Ćwiczenie DD

Drzewa decyzyjne

Część teoretyczna

Wykład na temat drzew decyzyjnych.

Zadania pomocnicze

Ćwiczenie wykorzystuje algorytmy drzew decyzyjnych opisane w rozdz. 8.3 książki G. James, D. Witten, T. Hastie, R. Tibshirani, J. Taylor: An Introduction to Statistical Learning with Applications in Python (https://www.statlearning.com/). Zapoznaj się z tym materiałem.

Zadania do wykonania

Według Światowej Organizacji Zdrowia (WHO) udar jest drugą najczęstszą przyczyną zgonów na świecie i odpowiada za około 11% wszystkich zgonów.
Zadanie polega na budowie klasyfikatora opartego na drzewach decyzyjnych do przewidywania, czy pacjent może doznać udaru na podstawie parametrów wejściowych, takich jak płeć, wiek, schorzenia i przebyte choroby oraz palenie tytoniu. Każdy wiersz danych zawiera istotne informacje o pacjencie.

1. Zaimportuj niezbędne moduły:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.model_selection as skm

from matplotlib.pyplot import subplots
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import PredictionErrorDisplay
from imblearn.over_sampling import RandomOverSampler
from sklearn.tree import (DecisionTreeClassifier as DTC,
                          DecisionTreeRegressor as DTR,
                          plot_tree,
                          export_text)
from sklearn.metrics import (accuracy_score,
                             confusion_matrix,
                             ConfusionMatrixDisplay,
                             mean_squared_error)
from sklearn.ensemble import \
     (RandomForestRegressor as RF,
      GradientBoostingRegressor as GBR)
```

2. Wczytaj dane, dokonaj selekcji danych (każdy student tworzy inny zbiór danych, który składa się z wylosowanych wierszy danych oryginalnych).

   Pobierz dane z https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

```
data = pd.read_csv("healthcare-dataset-stroke-data.csv")
```

```
nr_gr = ?
r_k = ?
np.random.seed(int(nr_gr*r_k))

q = data.shape
idx = np.random.permutation(q[0]) #permutacja wierszy
data = data.iloc[idx[0:5000],:] #selekcja 5000 losowych wierszy

data.head()
```

gdzie za `nr_gr` wstaw numer swojej sekcji a za `r_k` aktualny rok kalendarzowy.

3. Scharakteryzuj dane (przydatne funkcje: `data.describe()`, `data.info()`).

4. Oczyszczanie danych.

   - Usuń nieprzydatną kolumnę 'id' (`data=data.drop('id', axis = 1)`)
   - Znajdź i ew. usuń zduplikowane wiersze (`data.duplicated().sum()`)
   - Znajdź atrybuty z brakującymi wartościami (`data.isnull().sum()`)
   - Usuń wiersze z brakującymi wartościami (`data = data.dropna(axis = 0)`)

5. Implementacja drzew decyzyjnych w bibliotece `sklearn` nie akceptuje atrybutów nominalnych. Takie atrybuty należy zamienić na liczbowe (`int64`). W tym celu:

   - Utwórz nową tabelę z danymi (`data2 = pd.DataFrame()`)
   - Przepisz do tej tabeli wszystkie atrybuty liczbowe (np. `data2['age'] = data['age']`)
   - Jeśli atrybut nominalny ma tylko dwie wartości (możesz to sprawdzić funkcją `data.nunique()`, `data.unique()`), zamień te wartości na 0 i 1, np.: `data2['ever_married'] = np.int64(data['ever_married']=='Yes')`
   - Jeśli atrybut nominalny ma *k*>2 wartości, utwórz dla niego *k* nowych atrybutów, np.:

```
data2['work_type_Private'] = np.int64(data['work_type']=='Private')
data2['work_type_Self_emp'] = np.int64(data['work_type']=='Self-employed')
data2['work_type_Govt_job'] = np.int64(data['work_type']=='Govt_job')
data2['work_type_Children'] = np.int64(data['work_type']=='children')
data2['work_type_Never_worked'] = np.int64(data['work_type']=='Never_worked')
```

6. Przygotuj dane do uczenia modelu. Utwórz tabele:
   - `X` z przykładami (zawierającą wszystkie kolumny z wyjątkiem 'stroke') i
   - `y` z etykietami (kolumna 'stroke').

   Podziel dane na część treningową (80% danych) i testową (20%). Przeprowadź próbne uczenie modelu.

```
X = data2.iloc[:, : -1]
y = data2.iloc[:, -1 :]
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.2,
random_state = 1)
```

```
dt = DTC(criterion='entropy', max_depth=5, random_state=1)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

accuracy_score(y_test, y_pred)
```

7. W procedurze kroswalidacji znajdź optymalną wartość `max_depth` (wzoruj się na przykładzie podanym na wykładzie – str. 69). Pokaż wykres zależności dokładności klasyfikacji od `max_depth`. Odnotuj dokładność modelu wyznaczoną w kroswalidacji.

8. Przeprowadź uczenie drzewa na zbiorze treningowym z optymalną wartością hiperparametru `max_depth` znalezioną w p. 7. Oblicz dokładność klasyfikacji dla danych treningowych i testowych. Narysuj drzewo i przedstaw je w postaci reguł decyzyjnych:

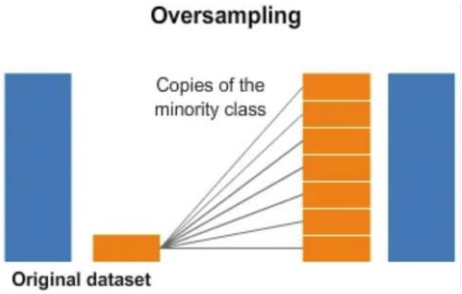
```
ax = subplots(figsize=(8,8))[1]
plot_tree(dt,
    feature_names=X.columns,
    ax=ax);

print(export_text(dt,
    feature_names=list(X.columns),
    show_weights=True))
```

9. Zwizualizuj wyniki za pomocą `confusion_matrix`.

```
cm = confusion_matrix(y_test, dt.predict(X_test), labels=dt.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=dt.classes_)
disp.plot()
plt.show()
```

Zwróć uwagę, że przykłady z klasy 1 są w większości błędnie klasyfikowane. Pomimo tego dokładność klasyfikatora jest bardzo wysoka. Wynika to z dysproporcji liczebności klas – liczebność klasy 0 przewyższa znacznie liczebność klasy 1. Jest to tzw. problem niezbalansowanych danych (*imbalanced data*). Aby zwiększyć dokładność klasyfikacji dla klasy mniej licznej, rozwiązaniem może być „rozmnożenie" danych treningowych dla tej klasy (*oversampling*).




```
ros = RandomOverSampler(random_state = 1)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)
```

10. Powtórz p. 7-9 dla danych „rozmnożonych". Czy osiągnięto poprawę wyników? Które atrybuty są najbardziej istotne (występują w początkowych węzłach)?

11. Używając drzewa regresyjnego dokonaj predykcji wskaźnika `'bmi'` na podstawie pozostałych atrybutów (wliczając atrybut `'stroke'`). W tym celu wykonaj ponownie p. 6-8, odpowiednio je modyfikując.

    Wskazówki:

    - Zamiast `DTC` użyj `DTR` z `criterion='squared_error'`
    - Do oceny modelu zamiast funkcji `accuracy_score()` użyj `mean_squared_error()`
    - W p. 7 do kroswalidacji użyj następującego wywołania funkcji `cross_val_score`:

```
cw[i]= -cross_val_score(dt,
                        X_train2, y_train2,
                        cv=10,
                        scoring='neg_mean_squared_error').mean()
```

    - Pamiętaj, że w kroswalidacji szukamy teraz `max_depth`, które minimalizuje błąd, a nie tak jak w klasyfikacji maksymalizuje dokładność.

    Dokonaj wizualizacji używając kodu zamieszczonego poniżej i oceń wyniki.

```
ax = subplots(figsize=(6,6))[1]
PredictionErrorDisplay.from_predictions(
    y_true=y_test2,
    y_pred=dt.predict(X_test2),
    #y_true=y_train2,
    #y_pred=dt.predict(X_train2),
    kind="actual_vs_predicted",
    ax=ax,
    random_state=0,
)
```

12. Użyj modelu lasu losowego (`RandomForestRegressor`) do rozwiązania problemu regresyjnego postawionego w p. 11. Przyjmij optymalną wartość `max_depth` z p. 11 i domyślne wartości pozostałych hiperparametrów z wyjątkiem `max_features` (liczba atrybutów do przeprowadzenia testu; oznaczony przez m na wykładzie, str. 128 i 129). W kroswalidacji znajdź optymalną wartość hiperparametru `max_features`.

13. Użyj modelu boostingowego opartego na drzewach regresyjnych (`GradientBoostingRegressor`) do rozwiązania problemu regresyjnego postawionego w p. 11. Przyjmij optymalną wartość `max_depth` z p. 11 i domyślne wartości pozostałych hiperparametrów z wyjątkiem `learning_rate` (odpowiednik parametru zawężania λ, patrz wykład str. 130-132). W kroswalidacji znajdź optymalną wartość hiperparametru `learning_rate` w zakresie od 0 do 0.1. Porównaj wyniki otrzymane w p. 11-13.

Co powinno znaleźć się w sprawozdaniu

A) Cel ćwiczenia.
B) Treść zadania.
C) Opis używanych w ćwiczeniu metod opartych na drzewach decyzyjnych (nie kopiuj treści wykładu, poszukaj w literaturze i Internecie).
D) Metodyka rozwiązania – poszczególne instrukcje z wynikami i komentarzem (zachowaj numerację zadań).
E) Wnioski końcowe.

Zadania dodatkowe dla ambitnych

1. Dokonaj optymalizacji modeli, dobierając jednocześnie wartości kilku hiperparametrów w procedurze *grid search*.
2. Dokonaj estymacji istotności atrybutów w modelu lasów losowych za pomocą dwóch metod: analizy zmniejszenia zanieczyszczenia węzłów (*feature importance based on mean decrease in impurity*) oraz metodą permutacji wartości atrybutów (*feature importance based on feature permutation*).
3. Użyj modelu zespołowego BART (*Bayesian Additive Regression Trees*) do rozwiązania problemu regresyjnego postawionego w p. 11. Porównaj z wynikami otrzymanymi w p. 11-13.
4. Wykonaj to ćwiczenie w innym środowisku, np. R, Matlab, ...

Przykładowe zagadnienia i pytania zaliczeniowe

1. Cel i plan ćwiczenia.
2. Materiał ze sprawozdania.
3. Tworzenie drzewa klasyfikacyjnego i regresyjnego.
4. Kryterium stopu i ustalenie etykiety.
5. Rodzaje testów.
6. Kryteria wyboru testów.
7. Ustalenie zbioru testów kandydujących.
8. Przycinanie drzewa.
9. Lasy losowe.
10. Metoda boostingowa oparta na drzewach decyzyjnych.

Do przygotowania na następne zajęcia

1. Zapoznać się z instrukcją do kolejnego ćwiczenia.
2. Zapoznać się z częścią teoretyczną do kolejnego ćwiczenia.
3. Wykonać zadania pomocnicze do kolejnego ćwiczenia.