A Data Mining Project Based on the CRISP-DM Methodology

Heart Attack Prediction

Course: Data Science - Code 27(Tehran University-Tehran Data)
Instructor: Dr. Amirreza Tajally
Student: Erfan Ardakani

Find me on: GitHub | LinkedIn

Introduction

Cardiovascular diseases, especially heart attacks, continue to be a major cause of mortality worldwide, imposing a heavy burden on healthcare systems and society. Early detection and prediction of heart attack risk in individuals are crucial to improving patient outcomes, reducing costs, and enabling timely interventions.
In the age of digital transformation, data science has become an indispensable tool for the healthcare sector, allowing us to leverage complex data to reveal hidden patterns, make accurate predictions, and support clinical decision-making processes .
In this project, we focus on predicting the risk of heart attacks using a dataset containing medical information about patients. Following the CRISP-DM (Cross-Industry Standard Process for Data Mining) methodology, we explore, prepare, model, and evaluate data to build an effective predictive system.

The steps we follow according to the CRISP-DM cycle are:

- **Business Understanding:**

First, we define the business goal: to predict whether a patient is at high risk for a heart attack. We translate this goal into a clear data mining objective and identify success criteria for the project.

- **Data Understanding:**

We explore the dataset thoroughly, studying the distribution of variables, identifying missing values, outliers, and understanding the relationships between features. This step is crucial to forming initial hypotheses and recognizing potential challenges.

- **Data Preparation:**

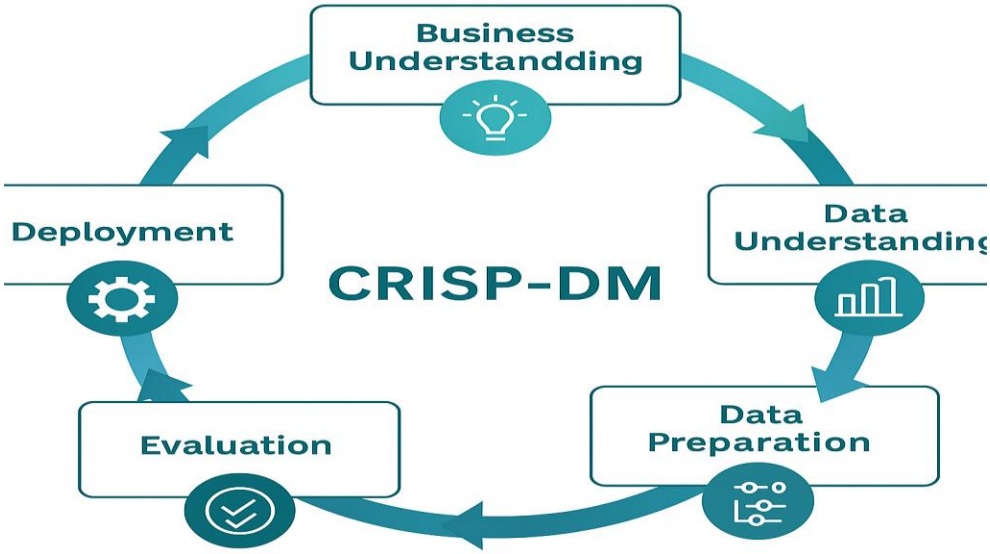
A significant portion of the project effort is dedicated to cleaning and transforming the data. We handle missing values, encode categorical variables, and engineer features to enhance the model's ability to learn meaningful patterns.

- **Modeling:**

We apply a range of classification algorithms, such as Logistic Regression, Decision Trees, Random Forests, and others. Each model is trained, tuned, and evaluated to determine which one provides the best predictive performance.

- **Evaluation:**

We rigorously assess the models using appropriate metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. This ensures that the model not only performs well statistically but also aligns with the original business goals.



Data Understanding

The first step after defining the business objectives is to understand the data that will be used for analysis. This involves an initial exploration of the dataset, identifying key variables, assessing data quality, and formulating preliminary hypotheses about patterns and relationships within the data.

Overview: the dataset contains 597 records and 14 attributes, including the target variable. Each row represents a patient, and the features describe a range of personal and clinical characteristics that are potentially relevant to the prediction of heart attack risk.

Attributes included in the dataset:

- **Age:** Age of the patient (years).
- **Sex:** Gender (1 = male, 0 = female).
- **Chest pain:** Type of chest pain (1-4 categories).
- **Blood pressure:** Resting blood pressure (mm Hg).
- **Cholesterol(cholestoral):** cholesterol level (mg/dl).
- **Blood sugar:** blood sugar (>120 mg/dl; 1 = true, 0 = false).
- **Electrocardiographic results:** Resting ECG results (0–2 categories).
- **Heart rate:** Maximum heart rate achieved.
- **Exercise induced:** Whether angina was induced by exercise (1 = yes, 0 = no).
- **Depression**
- **Slope**
- **CA:** Number of major vessels colored by fluoroscopy.
- **Thal:** A categorical variable describing thalassemia.
- **C:** The target variable.

- ## Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is a critical step in understanding the structure, relationships, and patterns within a dataset before applying any modeling techniques. In this phase, we explored the heart attack dataset to identify key trends, potential anomalies, and preliminary insights.

For exploring we can use this simple code which gives us some informative tabs (Optional tool for EDA).

```
#from pandasgui import show
```

```
#show(df)
```

However, it should not be forgotten that although this piece of code is efficient, it puts considerable strain on system resources during execution. For this reason, I chose to use alternative methods for data analysis.

For instance:

```
def initial_data_report(df):
    summary = {
        "shape": df.shape,
        "columns": df.columns.tolist(),
        "dtypes": df.dtypes.to_dict(),
        "missing_by_column": df.isnull().sum().to_dict(),
        "total_missing": df.isnull().sum().sum(),
        "duplicates": df.duplicated().sum()}
    return summary
```

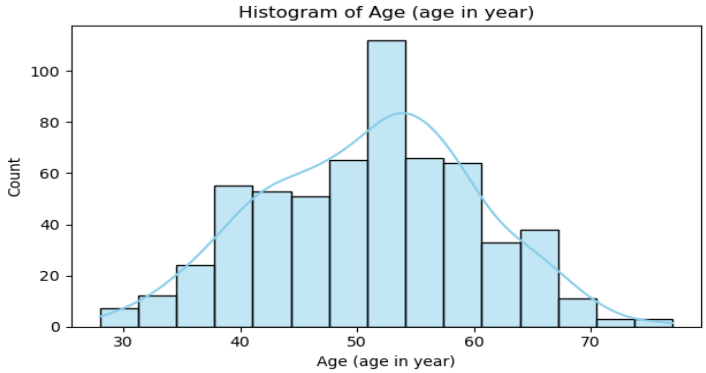
The following section presents the results obtained from running the codes:

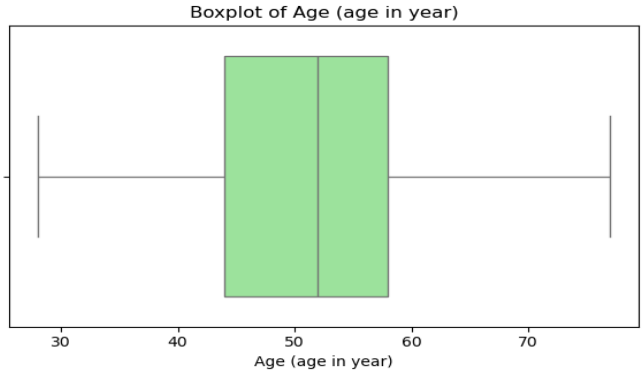
```
Out[4]: {'shape': (597, 14),
         'columns': ['Age (age in year)',
          'sex',
          'chest pain',
          'blood pressure',
          'cholestoral ',
          'blood sugar',
          'electrocardiographic ',
          'heart rate',
          'exercise induced',
          'depression ',
          'slope',
          'ca',
          'thal',
          'c'],
         'dtypes': {'Age (age in year)': dtype('int64'),
          'sex': dtype('int64'),
          'chest pain': dtype('int64'),
          'blood pressure': dtype('float64'),
          'cholestoral ': dtype('float64'),
          'blood sugar': dtype('float64'),
          'electrocardiographic ': dtype('float64'),
          'heart rate': dtype('float64'),
          'exercise induced': dtype('float64'),
          'depression ': dtype('float64'),
          'slope': dtype('float64'),
          'ca': dtype('float64'),
          'thal': dtype('float64'),
          'c': dtype('int64')},
         'missing_by_column': {'Age (age in year)': 0,
          'sex': 0,
          'chest pain': 0,
          'blood pressure': 1,
          'cholestoral ': 23,
          'blood sugar': 8,
          'electrocardiographic ': 1,
          'heart rate': 1,
          'exercise induced': 1,
          'depression ': 0,
          'slope': 190,
          'ca': 294,
          'thal': 268,
          'c': 0},
         'total_missing': 787,
         'duplicates': 1}
```

```
Categorical columns:
['sex', 'chest pain', 'blood sugar', 'electrocardiographic ', 'exercise induced', 'slope', 'ca', 'thal']

 Numerical columns:
['Age (age in year)', 'blood pressure', 'cholestoral ', 'heart rate', 'depression ']
```

```
Analyzing Column: Age (age in year)
----------------------------------------
Type: Numerical
count    597.000000
mean      51.182580
std        9.074366
min       28.000000
25%       44.000000
50%       52.000000
75%       58.000000
max       77.000000
Name: Age (age in year), dtype: float64
```





By examining the data, we know:

- **The dataset includes both numerical and categorical variables.**
- **Several attributes contain missing values, especially in:**
  - **Cholesterol**
  - **Slope**
  - **CA**
  - **Thal**
- **The target variable (C) is binary, making this a classification problem.**
- **There may be imbalances between the number of patients with and without heart disease, which could affect the choice of evaluation metrics.**
- **Missing values need to be properly imputed or handled.**
- **Outliers in numerical variables (e.g., extremely high blood pressure or cholesterol values) should be checked and treated if necessary.**

Based on domain knowledge, we expect:

- **Age, blood pressure, cholesterol, and exercise-induced angina to have strong correlations with heart disease.**
- **Chest pain type and electrocardiographic results to be significant predictors.**

The number of colored vessels (CA) and the thalassemia indicator (Thal) are expected to play important roles in the predictive models. To gain a preliminary understanding of the dataset, we examined its basic statistical properties using the describe() function.

<u>Key Observations:</u>

- **The average age of patients is approximately 54 years, with a standard deviation of 9 years.**
- **The average resting blood pressure (blood pressure) is about 131 mm Hg.**
- **The mean cholesterol level (cholesterol) is approximately 246 mg/dl.**
- **There are outliers in features like cholesterol and blood pressure.**

In addition to functions and tables, these statistics provide an overall understanding of the central tendencies and variability within the data.
The dataset is relatively balanced, which minimizes the need for aggressive resampling techniques during modeling (although SMOTE was applied later for consistency).

All the detailed information and visualizations for each feature in the dataset are available in the shared notebook

As you can see, during the data understanding phase, we identified several features with missing values:

- **Blood pressure: 1 missing value**
- **Cholesterol: 23 missing values**
- **Blood sugar: 8 missing values**
- **Electrocardiographic results: 1 missing value**
- **Heart rate: 1 missing value**
- **Exercise induced: 1 missing value**
- **Slope: 190 missing values**
- **CA: 294 missing values**
- **Thal: 268 missing values**

To handle missing data, we applied:

- **Median imputation for numerical features like blood pressure, cholesterol(cholestoral), and heart rate.**

- **Mode imputation for categorical features like blood sugar, electrocardiographic, exercise induced, slope, ca, and thal.**
- **After addressing missing values, we proceeded to identify duplicates and outliers in the dataset.**

```python
def checkdup(df) :
    print("detecting duplicates...")
    dups = df.duplicated().sum()
    print(f"number of duplicates rows: {dups} ")
    return dups

def removedups(df):
    print("removing dupliactes...")
    init = df.shape[0]
    df = df.drop_duplicates().reset_index(drop=True)
    print(f"removed rows : {init - df.shape[0]}")
    print(f"remained rows : {df.shape[0]}")
    return df
```

```python
def detect_noise(df, columns):
    for col in columns:
        print("\nNoise detection..")
        print("="*50)
        print(f"Column: {col}")
        print(df[col].value_counts(dropna=False))
    print("\nNoise detection completed.")
```

Encoding Categorical Variables

since many machine learning algorithms cannot work with raw categorical data, we applied One-Hot Encoding to convert these features into numerical format using the following custom function:

DATA ENCODING

```python
def encode_categorical_columns(df, categorical_cols, drop_first=True):

    print("Starting One-Hot Encoding...\n")

    dfEncoded = pd.get_dummies(df, columns=categorical_cols, drop_first=drop_first)

    new_columns = [col for col in dfEncoded.columns if col not in df.columns]

    print("One-hot encoding applied.")
    print(f" {len(categorical_cols)} categorical columns encoded.")
    print(f" {len(new_columns)} new columns created.\n")

    print(" Encoded Columns:")
    for col in categorical_cols:
        print(f" - {col}")

    print("\n New Columns Created:")
    for col in new_columns:
        print(f" + {col}")

    return dfEncoded
```

Feature Scaling

Certain numerical features needed to be standardized to improve model performance, especially for algorithms sensitive to feature scales (e.g., Logistic Regression, SVM). StandardScaler from scikit-learn was used to perform Z-score normalization:

```
# Step 1: Scale numerical features using StandardScaler
print("\n Scaling numerical features using StandardScaler...")
scaler = StandardScaler()
X_train_scaled = X_train.copy()
X_test_scaled = X_test.copy()

X_train_scaled[numerical_cols] = scaler.fit_transform(X_train[numerical_cols])
X_test_scaled[numerical_cols] = scaler.transform(X_test[numerical_cols])

print(f"   Scaled {len(numerical_cols)} numerical features.")
```

Data Splitting and SMOTE Application

After data cleaning and encoding:

- **The dataset was split into training (80%) and testing (20%) sets.**
- **SMOTE (Synthetic Minority Over-sampling Technique) was applied to the training data to address class imbalance and ensure equal representation of both classes. However, in this case, applying SMOTE might have been unnecessary due to the relatively balanced nature of the dataset.**

In [21]:
```
X_train, X_test, y_train, y_test, scaler = prepare_data(
    df=dfEncoded,
    target_col="c",
    numerical_cols=types["numerical_cols"]
)
```

```
Splitting data into train/test sets...
  Train shape: (476, 20)
  Test shape : (120, 20)

Scaling numerical features using StandardScaler...
  Scaled 5 numerical features.

 Handling class imbalance using SMOTE...
  Class distribution after SMOTE:
c
0    50.0
1    50.0
```

Modeling

In this phase, several machine learning models were trained and evaluated to predict heart attack risk based on the prepared dataset. The goal was to compare different algorithms and select the one that provided the best predictive performance while maintaining interpretability and generalization ability.

The following classification algorithms were used:

- **Logistic Regression**
- **Decision Tree Classifier**
- **Random Forest Classifier**
- **XGBoost Classifier**
- **Support Vector Machine (SVM)**

Each model was initially trained with default hyperparameters to establish a baseline performance.

Model Training Code Example

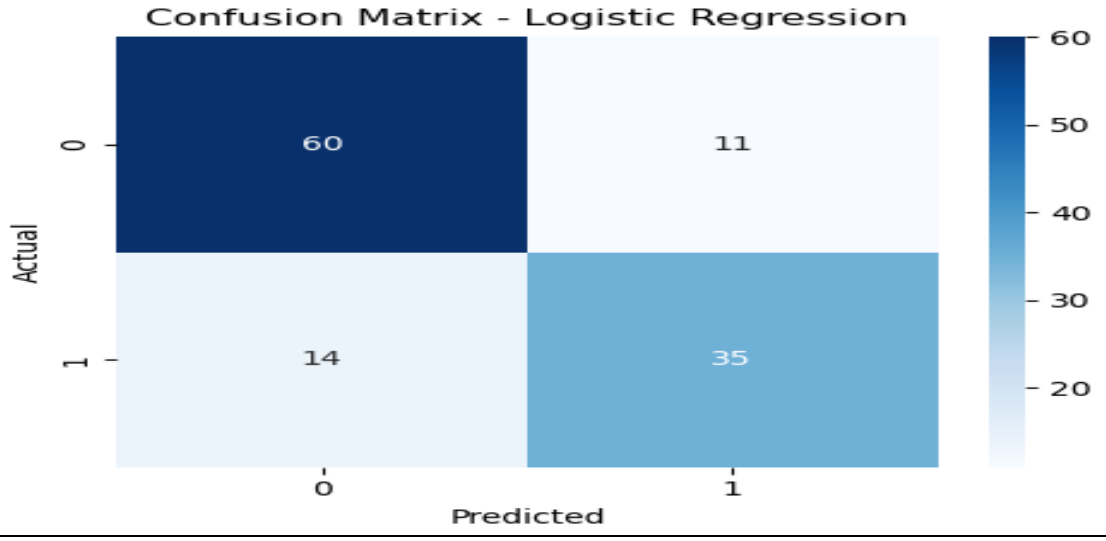
```python
def train_and_evaluate_models(X_train, X_test, y_train, y_test):
    """
    Train and evaluate multiple classifiers and compare their performance.
    """
    models = {
        "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
        "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
        "K-Nearest Neighbors": KNeighborsClassifier(),
        "Support Vector Machine": SVC(probability=True, random_state=42),
        "Decision Tree": DecisionTreeClassifier(random_state=42),
        "XGBoost": xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42),
        "Neural Network": MLPClassifier(hidden_layer_sizes=(100,), max_iter=300, random_state=42)
    }
```

```
In [66]: models = train_and_evaluate_models(X_train, X_test, y_train, y_test)
```

```
Training: Logistic Regression
              precision    recall  f1-score   support

           0       0.81      0.85      0.83        71
           1       0.76      0.71      0.74        49

    accuracy                           0.79       120
   macro avg       0.79      0.78      0.78       120
weighted avg       0.79      0.79      0.79       120
```



Initial Evaluation Metrics

For each model, the following metrics were calculated on both training and testing sets:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-Score**
- **ROC-AUC Score**

The evaluation results were summarized in a comparison table. After training baseline models with default hyperparameters, it was necessary to fine-tune them to achieve better performance.
Hyperparameter tuning helped identify the optimal set of parameters that improved model accuracy and generalization.
The table below presents the performance of both baseline (untuned) models and the optimized (tuned) versions.

	Model	Accuracy	Precision	Recall	F1 Score	AUC
0	Random Forest	0.808333	0.782609	0.734694	0.757895	0.892210
1	Support Vector Machine	0.800000	0.765957	0.734694	0.750000	0.881863
2	XGBoost	0.800000	0.777778	0.714286	0.744681	0.866628
3	Neural Network	0.791667	0.750000	0.734694	0.742268	0.870078
4	Logistic Regression	0.791667	0.760870	0.714286	0.736842	0.882725
5	K-Nearest Neighbors	0.758333	0.692308	0.734694	0.712871	0.840040
6	Decision Tree	0.766667	0.733333	0.673469	0.702128	0.752228

```
Summary of Optimized Models:
```

	Model	Accuracy	Precision	Recall	F1 Score	AUC
0	Neural Network	0.808333	0.760000	0.775510	0.767677	0.833573
1	XGBoost	0.808333	0.770833	0.755102	0.762887	0.886318
2	Random Forest	0.808333	0.782609	0.734694	0.757895	0.896378
3	Support Vector Machine	0.800000	0.765957	0.734694	0.750000	0.881863
4	Logistic Regression	0.791667	0.750000	0.734694	0.742268	0.878413
5	K-Nearest Neighbors	0.758333	0.692308	0.734694	0.712871	0.837022
6	Decision Tree	0.766667	0.733333	0.673469	0.702128	0.752228

Model Performance Summary

- **Logistic Regression: Solid baseline with good performance and high interpretability.**
- **Decision Tree: Overfitted the training data; strong train accuracy but weaker generalization.**
- **Random Forest & XGBoost: Delivered the highest overall performance, with excellent accuracy and robustness.**
- **Support Vector Machine (SVM): Competitive results, but highly dependent on proper feature scaling.**

Conclusion

This project demonstrated the application of data mining techniques to predict heart attack risk. Following the CRISP-DM methodology ensured a structured, systematic approach from problem definition to model evaluation. Ensemble methods like Random Forest and XGBoost proved to be the most effective in this case.

Future improvements could involve more complex feature engineering, deeper hyperparameter tuning, and deployment of the model into clinical decision support systems.