk odległości**

W KNN zmień metric na 'manhattan' i 'minkowski':

* Która metryka daje najlepsze wyniki?
* Czy optymalne k się zmienia?

**Zadanie 4: Analiza błędów**

Dla każdego modelu przeanalizuj macierz pomyłek:

* Które klasy są najczęściej mylone?
* Dlaczego model popełnia te konkretne błędy?
* Co można poprawić?

**Zadanie 5: Własny dataset**

Załaduj inny dataset (np. wine, breast\_cancer) i:

* Przeprowadź pełną analizę wszystkimi trzema modelami
* Który model działa najlepiej?
* Czy wnioski są inne niż dla Iris?

## PODSUMOWANIE

| **Model** | **Szybkość** | **Interpretowalność** | **Dokładność** | **Najlepsze dla** |
| --- | --- | --- | --- | --- |
| Decision Tree | Szybka | Wysoka | Średnia-wysoka | Dane mieszane, wyjaśnialność |
| KNN | Wolna | Średnia | Średnia-wysoka | Małe zbiory, granice nieliniowe |
| Naive Bayes | Bardzo szybka | Średnia | Średnia | Tekst, wysokie wymiary |

**Złota zasada:** Nie ma jednego "najlepszego" algorytmu. Zawsze testuj kilka podejść i wybieraj na podstawie wyników walidacji krzyżowej oraz wymagań projektu!