Projekt_ssi_

June 12, 2024

1 Wstęp

W dzisiejszych czasach, wraz z rozwojem technologii, metody sztucznej inteligencji znajdują coraz szersze zastosowanie w różnych dziedzinach życia. Jednym z obszarów, w którym sztuczna inteligencja może być wykorzystana, jest rozwiązywanie gier logicznych, takich jak Sudoku. Sudoku to popularna łamigłówka logiczna, w której gracz musi wypełnić dziewięć identycznych kwadratów 3x3 cyframi od 1 do 9, w taki sposób, aby każda cyfra nie powtarzała się w wierszu, kolumnie i kwadracie 3x3.

Celem naszego projektu jest porównanie dwóch metod uczenia maszynowego - Maszyn Wektorów Wspierających (SVM) oraz Drzewa Decyzyjnego - pod kątem ich skuteczności w rozwiązaniu łamigłówki Sudoku. Metoda SVM jest techniką klasyfikacji wykorzystującą hiperpłaszczyzny, które oddzielają różne klasy w przestrzeni cech. Z kolei Drzewo Decyzyjne jest modelem predykcyjnym, który dzieli przestrzeń wejściową na podprzestrzenie, stosując warunki decyzyjne oparte na cechach wejściowych.

W ramach projektu przeprowadzimy analizę zbioru danych zawierającego różnorodne łamigłówki Sudoku. Następnie wytrenujemy modele oparte na metodach SVM i Drzewa Decyzyjnego, a następnie przetestujemy ich skuteczność w rozwiązaniu próbnych łamigłówek. Ostatecznie porównamy wyniki uzyskane przez obie metody pod kątem ich dokładności w rozwiązaniu Sudoku oraz ewentualnych zalet i ograniczeń każdej z metod.

Poprzez przeprowadzenie tego projektu, będziemy mogli lepiej zrozumieć, jak różne metody uczenia maszynowego radzą sobie z problemem rozwiązywania gier logicznych oraz jak można je zoptymalizować w celu uzyskania najlepszych wyników.

Sprawdzic można na tej stronie https://sudoku.com/pl/sudoku-solver

A to jest sprawdzenie

2 Analiza

Porównanie co najmniej dwóch algorytmów uczenia maszynowego W ramach analizy porównaliśmy dwie popularne metody uczenia maszynowego: Maszyny Wektorów Wspierających (SVM) oraz Drzewa Decyzyjnego. SVM jest techniką klasyfikacji, która dzieli przestrzeń wejściową na hiperpłaszczyzny, które oddzielają różne klasy. Z kolei Drzewo Decyzyjne to model oparty na strukturze drzewa, gdzie każdy węzeł reprezentuje test na jednej z cech, a każda krawędź reprezentuje możliwy wynik tego testu.

[ ]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import svm, tree
from sklearn.metrics import accuracy_score
import joblib

# Wczytaj zbiór danych cyfr z sklearn
digits = datasets.load_digits()
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

# Podziel dane na zestawy treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(data, digits.target,␣
 ↪test_size=0.5, shuffle=False)

# Utwórz i wytrenuj model SVM
svm_clf = svm.SVC(gamma=0.001)
svm_clf.fit(X_train, y_train)

# Zapisz model SVM
joblib.dump(svm_clf, 'svm_digits.pkl')

# Utwórz i wytrenuj model drzewa decyzyjnego
tree_clf = tree.DecisionTreeClassifier()
tree_clf.fit(X_train, y_train)

# Zapisz model drzewa decyzyjnego
joblib.dump(tree_clf, 'tree_digits.pkl')

# Przetestuj modele
svm_predicted = svm_clf.predict(X_test)
tree_predicted = tree_clf.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predicted)
tree_accuracy = accuracy_score(y_test, tree_predicted)
```

Rozwiązanie sudoku

[ ]:
```python
def solve_sudoku(board):
    def is_valid(board, row, col, num):
        for i in range(9):
            if board[row][i] == num or board[i][col] == num:
                return False
        start_row, start_col = 3 * (row // 3), 3 * (col // 3)
        for i in range(3):
            for j in range(3):
```

```
                if board[start_row + i][start_col + j] == num:
                    return False
        return True

    def solve(board):
        for row in range(9):
            for col in range(9):
                if board[row][col] == 0:
                    for num in range(1, 10):
                        if is_valid(board, row, col, num):
                            board[row][col] = num
                            if solve(board):
                                return True
                            board[row][col] = 0
                    return False
        return True

    solve(board)
    return board

def plot_sudoku(board):
    fig, ax = plt.subplots(figsize=(6, 6))
    min_val, max_val = 0, 9

    # Utwórz siatkę z liczbami
    for i in range(9):
        for j in range(9):
            c = board[i][j]
            if c != 0:
                ax.text(j + 0.5, 8.5 - i, str(c), va='center', ha='center',
                        fontsize=16, bbox=dict(facecolor='white',␣
 ↪edgecolor='none', pad=1))

    # Główne linie siatki
    for i in range(10):
        lw = 2 if i % 3 == 0 else 1
        ax.plot([i, i], [0, 9], 'k', lw=lw)
        ax.plot([0, 9], [i, i], 'k', lw=lw)

    # Ukryj osie
    ax.xaxis.set_ticks_position('none')
    ax.yaxis.set_ticks_position('none')
    ax.invert_yaxis()
    plt.axis('off')
    plt.show()

# Wprowadź łamigłówkę Sudoku jako 2D lista
```

```
input_sudoku = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]


print("Input Sudoku:")
plot_sudoku(input_sudoku)

# Rozwiąż łamigłówkę Sudoku
solved_sudoku = solve_sudoku(input_sudoku)

print("Solved Sudoku:")
plot_sudoku(solved_sudoku)
```

```
Input Sudoku:
```

				8			7	9
			4	1	9			5
	6					2	8	
7				2				6
4			8		3			1
8				6				3
	9	8					6	
6			1	9	5			
5	3			7				

Solved Sudoku:

3	4	5	2	8	6	1	7	9
2	8	7	4	1	9	6	3	5
9	6	1	5	3	7	2	8	4
7	1	3	9	2	4	8	5	6
4	2	6	8	5	3	7	9	1
8	5	9	7	6	1	4	2	3
1	9	8	3	4	2	5	6	7
6	7	2	1	9	5	3	4	8
5	3	4	6	7	8	9	1	2

Weryfikacja wyników za pomocą SVN i drzewa decyzyjnego

[ ]:
```
# Wczytaj wytrenowane modele
svm_clf = joblib.load('svm_digits.pkl')
tree_clf = joblib.load('tree_digits.pkl')

# Funkcja do przekształcania pojedynczej cyfry do obrazu 8x8 pikseli
def digit_to_image(digit):
    # Znajdź odpowiedni obraz ze zbioru danych cyfr
    index = np.where(digits.target == digit)[0][0]
    return digits.images[index].reshape(1, -1)

# Przekształć całe rozwiązane sudoku do formatu oczekiwanego przez modele
def sudoku_to_images(sudoku):
    images = []
    for row in sudoku:
        for digit in row:
            images.append(digit_to_image(digit))
    return np.vstack(images)
```

```
# Pobierz obrazy dla rozwiązanej sudoku
solved_sudoku_images = sudoku_to_images(solved_sudoku)

# Przewiduj każdą cyfrę w rozwiązanym sudoku za pomocą SVM i drzewa decyzyjnego
svm_sudoku_predictions = svm_clf.predict(solved_sudoku_images).reshape(9, 9)
tree_sudoku_predictions = tree_clf.predict(solved_sudoku_images).reshape(9, 9)



# Porównanie metod
svm_accuracy = np.mean(svm_sudoku_predictions == np.array(solved_sudoku))
tree_accuracy = np.mean(tree_sudoku_predictions == np.array(solved_sudoku))

print(f'Porównanie metod:')
print(f'SVN: {svm_accuracy * 100:.2f}% zgodności')
print(f'Drzewo decyzyjne: {tree_accuracy * 100:.2f}% zgodności')

# Funkcja do tworzenia zdjęć z wynikami
def plot_comparison(original, svm, tree):
    fig, axes = plt.subplots(3, 1, figsize=(6, 18))
    titles = ['Oryginalne Sudoku', 'Przewidywania SVN', 'Przewidywania drzewa␣
 ↪decyzyjnego']
    data = [original, svm, tree]

    for ax, title, board in zip(axes, titles, data):
        ax.set_title(title)
        min_val, max_val = 0, 9
        for i in range(9):
            for j in range(9):
                c = board[i][j]
                ax.text(j + 0.5, 8.5 - i, str(c), va='center', ha='center',
                        fontsize=16, bbox=dict(facecolor='white',␣
 ↪edgecolor='none', pad=1))
        for i in range(10):
            lw = 2 if i % 3 == 0 else 1
            ax.plot([i, i], [0, 9], 'k', lw=lw)
            ax.plot([0, 9], [i, i], 'k', lw=lw)
        ax.xaxis.set_ticks_position('none')
        ax.yaxis.set_ticks_position('none')
        ax.invert_yaxis()
        ax.axis('off')

    plt.tight_layout()
    plt.show()
```

```
plot_comparison(np.array(solved_sudoku), svm_sudoku_predictions,␣
  ↪tree_sudoku_predictions)
```

```
Porównanie metod:
SVN: 88.89% zgodności
Drzewo decyzyjne: 100.00% zgodności
```

Oryginalne Sudoku

3	4	5	2	8	6	1	7	9
2	8	7	4	1	9	6	3	5
9	6	1	5	3	7	2	8	4
7	1	3	9	2	4	8	5	6
4	2	6	8	5	3	7	9	1
8	5	9	7	6	1	4	2	3
1	9	8	3	4	2	5	6	7
6	7	2	1	9	5	3	4	8
5	3	4	6	7	8	9	1	2

Przewidywania SVN

3	4	9	2	8	6	1	7	9
2	8	7	4	1	9	6	3	9
9	6	1	9	3	7	2	8	4
7	1	3	9	2	4	8	9	6
4	2	6	8	9	3	7	9	1
8	9	9	7	6	1	4	2	3
1	9	8	3	4	2	9	6	7
6	7	2	1	9	9	3	4	8
9	3	4	6	7	8	9	1	2

Przewidywania drzewa decyzyjnego

3	4	5	2	8	6	1	7	9
2	8	7	4	1	9	6	3	5
9	6	1	5	3	7	2	8	4
7	1	3	9	2	4	8	5	6
4	2	6	8	5	3	7	9	1
8	5	9	7	6	1	4	2	3
1	9	8	3	4	2	5	6	7
6	7	2	1	9	5	3	4	8
5	3	4	6	7	8	9	1	2

[ ]:
```
Inny przykład dla Sudoku
# 1
input_sudoku = [
    [5, 3, 0, 0, 7, 0, 0, 0, 0],
    [6, 0, 0, 1, 9, 5, 0, 0, 0],
    [0, 9, 8, 0, 0, 0, 0, 6, 0],
    [8, 0, 0, 0, 6, 0, 0, 0, 3],
    [4, 0, 0, 8, 0, 3, 0, 0, 1],
    [7, 0, 0, 0, 2, 0, 0, 0, 6],
    [0, 6, 0, 0, 0, 0, 2, 8, 0],
    [0, 0, 0, 4, 1, 9, 0, 0, 5],
    [0, 0, 0, 0, 8, 0, 0, 7, 9]
]
# 2
input_sudoku = [
    [0, 2, 0, 0, 0, 0, 3, 0, 0],
    [0, 0, 0, 0, 0, 5, 4, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 5, 6],
    [0, 0, 0, 0, 6, 0, 0, 0, 0],
    [0, 0, 0, 0, 5, 0, 0, 0, 0],
    [0, 0, 0, 1, 0, 0, 0, 0, 0],
    [0, 0, 0, 7, 0, 0, 0, 0, 0],
    [0, 3, 1, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 4, 0]
]
```

3 Analiza wyników:

SVM: 88.89% zgodności

SVM osiągnął 88.89% zgodności, co oznacza, że w 88.89% przypadków model poprawnie przewidział wartości cyfr w rozwiązanych Sudoku. To wciąż bardzo dobre wyniki, ale może istnieć miejsce do drobnej optymalizacji. Drzewo decyzyjne: 100.00% zgodności

Drzewo decyzyjne osiągnęło imponujące 100% zgodności, co sugeruje, że model ten doskonale radzi sobie z rozpoznawaniem cyfr w rozwiązanych Sudoku. Jest to bardzo obiecujący wynik i świadczy o skuteczności tego podejścia. Wnioski: Na podstawie uzyskanych wyników wydaje się, że drzewo decyzyjne może być lepszym wyborem dla tego konkretnego problemu niż SVM. Jego zdolność do osiągnięcia 100% zgodności sugeruje, że jest ono w stanie dokładnie klasyfikować cyfry w rozwiązanych Sudoku, co czyni je bardziej odpowiednim modelem dla tego zadania.

Jednakże, zanim podjęlibyśmy ostateczną decyzję, warto byłoby zbadać inne aspekty, takie jak czas uczenia się modeli, ich wydajność w złożonych Sudoku czy potencjalne problemy z overfittingiem. Warto również eksperymentować z różnymi wartościami hiperparametrów, aby zoptymalizować wydajność obu metod.