Inżynieria_cech_data_split

April 15, 2025

Inżynieria cech - data split

Dominik Sakłaski, 415120

Załadowanie bibliotek

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

1. Wczytaj końcowy i przetworzony zbiór danych Titanic z poprzednich zajęć.

[2]:
```
data = pd.read_csv('titanic_data.csv')
data.head(10)
```

[2]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest	CabinReduced
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO	B
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN		
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN		
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0		
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN		
5	1	1	Anderson, Mr. Harry	male	48.0000	0	0	19952	26.5500	E12	S	3	NaN		
6	1	1	Andrews, Miss. Kornelia Theodosia	female	63.0000	1	0	13502	77.9583	D7	S	10	NaN		
7	1	0	Andrews, Mr. Thomas Jr	male	39.0000	0	0	112050	0.0000	A36	S	NaN	NaN		
8	1	1	Appleton, Mrs. Edward Dale (Charlotte Lamson)	female	53.0000	2	0	11769	51.4792	C101	S	D	NaN		
9	1	0	Artagaveytia, Mr. Ramon	male	71.0000	0	0	PC 17609	49.5042	NaN	C	NaN	22.0		

```
1  Montreal, PQ / Chesterville, ON          C
2  Montreal, PQ / Chesterville, ON          C
3  Montreal, PQ / Chesterville, ON          C
4  Montreal, PQ / Chesterville, ON          C
5                    New York, NY           E
6                      Hudson, NY           D
7                     Belfast, NI           A
8             Bayside, Queens, NY           C
9             Montevideo, Uruguay           n
```

2. Zapoznaj się z funkcja train_test_split wchodzącą w skład biblioteki Scikit-learn. Zapisz swoje obserwacje. Funkcja `train_test_split` pochodzi z biblioteki sklearn.model_selection. Umożliwia łatwe podzielenie zbioru danych na część `treningową` i `testową`.

- **test_size**: określa procent danych, który ma trafić do zbioru testowego (np. test_size=0.3 oznacza, że 30% danych pójdzie do testu, a 70% do treningu.)
- **random_state**: ustala „ziarno" losowania – żeby podział był zawsze taki sam.
- **shuffle**: określa, czy dane mają być przetasowane przed podziałem; ustawienie shuffle=False może być przydatne np. w analizach czasowych, gdzie kolejność ma znaczenie.
- **stratify**: można podać kolumnę, aby zachować proporcje klas w zbiorach; ustawiając stratify=y, zapewnia się, by podział odzwierciedlał rozkład klas w oryginalnych danych; stosujemy, gdy: mamy dane klasyfikacyjne (survived: 0-zginął, 1-przeżył), klasy są niezbalansowane (jedna klasa występuje częściej niż druga,np. 80% vs 10%), zależy nam na uczciwym i reprezentatywnym podziale danych.

3. Stwórz zmienną do której zapiszesz listę z nazwami trzech kolumn – kabiny, zredukowane kabiny oraz płeć. Nazwij ją col_name.

[3]:
```
col_name = ['cabin', 'CabinReduced', 'sex']
```

4. Podziel zbiór na treningowy i testowy używając `train_test_split`. Jako zmienną niezależną ustaw dane składające się z kolumn o nazwach zapisanych w col_name. Jako zmienną zależną ustaw kolumnę mówiącą o tym czy ktoś przeżył czy nie. Ustaw rozmiar zbioru testowego na 20 lub 30% całości. Parametr random_state ustaw jako 0.

Wyświetl wymiary zbiorów X_train, X_test, y_train, y_test używając .shape() i skomentuj wyniki.

[4]:
```
#zmienna niezależna: wybrane kolumny (cabina, cabinReduced, sex)
X = data[col_name]
#zmienna zależna : info czy pasażer przeżył
y = data['survived']

X_train, X_test, y_train, y_test = train_test_split(
    X, y,                    # dane wejściowe i etykiety
    test_size=0.3,          # 30% danych trafi do zbioru testowego
```

```
    random_state=0          # ziarno
)

# .shape służy do sprawdzenia wymiarów (kształtu) zbioru danych
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (916, 3)
X_test shape: (393, 3)
y_train shape: (916,)
y_test shape: (393,)
```

Zbiór danych został skutecznie podzielony na zbiór treningowy (70%) i testowy (30%) zgodnie z parametrem test_size=0.3. W zbiorze treningowym znajduje się 916 rekordów, a w testowym – 393. Każdy rekord posiada 3 cechy wejściowe: cabin, CabinReduced oraz sex.

- X_train shape: (916, 3) – zbiór treningowy zawiera 916 próbek (wierszy) i 3 cechy (kolumny)
- X_test shape: (393, 3) – zbiór testowy zawiera 393 próbki (30% danych), również z 3 cechami.
- y_train shape: (916,) – zmienna zależna (czy ktoś przeżył) dla zbioru treningowego ma 916 wartości.
- y_test shape: (393,) – zmienna zależna dla zbioru testowego ma 393 wartości.

5. Zbadaj w zależności od kardynalności danej zmiennej czy rozkład etykiet dla poszczególnych cech w zbiorach testowych i treningowych jest równomierny. Unique_test = [x for x in X_test.ZMIENNA.unique() if x not in X_train.ZMIENNA.unique()]

len(unique)

[5]:
```
for col in col_name:
    unique_test = [x for x in X_test[col].unique() if x not in X_train[col].
 ↪unique()]
    print(f"Zmienna: {col}")
    print(f"Liczba unikalnych etykiet tylko w zbiorze testowym:␣
 ↪{len(unique_test)}")
    print(f"Etykiety: {unique_test}\n")
```

```
Zmienna: cabin
Liczba unikalnych etykiet tylko w zbiorze testowym: 37
Etykiety: [nan, 'E12', 'C104', 'A31', 'D11', 'D48', 'D10 D12', 'B38', 'D45',
'C50', 'C31', 'B82 B84', 'A32', 'C53', 'B10', 'C70', 'A23', 'C106', 'C46',
'E58', 'B11', 'F E69', 'B80', 'E39 E41', 'D22', 'E40', 'A19', 'C32', 'B79',
'C45', 'B22', 'B39', 'C47', 'B101', 'A7', 'E52', 'F38']

Zmienna: CabinReduced
Liczba unikalnych etykiet tylko w zbiorze testowym: 0
Etykiety: []
```

```
Zmienna: sex
Liczba unikalnych etykiet tylko w zbiorze testowym: 0
Etykiety: []
```

Zmienna cabin:

- W testowym zbiorze znajduje się 37 etykiet, które nie występują w treningowym. To pokazuje, że rozkład nie jest równomierny – duża kardynalność (187 unikalnych wartości) prowadzi do tego, że model w testowym zobaczy zupełnie nowe etykiety.

Zmienne CabinReduced i sex:

- Wynik jest równy 0, czyli wszystkie etykiety występujące w zbiorze testowym były już wcześniej w zbiorze treningowym. Jest to wynikiem niskiej kardynalności tych etykiet - to oznacza, że rozkład jest równomierny.

Uwaga! PUNKTY 6 I 7 należy wykonać osobno dla zmiennej kabina, kabina zredukowana, płeć. Wszystkie wyniki mają zostać zapisane zarówno w zbiorze testowym jak i treningowym.

6. Wykonaj kodowanie zmiennych kategorycznych do zmiennych liczbowych. Wykorzystaj pętlę for i metodę enumerate(). Wynik powinien być podobny do tego na obrazku poniżej Przykład:

'E24' : 3,

'C22' : 4,

...

'B45' : 13

[6]:
```
mappings = {}  # słowniki dla wszystkich zmiennych

for col in ['cabin', 'CabinReduced', 'sex']:
    labels = X_train[col].dropna().unique()  # unikalne etykiety bez NaN

    # Dodatkowe czyszczenie dla CabinReduced
    if col == 'CabinReduced':
        labels = [val for val in labels if val != 'n']

    mapping = {label: idx + 1 for idx, label in enumerate(labels)}  # tworzenie␣
 ↪słownika
    mappings[col] = mapping  # zapisz słownik do mapowania

    # Wyświetlenie słownika (ograniczenie do 10 wpisów)
    print(f"\nSłownik dla zmiennej '{col}':")
    for k, v in list(mapping.items())[:10]:
```

```
        print(f"'{k}' : {v}")
```

```
Słownik dla zmiennej 'cabin':
'E36' : 1
'C68' : 2
'E24' : 3
'C22 C26' : 4
'D38' : 5
'B50' : 6
'A24' : 7
'C111' : 8
'F' : 9
'C6' : 10

Słownik dla zmiennej 'CabinReduced':
'E' : 1
'C' : 2
'D' : 3
'B' : 4
'A' : 5
'F' : 6
'T' : 7
'G' : 8

Słownik dla zmiennej 'sex':
'female' : 1
'male' : 2
```

7. Zastąp etykiety zmiennej (tu przykład dla kabina) słownikiem stworzonym w kroku 6. Do tego będzie potrzebne mapowanie. Przykładowy wynik dla zmiennej Kabina:

	Kabina_map	Kabina
234	0	NaN
345	0	NaN
344	13	B45
..	..	..

[7]:
```
for col in ['cabin', 'CabinReduced', 'sex']:
    map_col = f"{col}_map"
    mapping = mappings[col]  # pobierz słownik z kroku 6

    # Zastosuj mapowanie do zbioru treningowego i testowego
    X_train[map_col] = X_train[col].map(mapping).astype('Int64')
    X_test[map_col] = X_test[col].map(mapping).astype('Int64')
```

```
    print(f"\nZmienna: {col} (zbiór treningowy):")
    print(X_train[[map_col, col]].head())

    print(f"\nZmienna: {col} (zbiór testowy):")
    print(X_test[[map_col, col]].head())
```

```
Zmienna: cabin (zbiór treningowy):
      cabin_map cabin
501        <NA>   NaN
588        <NA>   NaN
402        <NA>   NaN
1193       <NA>   NaN
686        <NA>   NaN

Zmienna: cabin (zbiór testowy):
      cabin_map cabin
1139       <NA>   NaN
533        <NA>   NaN
459        <NA>   NaN
1150       <NA>   NaN
393        <NA>   NaN

Zmienna: CabinReduced (zbiór treningowy):
      CabinReduced_map CabinReduced
501               <NA>            n
588               <NA>            n
402               <NA>            n
1193              <NA>            n
686               <NA>            n

Zmienna: CabinReduced (zbiór testowy):
      CabinReduced_map CabinReduced
1139              <NA>            n
533               <NA>            n
459               <NA>            n
1150              <NA>            n
393               <NA>            n

Zmienna: sex (zbiór treningowy):
      sex_map     sex
501         1  female
588         1  female
402         1  female
1193        2    male
686         1  female

Zmienna: sex (zbiór testowy):
```

```
      sex_map     sex
1139        2    male
533         1  female
459         2    male
1150        2    male
393         2    male
```

Mapowanie: To przypisanie jednej wartości do innej — najczęściej używane przy zamianie wartości tekstowych (kategorycznych) na wartości liczbowe, które można łatwo przetwarzać matematycznie.

Dlaczego to robimy? Zalety?

- Modele uczenia maszynowego nie potrafią analizować danych typu tekst (np. „female", „male").
- Potrzebujemy reprezentacji liczbowej.
- Mapowanie jest łatwe, szybkie i skuteczne – tworzy nową kolumnę, bez zmieniania oryginalnej.
- Przekształca dane do postaci zrozumiałej dla algorytmów ML.
- Pozwala zachować porządek i spójność (np. jedna wartość zawsze odpowiada jednej liczbie).

8. Sprawdź liczbę brakujących wartości w zmodyfikowanych zbiorach. Zapisz wyniki i skomentuj.

[8]:
```
print("Braki danych w X_train:\n", X_train.isnull().sum())
print("\nBraki danych w X_test:\n", X_test.isnull().sum())
```

```
Braki danych w X_train:
 cabin               702
CabinReduced           0
sex                    0
cabin_map            702
CabinReduced_map     702
sex_map                0
dtype: int64

Braki danych w X_test:
 cabin               312
CabinReduced           0
sex                    0
cabin_map            354
CabinReduced_map     312
sex_map                0
dtype: int64
```

- W kolumnie cabin nadal występuje 702 braków danych w zbiorze treningowym oraz 312 w zbiorze testowym. Jest to oryginalna cecha, która nie została przekształcona – zawiera wartości typu NaN pochodzące bezpośrednio z danych wejściowych.
- W kolumnie cabin_map występuje 702 braków w X_train oraz 354 w X_test. Oznacza to, że część etykiet w kolumnie cabin nie miała swojego odpowiednika w słowniku mapującym lub była NaN. W X_test mogą pojawiać się etykiety, które nie były obecne w X_train, dlatego

nie zostały odwzorowane.

- W kolumnie CabinReduced braków nie ma (0 w obu zbiorach), ponieważ wszystkie wartości zostały przekształcone do pojedynczych liter lub wypełnione.
- W kolumnie CabinReduced_map pojawia się 702 braków w X_train i 312 w X_test, co sugeruje, że mimo iż CabinReduced nie zawiera już NaN, to w wyniku mapowania niektóre wartości nie zostały odwzorowane. To może oznaczać, że w danych znalazły się nowe etykiety ('n', np. po konwersji NaN do 'n'), które nie zostały ujęte w słowniku.
- W kolumnach sex i sex_map braków nie ma – dane są kompletne, a etykiety tekstowe zostały prawidłowo zamienione na liczby (1, 2) zarówno w zbiorze treningowym, jak i testowym.

Mapowanie nie eliminuje braków danych – jeśli oryginalna zmienna zawierała NaN lub nieznane etykiety (np. nowe wartości w zbiorze testowym), to efekt mapowania również będzie zawierał NaN.

9. Zastąp brakujące wartości liczbą 0. Czy jest to najlepsze wyjście?

[9]:
```
X_train_filled = X_train.fillna(0)
X_test_filled = X_test.fillna(0)

print("Braki w X_train po uzupełnieniu:\n", X_train_filled
      .isnull().sum())
print("\nBraki w X_test po uzupełnieniu:\n", X_test_filled
      .isnull().sum())
```

```
Braki w X_train po uzupełnieniu:
 cabin               0
CabinReduced         0
sex                  0
cabin_map            0
CabinReduced_map     0
sex_map              0
dtype: int64

Braki w X_test po uzupełnieniu:
 cabin               0
CabinReduced         0
sex                  0
cabin_map            0
CabinReduced_map     0
sex_map              0
dtype: int64
```

Zastąpienie braków 0 jest dobrym rozwiązaniem tymczasowym lub gdy brak danych sam w sobie niesie informację.

Zalety:

- Pozwala natychmiast użyć danych w modelach ML, które nie akceptują NaN.
- W przypadku danych zakodowanych liczbowo (np. cabin_map, sex_map) wartość 0 może symbolizować brak danych.

Wady:

- Może wprowadzać szum do modelu – 0 może być zinterpretowane jako „prawdziwa" wartość, co może wpłynąć na jakość predykcji.
- W przypadku zmiennych liczbowych (np. wiek, cena) lepszym podejściem może być użycie średniej, mediany lub imputacji modelowej.
- W przypadku cech kategorycznych – warto rozważyć oddzielną etykietę, np. „Brak informacji" (missing, Unknown).

10. Porównaj ile unikalnych wartości jest w zbiorze treningowym, a ile w zbiorze testowym (funkcja len). Jaka jest różnica pomiędzy liczbą etykiet przed i po redukcji oraz mapowaniu? Czy cały proces, który został do tej pory wykonany może mieć wpływ na końcowy wynik predykcji i jakość modelu?

[10]:
```
# dla kolumny 'cabin'
print("Unikalne wartości cabin w X_train:", len(X_train['cabin']
                                                .unique()))
print("Unikalne wartości cabin w X_test:", len(X_test['cabin']
                                               .unique()))

# dla kolumny 'CabinReduced'
print("Unikalne wartości CabinReduced w X_train:", len(X_train['CabinReduced']
                                                       .unique()))
print("Unikalne wartości CabinReduced w X_test:", len(X_test['CabinReduced']
                                                      .unique()))

# dla kolumny 'sex'
print("Unikalne wartości sex w X_train:", len(X_train['sex']
                                              .unique()))
print("Unikalne wartości sex w X_test:", len(X_test['sex']
                                             .unique()))
```

```
Unikalne wartości cabin w X_train: 151
Unikalne wartości cabin w X_test: 71
Unikalne wartości CabinReduced w X_train: 9
Unikalne wartości CabinReduced w X_test: 8
Unikalne wartości sex w X_train: 2
Unikalne wartości sex w X_test: 2
```

Zmienna cabin: - Wysoka kardynalność: 151 w treningowym i 71 w testowym - co oznacza bardzo dużą liczbę różnych wartości. Taka zmienność może prowadzić do problemów w modelu (przeuczenie).

Zmienna CabinReduced: - Po redukcji do pierwszej litery liczba unikalnych etykiet drastycznie spadła kolejno do 9 i 8. - Taka redukcja upraszcza model i ogranicza ryzyko, że w zbiorze testowym pojawi się nieznana wcześniej wartość. - Pomimo uproszczenia zachowana zostaje część informacji (np. o lokalizacji kabiny na statku).

Zmienna sex: - Jest idealnie zbalansowana i prosta – tylko dwie unikalne etykiety (male, female) w obu zbiorach. - Takie zmienne są bardzo łatwe do zakodowania i nie powodują problemów z generalizacją.

Różnica między liczbą etykiet przed i po redukcji oraz mapowaniu polega na zmniejszeniu kardynalności cech kategorycznych, co wpływa na prostotę, zrozumiałość oraz efektywność analizy danych i modelowania.

- Przed redukcją:

`X_train`: 151 unikalnych etykiet

`X_test`: 71 unikalnych etykiet

- Po redukcji do CabinReduced:

`X_train`: 9 unikalnych etykiet

`X_test`: 8 unikalnych etykiet

Różnica: zredukowano z 151 do 9 (w X_train), co oznacza spadek o ponad 94%

- Po mapowaniu:

`CabinReduced_map`: zawiera te same wartości co CabinReduced, ale zakodowane jako liczby całkowite. Liczba unikalnych etykiet nie uległa zmianie, zmieniła się tylko ich forma zapisu (np. 'C', 'B', 'D' → 1, 2, 3).

`sex_map`: zamienia 'female' i 'male' na wartości liczbowe (np. 1 i 2), nie zmieniając liczby kategorii – nadal są to tylko 2 unikalne etykiety.

`cabin_map`: również została zakodowana na liczby całkowite. Liczba etykiet pozostała taka sama jak przed kodowaniem (czyli bardzo wysoka) – nie wykonano redukcji, a jedynie przekształcono tekstowe etykiety kabin (np. 'C85', 'D36', 'B28') na liczby (np. 1, 2, 3). Kardynalność nadal pozostaje wysoka, dlatego CabinReduced jest preferowaną wersją do dalszych analiz.

Tak, cały dotychczasowy proces ma istotny wpływ na końcowy wynik predykcji i jakość modelu.

- **Redukcja kardynalności cech** - Redukcja liczby kategorii (np. do samej litery kabiny) upraszcza model i zmniejsza szum w danych, co zwykle poprawia stabilność i jakość predykcji.
- **Mapowanie (kodowanie) zmiennych kategorycznych** - Zamiana etykiet tekstowych na liczby całkowite umożliwia ich użycie w algorytmach uczenia maszynowego, które wymagają danych liczbowych. Jest niezbędnym krokiem przygotowującym dane do modelowania.
- **Zastępowanie wartości brakujących** - Umożliwia dalsze przetwarzanie danych i trenowanie modelu bez błędów technicznych; choć zastąpienie zerem nie zawsze jest idealne, to pozwala zachować spójność i kompletność zbiorów danych.
- **Podział na zbiory treningowe i testowe** - Zapewnia możliwość oceny jakości modelu na danych niewidzianych wcześniej, co pozwala sprawdzić, czy model dobrze się generalizuje, a nie tylko uczy „na pamięć".