SIECI NEURONOWE

Raport z laboratoriów 1-2

Student: Van Hien Le - 257795

Narzędzia i technologie

- Python, Pandas, Numpy, Scikit-learn
- Jupyter Notebook

Dataset

Zbiór danych używany w tym ćwiczeniu to zbiór danych chorób serca, do którego można uzyskać dostęp pod adresem https://archive.ics.uci.edu/dataset/45/heart+disease.

Analiza eksploracyjna

1. **Import danych**

   Instalacja bibliotek i pobieranie zbioru danych:

```
!pip install ucimlrepo
!pip install scikit-learn
```

```
from ucimlrepo import fetch_ucirepo, list_available_datasets
list_available_datasets()
```

```
heart_disease = fetch_ucirepo(id=45)
X = heart_disease.data.features
y = heart_disease.data.targets
```

   Pierwsze 10 przykładów zbioru danych:

```
dataset = pd.concat([X, y], axis=1)
dataset[0:10]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
0	63	1	1	145	233	1	2	150	0	2.3	3	0.0	6.0	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3.0	3.0	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2.0	7.0	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0.0	3.0	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0.0	3.0	0
5	56	1	2	120	236	0	0	178	0	0.8	1	0.0	3.0	0
6	62	0	4	140	268	0	2	160	0	3.6	3	2.0	3.0	3
7	57	0	4	120	354	0	0	163	1	0.6	1	0.0	3.0	0
8	63	1	4	130	254	0	2	147	0	1.4	2	1.0	7.0	2
9	53	1	4	140	203	1	2	155	1	3.1	3	0.0	7.0	1

   Kształt zbioru danych:

```
dataset.shape
```

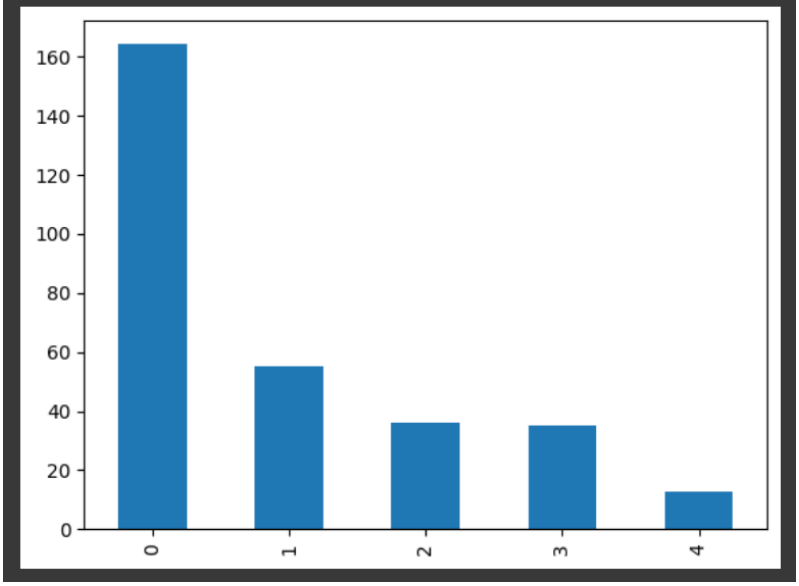
```
(303, 14)
```

2. Balans klas

- Czy zbiór jest zbalansowany pod względem liczby próbek na klasy?

```
dataset['num'].value_counts().plot.bar()
```

```
<Axes: >
```



Zbiór danych nie jest zbalansowany pod względem liczby próbek na klasę. Liczba próbek klasy 0 jest znacznie wyższa niż pozostałych.

3. Cechy liczbowe

- Jakie są średnie i odchylenia cech liczbowych?

  Cechy liczbowe w zbiorze danych to "age", "trestbps", "chol", "thalach" i "oldpeak".

  Średnie i odchylenia:

```
num_features = ['age', 'trestbps', 'chol', 'thalach', 'oldpeak']
dataset[num_features].describe().loc[['mean', 'std']]
```

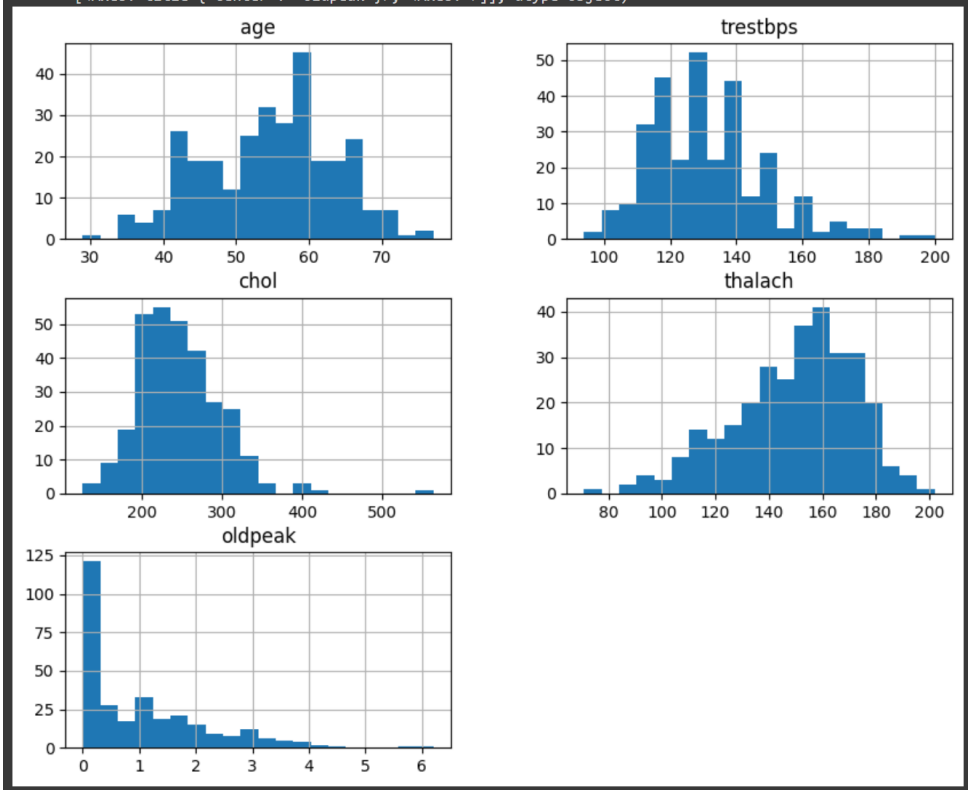
	age	trestbps	chol	thalach	oldpeak
mean	54.438944	131.689769	246.693069	149.607261	1.039604
std	9.038662	17.599748	51.776918	22.875003	1.161075

- Dla cech liczbowych: czy ich rozkład jest w przybliżeniu normalny?

  Histogram cech liczbowych:

```
dataset[num_features].hist(figsize=(10, 8), bins=20)
```

```
array([[<Axes: title={'center': 'age'}>,
        <Axes: title={'center': 'trestbps'}>],
       [<Axes: title={'center': 'chol'}>,
        <Axes: title={'center': 'thalach'}>],
       [<Axes: title={'center': 'oldpeak'}>, <Axes: >]], dtype=object)
```



Patrząc na histogramy, możemy rozpoznać, że rozkład normalny istnieje w cechach "age", "chol" i "thalach", ale wydaje się, że nie istnieje w cechach "trestbps" i "oldpeak".

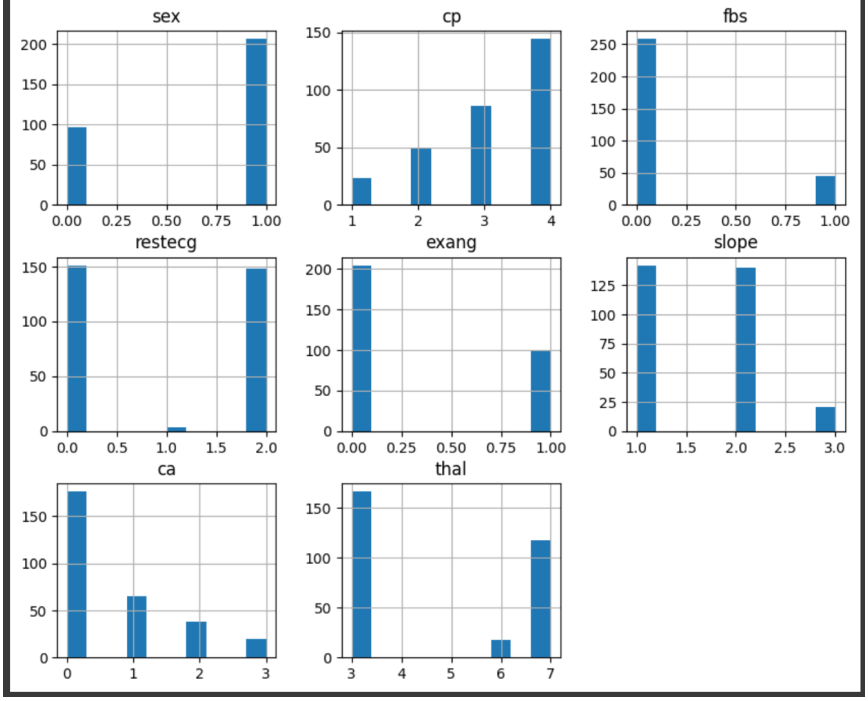
4. Cechy kategoryczne

- Dla cech kategorycznych: czy rozkład jest w przybliżeniu równomierny?
  Cechy kategoryczne w zbiorze danych to "sex", "cp", "fbs", "restecg", "exang", "slope", "ca" i "thal".

  Histogram cech kategorycznych:

```
cat_features = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal']
dataset[cat_features].hist(figsize=(10, 8))
```

```
array([[<Axes: title={'center': 'sex'}>, <Axes: title={'center': 'cp'}>,
        <Axes: title={'center': 'fbs'}>],
       [<Axes: title={'center': 'restecg'}>,
        <Axes: title={'center': 'exang'}>,
        <Axes: title={'center': 'slope'}>],
       [<Axes: title={'center': 'ca'}>, <Axes: title={'center': 'thal'}>,
        <Axes: >]], dtype=object)
```



Patrząc na histogramy, możemy zauważyć, że cechy kategoryczne tego zbioru danych nie mają równomiernego rozkładu.

5. Brakujące dane

- Czy w zbiorze danych brakuje jakichś danych?

```
dataset.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          4
thal        2
num         0
dtype: int64
```

W zbiorze danych brakuje 6 wartości, 4 w kolumnie "ca" i 2 w kolumnie "thal".

- Jaką strategię można zastosować, aby zastąpić brakujące dane?

  Dla cechy numerycznej "ca" możemy zastąpić brakujące dane wartościami mediany. Dla cechy kategorycznej 'thal' możemy zastąpić brakujące wartości najczęściej występującą wartością tej cechy:

```
ca_median = dataset['ca'].median()
ca_median
```

```
0.0
```

```
thal_mode = dataset['thal'].mode()[0]
thal_mode
```

```
3.0
```

```
dataset['ca'].fillna(ca_median, inplace=True)
dataset['thal'].fillna(thal_mode, inplace=True)
```

Kod przekształcający zbiór danych w macierz cech liczbowych (przykłady x cechy)

W przypadku cech kategorycznych używam ‘one hot encoder’ do kodowania wartości:

```
mapped_column_names = {'cp_1': 'cp_typical_angina', 'cp_2': 'cp_atypical_angina', 'cp_3':
'cp_non_anginal_pain', 'cp_4': 'cp_asymptomatic',
                       'restecg_0': 'restecg_normal', 'restecg_1': 'restecg_wave_abnormality',
'restecg_2': 'restecg_definite_lvh',
                       'thal_3': 'thal_normal', 'thal_6': 'thal_fixed', 'thal_7':
'thal_reversable',
                       'slope_1': 'slope_upsloping', 'slope_2': 'slope_flat', 'slope_3':
'slope_downsloping'}
```

```
column_names = ['cp', 'restecg', 'thal', 'slope']
dataset[column_names] = dataset[column_names].astype('int64')
dataset_encoded = pd.get_dummies(dataset, columns=column_names)
dataset_encoded.rename(columns=mapped_column_names, inplace=True)
```

Kształt zbioru danych po przekształceniu to (303, 23)

Regresja logistyczna

1. **Przygotowanie i wstępne przetwarzanie danych**
   Ponieważ celem jest utworzenie klasyfikatora binarnego, przedefiniowujemy klasy w następujący sposób:
   - 0: brak chorób serca
   - 1: ma chorobę serca

```
X = dataset_encoded.drop('num', axis=1)
y = dataset_encoded['num']

y = y.replace([1, 2, 3, 4], 1)
y.value_counts()
```

```
0    164
1    139
Name: num, dtype: int64
```

W kroku wstępnego przetwarzania danych używamy standardowego skalera ze scikit-learn do skalowania cech numerycznych:

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler().fit(X[num_features])
X[num_features] = scaler.transform(X[num_features])
```

2. Implementacja regresji logistycznej

```
import numpy as np

def sigmoid(z):
  return 1 / (1 + np.exp(-z))

class LogisticRegression():
  def __init__(self, learning_rate=1e-3, max_iterations=500,
batch_size=5, convergence_threshold=1e-5, random_state=0):
    self.learning_rate = learning_rate
    self.max_iterations = max_iterations
    self.batch_size = batch_size
    self.convergence_threshold = convergence_threshold
    self.random_state = random_state
    self.cost_list = []
    self.theta = None

  def loss(self, y, pred):
    return -(y * np.log(pred) + (1 - y) * np.log(1 - pred))

  def fit(self, X, y):
    self.theta = np.zeros(X.shape[1] + 1)
    X_bias = np.c_[np.ones(X.shape[0]), X]
    np.random.seed(self.random_state)
    for epoch in range(self.max_iterations):
      loss_batch = []
      for i in range(0, len(y), self.batch_size):
        X_batch = X_bias[i:i + self.batch_size]
        y_batch = y[i:i + self.batch_size]
```

```
        preds = sigmoid(X_batch.dot(self.theta))
        gradient = X_batch.T.dot(preds - y_batch) / len(y_batch)
        self.theta -= self.learning_rate * gradient
        loss_batch.append(np.mean(self.loss(y_batch, preds)))

      cost = np.mean(loss_batch)
      if (len(self.cost_list) > 0 and abs(cost - self.cost_list[-1]) <
self.convergence_threshold):
        break
      self.cost_list.append(cost)

  def predict(self, X):
    X_bias = np.c_[np.ones(X.shape[0]), X]
    preds = sigmoid(X_bias.dot(self.theta))
    y_preds = [1 if pred > 0.5 else 0 for pred in preds]
    return y_preds
```

+ Używana funkcja aktywacji to funkcja sigmoidalna. Hiperparametry modelu to “learning_rate”, “max_iterations”, “batch_size”, “convergence_threshold”, i “random_state”.

+ Funkcja strat jest funkcją entropii krzyżowej.

+ Model uczy się według rozmiaru paczki, który jest przekazywany jako hiperparametr.

+ Proces uczenia się zostaje zakończony, gdy osiągnie maksymalną liczbę iteracji lub osiągnie próg zbieżności.

3. Weryfikacja modelu

Podział danych na dane treningowe i dane testowe:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Dopasowanie modelu i metryki:

```
model = LogisticRegression(learning_rate=2e-3, max_iterations=1000, batch_size=5, convergence_threshold=1e-5, random_state=42)
model.fit(X_train, y_train)

y_preds = model.predict(X_test)

from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

print("Recall: %.2f" % recall_score(y_test, y_preds))
print("Precision: %.2f" % precision_score(y_test, y_preds))
print("F1: %.2f" % f1_score(y_test, y_preds))
print("Accuracy: %.2f" % accuracy_score(y_test, y_preds))
```

```
Recall: 0.91
Precision: 0.85
F1: 0.88
Accuracy: 0.87
```

Wykres kosztu trenowania w każdej iteracji: