Politechnika Warszawska

Wydział Elektroniki i Technik Informacyjnych

Wstęp do Sztucznej Inteligencji

ćwiczenie 3 - Dwuosobowe gry deterministyczne

Autor:

Jarosław Jaworski 342189

Data oddania ćwiczenia:

26.04.2025

SPIS TREŚCI

1 Cel ćwiczenia 2

2 Przygotowanie zbioru danych 2

3 Stworzenie niestandardowego modelu lasu losowego 3

4 Ocena modelu, porównanie z gotowym rozwiązaniem scikit-learn 6

5 Wnioski 11

1 Cel ćwiczenia

Celem ćwiczenia było zaimplementowanie algorytmu lasu losowego do predykcji wystąpienia choroby serca u przykładowego pacjenta, bazując na jego cechach: płeć, typ bólu w klatce piersiowej, poziom cukru we krwi, itd.

2 Przygotowanie zbioru danych

Pierwszym krokiem przy uczeniu modeli sztucznej inteligencji jest przygotowanie tzw. ***"setów"***, czyli podzestawów danych zestawu źródłowego. W tym wypadku, plik lab-4-dataset zawierał łącznie ponad 900 rekordów z danymi, które podzielono na zestaw uczący (training_set) oraz testujący (test_set). W tym celu zastosowano funkcję ***train_test_split***, wchodzącą w skład biblioteki **scikit learn**. Dzięki temu dane podzielono w stosunku 4:1 (training:test). Dane umieszczone w kolumnach podzielono na zbiór klas wyjściowych (tu: pojedyncza kolumna z informacją binarną o występowaniu choroby) oraz zbiór atrybutów wejściowych (pozostałe kolumny, będące cechami). Ze względu na fakt, że modele AI nie mogą przyjmować napisów (*labeli*), użyto biblioteki **pandas** aby nadać każdej z występujących wartości w kategoriach odpowiednią wartość numeryczną. Całość implementacji przedstawia Wycinek kodu 1.

```
1 def load_and_prepare_data(filepath : str) -> list:
2     # Sczytanie danych z pliku
3     df = pd.read_csv(filepath)
4     # Podzial na podzbiory klas i atrybutow wejsciowych
5     X = df.drop('HeartDisease', axis=1)
6     y = df['HeartDisease']
7     # Utworzenie kolumn binarnych dla kazdego labela
        kazdej kategorii atrybutow wejsciowych
```

```
8     X = pd.get_dummies(X, drop_first=True)
9     # Standardyzacja danych - nadanie sredniej i
          odchylenia standardowego dla kazdej kolumny w X
10    scaler = StandardScaler()
11    X_scaled = scaler.fit_transform(X)
12    # Zwrot listy danych podzielonych na zestaw
          treningowy i testujacy
13    return train_test_split(X_scaled, y, test_size=0.2,
          random_state=42)
```

Wycinek kodu 1: Przygotowanie danych do zadanai klasyfikacji

3 Stworzenie niestandardowego modelu lasu losowego

Kluczowym elementem było utworzenie klasy drzewa decyzyjnego potrafiącej obliczać własną entropię (miarę nieuporządkowania). Z powodu tej potrzeby, zastosowano model Claude'a Shannona stosowany dla teorii informacji. Zależność przedstawia wzór (1).

$$H(S) = -\sum_{i=1}^{n} p_i \cdot \log_2(p_i) \tag{1}$$

,gdzie:

n - liczba klas

p_i - prawdopodobieństwo wystąpienia etykiety danej klasy

Dzięki temu zapewniono poprawiony zysk informacyjny i wybranie najlepszego atrybutu jako kryterium podziału danych w kodzie budującym drzewo. Samo budowanie drzewa odbywało się rekurencyjnie zgodnie z algorytmem ID3.

W każdej iteracji na podstawie węzła rodzica tworzono dzieci zawierające losowo przydzielaną część cech danego pacjenta. Następnie z tych cech wybierano najlepszą co do wartości entropii (najniższą) względem parametru wyjściowego, tym samym zapewniając największy *gain* - zysk informacyjny. Za próg podziału ustalono średnią arytmetyczną kolejnych unikalnych wartości danej cechy (jako, że były to skonwertowane wartości słowne do kolejnych całkowito-liczbowych, to de facto sprowadza się to do thresholdów dzielących pomiędzy kolejnymi wartościami całkowitymi). W kolejnych iteracjach wywoływano algorytm z pominięciem najlepszej znalezionej cechy w poprzedniej pętli.

Każde drzewo zawierało algorytm przewidywania, który w połączeniu wszystkich drzew umożliwiał podjęcie decyzji większościowej. Przedstawiono go na Wycinku kodu 2.

```
# Metoda przewidujaca dla pojedynczego rekordu
def predict_one(self, x):
    node = self.root
    while node is not None:
        # If in leaf, return its value
        if node.feature is None or node.value is not
           None:
            return node.value

        # Get record val
        val = x[node.feature] if isinstance(x,
           (pd.Series, dict)) else x[node.feature]

        # Traverse
        if val < node.threshold:
            node = node.left
```

```
15          else:
16              node = node.right
17
18  # Metoda zbiorcza - wykonujaca przewidywanie na calym
        zbiorze danych (np. testowym)
19  def predict(self, X):
20      if isinstance(X, pd.DataFrame):
21          return X.apply(lambda row:
                self.predict_one(row), axis=1).values
22      else:
23          X_df = pd.DataFrame(X)
24          return X_df.apply(lambda row:
                self.predict_one(row), axis=1).values
```

Wycinek kodu 2: Algorytm predykcji w klasie drzewa decyzyjnego

Ostatnim elementem było nałożenie kolejnej wartstwy abstrakcji poprzez dodanie lasu zawierającego drzewa decyzyjne. W klasie *RandomForest* zaimplementowano metodę fit(), która pozwalała na utworzenie na podstawie danych trenujących zbudować n (hiperparametr) drzew decyzyjnych.

4 Ocena modelu, porównanie z gotowym rozwiązaniem scikit-learn

W celu porównania z utworzonym modelem lasu losowego zastosowano wbudowany model biblioteki scikit-learn *RandomForestClassifier* dla zbioru danych trenujacych. Dzięki temu, możliwa była prosta ocena parametrów, korzystając z interfejsu klasy, czyli jej wbudowanych metod. Wykorzystano następujące metryki klasyfikacji:

- **Dokładność** (accuracy), definiowaną jako udział poprawnych przewidywań w całym zbiorze testowym, obliczana na podstawie zależności (1)

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

,gdzie: TP, TN, FP, FN - TRUE lub FALSE + POSITIVE lub NEGATIVE

- **Precyzja**, czyli miara prawdopowdobieństwa poprawnego przewidywania, obliczana na podstawie zależności (2)

$$precision = \frac{TP}{TP + FP} \tag{3}$$

- **Czułość** (recall), czyli ile faktycznie chorych osób zostało poprawnie wykrytych, obliczana na podstawie zależności (3)

$$recall = \frac{TP}{TP + FN} \tag{4}$$

- **AUC** - pole pod krzywą ROC, obliczane jako pole pod wykresem krzywej ROC (zależności czułości od wskaźnika fałszywych pozytywów).

Wartości metryk wyznaczono dla różnych wielkości zestawu trenującego dla obu modeli, w celu oceny jakości przewidywań. Wyniki przedstawia Tabela 1 dla modelu z biblioteki scikit-learn oraz Tabela 2 dla niestandardowego modelu. W obu wypadkach liczba drzew decyzyjnych jako hiperparametr wynosiła 100.

Tabela 1. Wyniki pomiarów dla różnej wielkości zestawu trenującego - model scikit-learn

Wielkość zestawu trenującego [%]	Dokładnosć [%]	Precyzja [%]	Czułość [%]	AUC
90	85,87	85,96	90,74	0,9418
80	89,13	90,65	90,65	0,9342
70	87,68	90,62	88,41	0,941
60	86,41	91,3	85,52	93,99
50	87,8	90	88,6	0,938
40	87,48	89,2	88,42	0,9377

Dla wybranego rozmiaru zestawu treningowego, stanowiącego 80% całego zbioru danych, przeprowadzono próby optymalizacji modelu lasu losowego poprzez ustalenie liczby drzew jako hiperparametru. Decyzję tę podjęto na podstawie najlepszych uzyskanych wartości metryk oceny modelu.

Wykorzystując klasę *GridSearchCV* zaimplementowano funkcję, badającą najlepszy zestaw hiperparametrów dla algorytmu lasu losowego z scikit-learn. Przedstawiono to w Wycinku kodu 2.

```
1 def optimize_hyperparameters(X_train, y_train):
2     # Utworzenie listy hiperparametrow
3     param_grid = {'n_estimators': [10, 50, 100, 200,
          300, 400, 500]}
4     # Utworzenie automatycznego testera modeli, ktory
          buduje las o hiperparametrze z listy param_grid i
```

Tabela 2. Wyniki pomiarów dla różnej wielkości zestawu trenującego - model niestandardowy

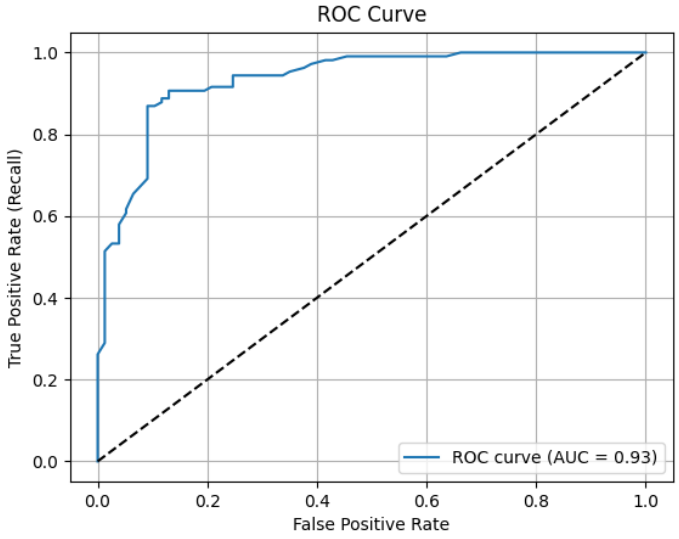
Wielkość zestawu trenującego [%]	Dokładnosć [%]	Precyzja [%]	Czułość [%]	AUC
90	89,13	89,29	92,59	93,86
80	89,13	90,675	90,65	92,93
70	96,96	90,0	87,8	93,56
60	85,6	90,78	84,62	94,02
50	86,27	89,45	86,42	93,58
40	86,75	89,35	87,38	93,25

```
        ocenia na podstawie czulosci modelu
    grid_search =
        GridSearchCV(RandomForestClassifier(random_state=42),
        param_grid, cv=5, scoring='recall')
    # Przeprowadzenie wszystkich prob
    grid_search.fit(X_train, y_train)
    # Wyswietlenie najlepszego znalezionego modelu i
        jego zwrocenie
    print(f"\nNajlepsze parametry:
        {grid_search.best_params_}")
    return grid_search.best_estimator_
```

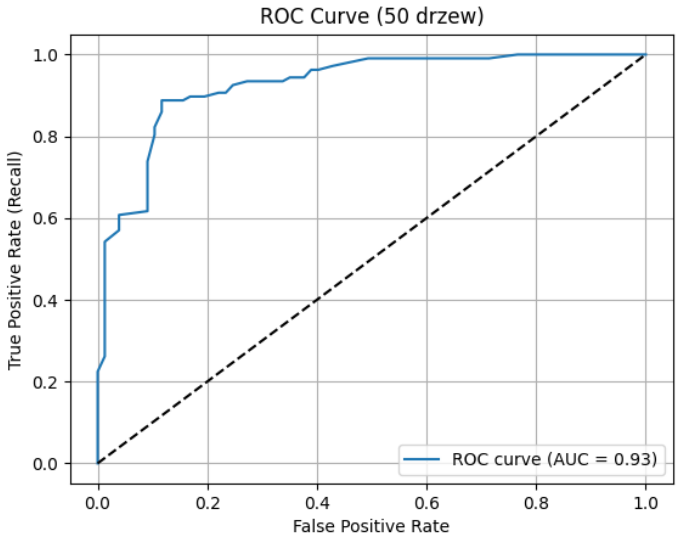
Wycinek kodu 3: Próba optymalizacji przez narzucenie hiperparametru - liczby drzew

Wynikiem testu było znalezienie najlepszej ilości drzew dla uzyskania najlepszego co do wartości wyniku metryki czułości, wynoszącej 100. Dla tego wyniku wykreślono krzywą ROC, przedstawioną na Rys. 1. Dla przykładu zaprezentowano również inny wykres ROC dla nieidealnego modelu na Rys. 2.

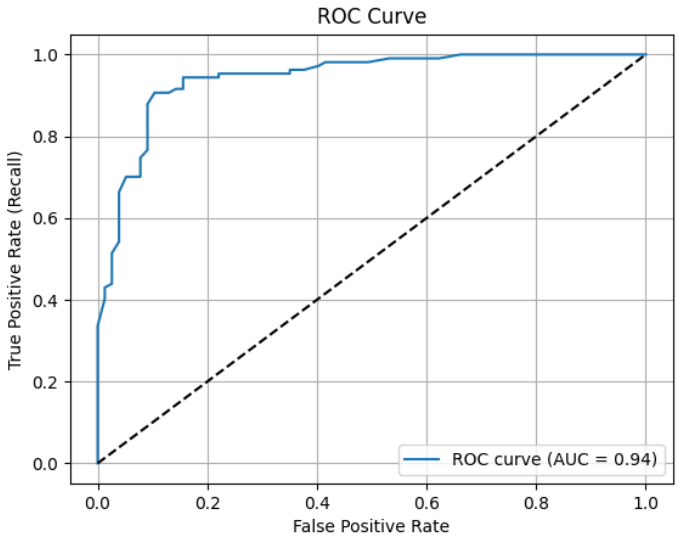
Następnie porównano otrzymane wyniki z otrzymanymi metrykami dla modelu niestandardowego. Wyniki dla kolejnych ilości drzew w lesie przedstawia Tabela 3. Wykres krzywej ROC modelu niestandardowego dla najlepszych wartości metryk oraz dla innych nieoptymalnych przedstawiono na Rys. 3-4. W wypadku modelu niestandardowego również najoptymalniejszym rozwiązaniem okazał się las z 100 drzewami.



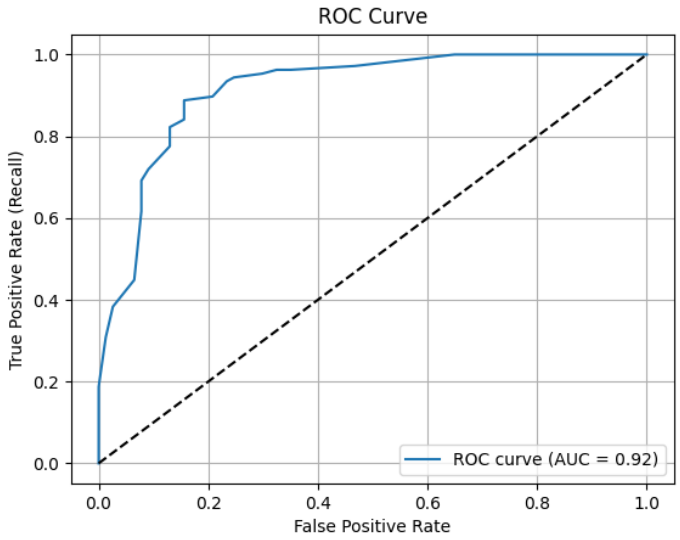
Rys. 1. Wykres ROC dla najlepszego modelu, n = 100



Rys. 2. Wykres ROC dla innego modelu, n = 50



Rys. 3. Wykres ROC dla niestandardowego modelu, n = 100



Rys. 4. Wykres ROC dla niestandardowego modelu, n = 20

5 Wnioski

Algorytm lasu losowego był czuły na zmiany w wielkości zestawu trenującego. Jego dokładność była co do wartości podobna, z najlepszymi wynikami pozostałych metryk dla wielkości zestawu - 80%.

Wpływ na jakość działania algorytmu ma zastosowanie hiperparametru w postaci narzucenia liczby drzew. Na podstawie dokumentacji klasy *RandomForestClassifier* wiadomo, że domyślnie liczba ta wynosi 100. Testy przeprowadzone dla kolenych liczb drzew wykazały, że była to optymalna ilość, jeżeli oceniamy modele przez przymat ich czułości.

Wybór kryterium oceny modeli nie był przypadkowy. Czułość z punktu widzenia medycyny powinna być najistotniejszym parametrem do maksymalizacji w modelu klasyfikującym. Jest tak, ponieważ daje ona informację ile spośród osób chorych zostało wykrytych w procesie predykcji. Tym samym uzyskuje się najlepsze pojęcie o szansie na poprawne identyfikowanie chorego przez model.