1. Strona tytułowa

Nazwa projektu: Klasyfikacja Gatunków Kwiatów Iris z Wykorzystaniem Uczenia Maszynowego

Członkowie grupy:

- Emil Sadkowski
- Jakub Sokołowski
- Szymon Sobczak

Numer grupy: 7 grupa
Kierunek: Informatyka
Rok akademicki: 2025/2026

2. Wstęp teoretyczny

2.1 Omówienie problemu

Klasyfikacja gatunków kwiatów Iris jest klasycznym problemem w dziedzinie uczenia maszynowego, często wykorzystywanym jako benchmark dla algorytmów klasyfikacji. Zadanie polega na przypisaniu kwiatów do jednego z trzech gatunków (*Iris setosa*, *Iris versicolor*, *Iris virginica*) na podstawie czterech cech morfologicznych: długości i szerokości działki kielicha oraz długości i szerokości płatka.

2.2 Znaczenie problemu

Problem klasyfikacji Iris, choć pozornie prosty, stanowi doskonałe wprowadzenie do zagadnień uczenia maszynowego ze względu na:

- Czytelność i interpretowalność danych
- Wyraźnie separowalne klasy
- Niski wymiar przestrzeni cech
- Równowagę klas

2.3 Podstawowe pojęcia

- **Klasyfikacja wieloklasowa**: Zadanie przypisania obiektów do więcej niż dwóch kategorii

- **UCzenie nadzorowane**: Algorytm uczy się na podstawie oznaczonych danych treningowych
- **Walidacja krzyżowa**: Technika oceny modeli poprzez podział danych na k foldów
- **Hiperparametry**: Parametry konfiguracyjne algorytmów ustalane przed procesem uczenia
- **Metryki ewaluacji**: Miary służące do oceny jakości modeli (accuracy, precision, recall, F1-score)

2.4 Przegląd istniejących podejść

Do rozwiązania problemu klasyfikacji Iris stosuje się różne algorytmy:

- **K-Najbliższych Sąsiadów (KNN)**: Algorytm oparty na odległości między punktami
- **Maszyny Wektorów Nośnych (SVM)**: Algorytm znajdujący hiperpłaszczyznę maksymalnego marginesu
- **Las Losowy (Random Forest)**: Ensemble method oparta na drzewach decyzyjnych

3. Opis algorytmu / metody

3.1 Schemat działania

```
[Pobranie danych]
    ↓
[Eksploracja i wizualizacja]
    ↓
[Przygotowanie danych]
    ↓
[Podział na zbiory treningowy/testowy]
    ↓
[Trening wielu modeli]
    ↓
[Walidacja krzyżowa i tuning]
    ↓
[Ewaluacja wyników]
    ↓
[Analiza i interpretacja]
```

3.2 Opis kroków przetwarzania danych

1. **Wczytanie danych**: Import zbioru Iris z biblioteki scikit-learn
2. **Eksploracja**: Analiza statystyk, rozkładu klas, korelacji między cechami
3. **Wizualizacja**: Wykresy pudełkowe, pairplot, macierz korelacji

4. **Przygotowanie**: Podział na zbiory treningowy i testowy z zachowaniem proporcji klas
5. **Normalizacja**: Standaryzacja cech do średniej 0 i wariancji 1
6. **Trening**: Uczenie trzech różnych algorytmów klasyfikacji
7. **Tuning**: Optymalizacja hiperparametrów za pomocą Grid Search
8. **Ewaluacja**: Ocena modeli na zbiorze testowym przy użyciu multipleks metryk

3.3 Uzasadnienie wyboru algorytmów

Wybrano trzy reprezentatywne algorytmy o różnej charakterystyce:

- **KNN**:
  - Prostota implementacji i interpretacji
  - Brak założeń o rozkładzie danych
  - Wrażliwość na skalę danych (wymaga normalizacji)
- **SVM**:
  - Skuteczność w przestrzeniach o wysokiej wymiarowości
  - Odporność na overfitting przy odpowiedniej regularyzacji
  - Zdolność do modelowania nieliniowych granic decyzyjnych
- **Random Forest**:
  - Odporność na overfitting dzięki ensemble learning
  - Zdolność do pracy z danymi niezsynchronizowanymi
  - Dostarcza informacji o ważności cech

4. Użyte dane i przygotowanie danych

4.1 Źródło danych

Iris Dataset - klasyczny zbiór danych wprowadzony przez R.A. Fishera w 1936 roku, dostępny w bibliotece scikit-learn oraz na platformie Kaggle.

4.2 Opis zbioru danych

Zbiór zawiera 150 próbek, po 50 dla każdego z trzech gatunków Iris. Każda próbka opisana jest czterema cechami:

- Sepal length (długość działki kielicha) [cm]
- Sepal width (szerokość działki kielicha) [cm]
- Petal length (długość płatka) [cm]
- Petal width (szerokość płatka) [cm]

4.3 Przygotowanie danych

```
# Podział na zbiory treningowy i testowy
X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.2, random_state=42, stratify=y
```

```
)

# Normalizacja danych
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4.4 Wizualizacja danych

Wykonano następujące wizualizacje:

- **Pairplot**: Pokazuje relacje między wszystkimi parami cech
- **Boxploty**: Prezentują rozkład każdej cechy w podziale na gatunki
- **Macierz korelacji**: Ukazuje współzależności między cechami
- **Wykres słupkowy rozkładu klas**: Weryfikuje balans danych

5. Implementacja

5.1 Architektura programu

Program został zaimplementowany w języku Python przy użyciu notebooka Jupyter/Google Colab. Architektura opiera się na modularnym podejściu z podziałem na logiczne sekcje.

5.2 Użyte biblioteki

- **scikit-learn**: Algorytmy ML, preprocessing, metryki
- **pandas**: Manipulacja i analiza danych
- **numpy**: Operacje numeryczne
- **matplotlib**: Podstawowe wizualizacje
- **seaborn**: Zaawansowane wizualizacje statystyczne

5.3 Kluczowe fragmenty kodu

```
# Definicja modeli
models = {
    'KNN': KNeighborsClassifier(),
    'SVM': SVC(probability=True, random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42)
}

# Tuning hiperparametrów
param_grid_knn = {
    'n_neighbors': range(1, 15),
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan']
```

}

6. Ewaluacja i wyniki

6.1 Metryki ewaluacji

- Accuracy: Ogólna dokładność klasyfikacji
- Precision: Precyzja dla każdej klasy
- Recall: Czułość klasyfikacji
- F1-score: Średnia harmoniczna precyzji i czułości
- ROC-AUC: Powierzchnia pod krzywą ROC

6.2 Wyniki walidacji krzyżowej

Tabela 1: Wyniki 5-krotnej walidacji krzyżowej (średnie)

Model	Accuracy	Precision	Recall	F1-score
KNN	0.9583	0.9644	0.9583	0.9580
SVM	0.9667	0.9704	0.9667	0.9665
Random Forest	0.9500	0.9585	0.9500	0.9494

6.3 Wyniki na zbiorze testowym PRZED tuningiem

Tabela 2: Wyniki na zbiorze testowym przed tuningiem

Model	Accuracy	Precision (macro)	Recall (macro)	F1-score (macro)
KNN	0.9333	0.94	0.93	0.93
SVM	0.9667	0.97	0.97	0.97
Random Forest	0.9000	0.90	0.90	0.90

6.4 Tuning hiperparametrów

Decyzja: Tuning przeprowadzono tylko dla modelu SVM, który osiągnął najlepsze wyniki w walidacji krzyżowej.

Przestrzeń parametrów SVM:

- C: [0.1, 1, 10, 100]
- gamma: ['scale', 'auto', 0.1, 0.01]
- kernel: ['rbf', 'linear']

Wyniki tuningu:

- Najlepsze parametry: {'clf__C': 0.1, 'clf__gamma': 'scale', 'clf__kernel': 'linear'}
- Najlepsze CV accuracy: 0.9750
- Liczba kandydatów: 32
- Łączna liczba fitowań: 160

6.5 Wyniki końcowe PO tuningowaniu

Ostateczny model: SVM z dostrojonymi parametrami
Accuracy na zbiorze testowym: 0.9333

Tabela 3: Szczegółowy raport klasyfikacji dla finalnego modelu (SVM)

Klasa	Precision	Recall	F1-score	Support
setosa	1.00	1.00	1.00	10

versicolor	0.90	0.90	0.90	10
virginica	0.90	0.90	0.90	10
accuracy			0.93	30
macro avg	0.93	0.93	0.93	30
weighted avg	0.93	0.93	0.93	30

7. Analiza wyników i wnioski

7.1 Interpretacja uzyskanych rezultatów

SVM był najlepszy w walidacji krzyżowej (96.67%), co potwierdziło się również na zbiorze testowym przed tuningiem (96.67%)

Po tuningowaniu nastąpił spadek dokładności z 96.67% do 93.33% na zbiorze testowym, pomimo wyższej dokładności CV (97.50%)

Przyczyna rozbieżności:

- Model po tuningowaniu (kernel='linear') jest prostszy i może lepiej generalizować
- Mały zbiór testowy (30 próbek) - 2 błędne klasyfikacje = spadek o 6.67%
- Specyficzny rozkład zbioru testowego

KNN osiągnął bardzo dobre wyniki (95.83% CV, 93.33% test), co potwierdza skuteczność prostych algorytmów dla tego problemu

Random Forest najsłabszy bez tuningu, co sugeruje potrzebę dostrojenia hiperparametr

7.2 Analiza błędów i obserwacje

- Model SVM po tuningowaniu popełnił **2 błędy** na 30 próbek testowych
- Oba błędy dotyczyły rozróżnienia między **versicolor a virginica**
- Klasa **setosa była zawsze poprawnie klasyfikowana** przez wszystkie modele
- **ROC-AUC bliskie 1.0** potwierdzają doskonałą separowalność klas

7.3 Napotkane problemy i rozwiązania

Problem 1: Rozbieżność między wynikami CV a testowymi po tuningowaniu

- Przyczyna: Mały rozmiar zbioru testowego (30 próbek) i specyficzny rozkład danych
- Rozwiązanie: Większy zbiór testowy lub wielokrotne podziały z różnymi seedami

Problem 2: Tuning tylko dla jednego modelu

- Przyczyna: Ograniczenia czasowe - tuning wszystkich modeli byłby obliczeniowo kosztowny
- Rozwiązanie: Skupienie się na najlepszym modelu z CV jest uzasadnione metodologicznie

Problem 3: Brak analizy ważności cech

- Wyjaśnienie: Analiza ważności cech nie została wykonana, ponieważ Random Forest nie był ostatecznym modelem. W kodzie jest ona przewidziana tylko gdy Random Forest jest najlepszy.

7.4 Możliwe kierunki ulepszeń

- Tuning pozostałych modeli (KNN i Random Forest) dla pełnego porównania
- Zastosowanie ensemble methods - głosowanie większościowe między modelami
- Bayesian Optimization zamiast Grid Search dla efektywniejszego tuningu
- Większa liczba podziałów w walidacji krzyżowej

8. Podział pracy w grupie

Członek grupy	Zadania	Wkład (%)
Jakub Sokołowski	Przygotowanie danych, implementacja KNN i SVM, wizualizacje	40%
Szymon Sobczak	Implementacja Random Forest, tuning hiperparametrów, walidacja	35%
Emil Sadkowski	Ewaluacja wyników, analiza ważności cech, dokumentacja	25%

Szczegółowy opis zadań:

Jakub Sokołowski:

- Wczytanie i eksploracja danych
- Implementacja algorytmów KNN i SVM
- Przygotowanie wizualizacji danych
- Analiza wstępnych wyników

Szymon Sobczak:

- Implementacja Random Forest
- Konfiguracja Grid Search dla wszystkich modeli
- Przeprowadzenie walidacji krzyżowej
- Optymalizacja hiperparametrów

Emil Sadkowski:

- Obliczenie metryk ewaluacji
- Analiza ważności cech
- Przygotowanie macierzy pomyłek
- Dokumentacja projektu i wnioski

9. Bibliografia i źródła

Źródła danych

1. Fisher, R. A. (1936). "The use of multiple measurements in taxonomic problems". Annals of Eugenics. 7 (2): 179–188.
2. UCI Machine Learning Repository: Iris Data Set
3. Scikit-learn: `sklearn.datasets.load_iris()`

Dokumentacje i tutoriale

1. Scikit-learn documentation: https://scikit-learn.org/stable/
2. Pandas documentation: https://pandas.pydata.org/
3. Matplotlib documentation: https://matplotlib.org/
4. Seaborn documentation: https://seaborn.pydata.org/

Materiały edukacyjne

1. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning
2. Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras and TensorFlow

Narzędzia

1. Python 3.8+
2. Jupyter Notebook / Google Colab
3. Biblioteki: scikit-learn 1.2+, pandas 1.5+, matplotlib 3.6+

Repozytorium kodu

https://github.com/sadkovvsky/psi/tree/main/Zad1