Heart Disease Prediction Using Logistic Regression

- **Author:** Favero Daniele (VR506229)
- **Course:** *Machine Learning & Deep Learning* (4S010673), MSc in AI, Università di Verona
- **Academic year:** 2024/25

Contents

The task

The goal of this project is to develop a machine learning model capable of discerning whether a patient has heart disease based on a set of medical parameters such as age, cholesterol levels, blood pressure, chest pain type, etc.

> ***Heart disease:*** *also known as cardiovascular disease or cardiopathy; set of heart conditions that include diseased vessels, structural problems, and blood clots.*

The approach

Given the nature of the task (binary classification) and the need for interpretability that medical ML models require, the model chosen for this project is **logistic regression**.

Tools used

- Python
- Numpy
- Pandas
- Scikit-learn
- Joblib

The dataset

The Cleveland Heart Disease dataset is one of the most used databases for heart disease prediction research in machine learning.

- **Number of instances:** 303
- **Number of features:** 13 + 1 target label
- **Feature types:** categorical, integer, real
- **Contains missing values:** yes

The following is an overview of the features and their possible values:

- `age`: age in years
- `sex`: patient sex

    - 1 = male
    - 0 = female
- `cp`: chest pain type
    - 1 = typical angina
    - 2 = atypical angina
    - 3 = non-anginal pain
    - 4 = asymptomatic
- `trestbps`: blood pressure at rest, in mmHg
- `chol`: serum cholesterol, in mg/dl
- `fbs`: fasting blood sugar > 120 mg/dl
    - 1 = true
    - 0 = false
- `restecg`: resting electrocardiographic results
    - 0 = normal
    - 1 = having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria
- `thalach`: maximum heart rate achieved
- `exang`: exercise induced angina
    - 1 = true
    - 0 = false
- `oldpeak`: ST depression induced by exercise relative to rest
- `slope`: the slope of the peak exercise ST segment
    - 1 = upsloping
    - 2 = flat
    - 3 = downsloping
- `ca`: number of major vessels colored by flouroscopy (integer values 0 to 3)
- `thal`: thallium stress test results
    - 3 = normal
    - 6 = fixed defect
    - 7 = reversable defect
- `target`: diagnosis of heart disease
    - 0 = false
    - 1 = true
    - 2 = true
    - 3 = true
    - 4 = true

The dataset is saved in the `data` directory under the file name `dataset.data`.

Data preprocessing

Data loading

First, on `src/preprocessing.py`, the names of all 14 columns are defined and stored on the `columns` variable. The dataset is then loaded into a Pandas dataframe. The values on the `dataset.data` file are separated by commas, so `pd.read_csv()` can be used while setting `delimiter` to `","`.

Printing the first few rows with `print(dataFrame.head())`, resulting in this table:

```
    age  sex   cp  trestbps   chol  ...  oldpeak  slope   ca  thal  target
0  63.0  1.0  1.0     145.0  233.0  ...      2.3    3.0  0.0   6.0       0
1  67.0  1.0  4.0     160.0  286.0  ...      1.5    2.0  3.0   3.0       2
2  67.0  1.0  4.0     120.0  229.0  ...      2.6    2.0  2.0   7.0       1
3  37.0  1.0  3.0     130.0  250.0  ...      3.5    3.0  0.0   3.0       0
4  41.0  0.0  2.0     130.0  204.0  ...      1.4    1.0  0.0   3.0       0
```

Finding and handling missing values

Missing values can take many shapes and forms. They can be truly missing, they can be numbers outside the expected range, or they can be of an invalid type. In order to find all missing values, the best approach is to define what constitutes a valid variable for each of the features, then single out the outliers. This strategy requires domain knowledge but it guarantees that only meaningful values remain and the data is thoroughly cleaned.

The result of the search are 6 invalid values that take the form of `?`. Given that few missing values were found compared to the total number of rows of the database, the most practical approach is to remove their rows entirely using `dataFrame.dropna()`, bringing the number of rows to 297.

Target column value conversion

As the goal of the model is to perform a binary classification task, the values of the `target` feature are not valid:

- `target`: diagnosis of heart disease

- 0 = false
- 1 = true
- 2 = true
- 3 = true
- 4 = true

`dataFrame_cleaned.loc[:, "target"] = (dataFrame_cleaned["target"] > 0).astype(int)` converts every value that is larger than 0 to 1, solving this issue.

Assessing class distribution

`print(dataFrame_cleaned["target"].value_counts())` outputs

```
0   160
1   137
```

which means the two classes are fairly balanced and there is no need for corrective measures.

Data splitting

In order to avoid data leakage, data splitting has to be applied *before* some pre-processing steps such as one-hot encoding and scaling.

The dataset is first split into features (X) and target labels (y):

```
X = dataFrame_cleaned.drop(columns=["target"])
y = dataFrame_cleaned["target"]
```

Then X and y are both split 80-20%:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=7, stratify=y)
```

The dataset is now properly split into four subsets:

- `X_train` is the set of 80% of the features, used for training
- `X_test` is the set of target labels corresponding to `X_train`, used for training
- `y_train` is the set of 20% of the features, used for testing
- `y_test` is the set of target labels corresponding to `y_train`, used for testing

One-hot encoding

When applying one-hot encoding the transformation is carried out separately for the training and the test sets:

```
X_train_OHE = pd.get_dummies(X_train, columns=columns_OHE, drop_first=True)
X_test_OHE = pd.get_dummies(X_test, columns=columns_OHE, drop_first=True)
```

To avoid errors due to missing columns:

```
X_test_OHE = X_test_OHE.reindex(columns=X_train_OHE.columns, fill_value=0)
```

Z-score normalization

When applying scaling such as z-score normalization, the scaling parameters have to be calculated on the training set `X_train_OHE`, then applied to both `X_train_OHE` and `X_test_OHE`.

In order to avoid data leakage – the possibility that information from the training set is introduced in the test set, leading to overly optimistic validation scores – normalization parameters must not be not calculated from scratch for the test set.

```
scaler = StandardScaler()
X_train_OHE_scaled = scaler.fit_transform(X_train_OHE[columns_scaling])
X_test_OHE_scaled = scaler.transform(X_test_OHE[columns_scaling])
```

Finally, the cleaned and pre-processed data splits are saved as `.csv` files onto the `data` directory.

Model training

After loading the pre-processed data splits and initializing the model using `logReg = LogisticRegression(random_state=7)`, the model is fit on the training set of features `X_train` and the target labels `y_train`:

```
logReg.fit(X_train, y_train)
```

The model is subsequently tested on `y_test`, outputting a prediction of `X_train`'s labels.

```
y_prediction = logReg.predict(X_test)
```

Model validation

To avoid variance in performance metrics, Monte Carlo cross-validation (MCCV) has been implemented in the model validation process, with the number of random data splits set to 50.

The following are a mean of all 50 values for each performance metric. Both accuracy and precision are above the mean of logistic regression models that use the same dataset (source; acc. mean: 0.81579, prec. mean: 0.83185).

```
Accuracy:   0.8449999999999999
Precision:  0.8707030703021514
Recall:     0.7885714285714286
F1 score:   0.8250964627803223
```

Future improvements

There are multiple ways to improve the results of the model:

- Combine the current dataset with other datasets that contain the features currently used by this model.

- Hyperparameter tuning (e.g., C regularization strength) maximizing recall performance (minimizing false negatives aligns with the goals of medical machine learning software users).
- Different penalty terms could improve performance.
- Quantify uncertainty in model predictions.