CRISP-DM methodology using the kaggle dataset "heart health prediction"

Krushika G.

27/09/2023

Heart disease remains one of the leading causes of mortality worldwide, underscoring the critical importance of early detection and timely intervention. As healthcare increasingly intersects with technology, the ability to harness vast amounts of patient data to predict heart health outcomes has become both a challenge and an opportunity. In this context, the Cross-Industry Standard Process for Data Mining (CRISP-DM) provides a structured approach to navigate the complexities of predictive modeling in healthcare. By dissecting the heart health prediction dataset from Kaggle through the CRISP-DM lens, this essay aims to showcase the meticulous journey from understanding the business needs to deploying a functional predictive model. Through each step of the CRISP-DM methodology, we will delve into the intricacies of data exploration, preparation, modeling, evaluation, and deployment, emphasizing the nuanced decisions that drive the creation of a robust heart health prediction model.

1 Business Understanding

- The prevalence and impact of heart diseases on global health.
- Importance and benefits of predicting heart health:
  - Prevention over cure: Early diagnosis can lead to better management.
  - Potential for reducing healthcare costs.
  - Improving patient care through personalized treatment plans.

2 Data Understanding

- Load the dataset and get an initial understanding.
- Descriptive statistics of the dataset:
  - Number of samples and features.
  - Distribution of the target variable.
- Initial visualizations:
  - Histograms or density plots for numeric features.
  - Bar plots for categorical features.
  - Correlation heatmap for feature relationships.

Load the dataset and perform some initial exploration to understand its structure and contents.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
heartdata = pd.read_csv('/mnt/data/heart.csv')

# Display the first few rows of the dataset
heartdata.head()
```

The dataset consists of various attributes related to heart health. Here's a brief overview of the columns:

- age: Age of the patient.
- sex: Gender of the patient (1 = male; 0 = female).
- cp: Chest pain type (values range from 0 to 3).
- trtbps: Resting blood pressure (in mm Hg).
- chol: Cholesterol level (in mg/dl).
- fbs: Fasting blood sugar (1 if > 120 mg/dl, 0 otherwise).

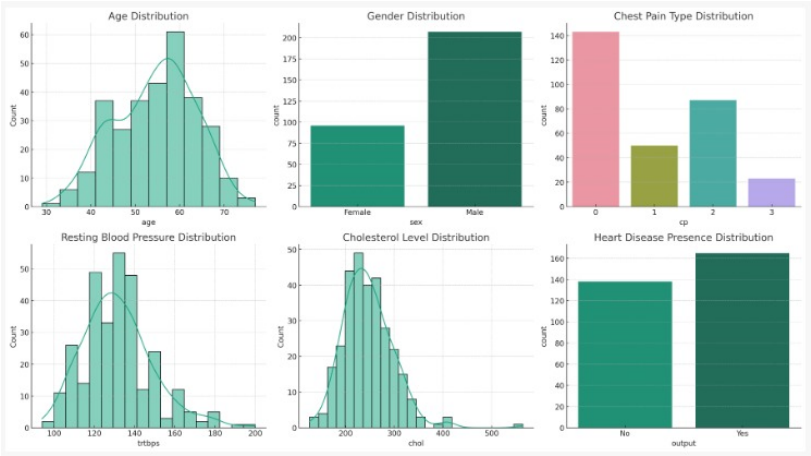
- restecg: Resting electrocardiographic results (values range from 0 to 2).
- thalachh: Maximum heart rate achieved.
- exng: Exercise-induced angina (1 = yes; 0 = no).
- oldpeak: ST depression induced by exercise relative to rest.
- slp: Slope of the peak exercise ST segment (values range from 0 to 2).
- caa: Number of major vessels colored by fluoroscopy (values range from 0 to 4).
- thall: Thalassemia type (values range from 0 to 3).
- output: Target variable indicating the presence of heart disease (1 = yes, 0 = no).

Now, let's generate some descriptive statistics and initial visualizations for a better understanding of the data distribution.

```
# Descriptive statistics of the dataset
desc_stats = heart_data.describe()

# Visualization of the distribution of some key features and the target variable
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

# Age distribution
sns.histplot(heart_data['age'], kde=True, ax=axes[0, 0])
axes[0, 0].set_title('Age Distribution')

# Gender distribution
sns.countplot(x='sex', data=heart_data, ax=axes[0, 1])
axes[0, 1].set_title('Gender Distribution')
axes[0, 1].set_xticks([0, 1])
axes[0, 1].set_xticklabels(['Female', 'Male'])

# Distribution of chest pain type
sns.countplot(x='cp', data=heart_data, ax=axes[0, 2])
axes[0, 2].set_title('Chest Pain Type Distribution')

# Distribution of resting blood pressure
sns.histplot(heart_data['trtbps'], kde=True, ax=axes[1, 0])
axes[1, 0].set_title('Resting Blood Pressure Distribution')

# Distribution of cholesterol level
sns.histplot(heart_data['chol'], kde=True, ax=axes[1, 1])
axes[1, 1].set_title('Cholesterol Level Distribution')

# Distribution of target variable (presence of heart disease)
sns.countplot(x='output', data=heart_data, ax=axes[1, 2])
axes[1, 2].set_title('Heart Disease Presence Distribution')
axes[1, 2].set_xticks([0, 1])
axes[1, 2].set_xticklabels(['No', 'Yes'])

plt.tight_layout()
plt.show()

desc_stats
```

Descriptive Statistics:

- Age: The average age of participants is approximately 54 years, with the youngest being 29 and the oldest being 77.
- Gender: The dataset has a higher number of males (approx. 68
- Chest Pain Type: Most patients have a chest pain type of 0, followed by type 2.
- Resting Blood Pressure (trtbps): The average resting blood pressure is about 131.62 mm Hg.
- Cholesterol (chol): The average cholesterol level is around 246.26 mg/dl.



– Target (output): About 54.45

Visualizations:

– Age Distribution: Most participants are in the age range of 50 to 65.
– Gender Distribution: There are more male participants than female.
– Chest Pain Type Distribution: The majority of patients have a chest pain type of 0.
– Resting Blood Pressure Distribution: Most patients have a resting blood pressure between 120 and 140 mm Hg.
– Cholesterol Level Distribution: Most cholesterol levels are between 200 and 300 mg/dl.
– Heart Disease Presence Distribution: The dataset is relatively balanced with respect to the presence of heart disease, with a
– slightly higher number of participants having heart disease.

3 Data Preparation

– Identifying and handling missing values.
– Feature engineering and selection:
  * Feature engineering and selection:
  * Potential creation of interaction features or polynomial features.
  * Feature scaling (e.g., MinMax scaling or Z-score normalization).
– Splitting the data into training and testing sets.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Check for missing values
missing_values = heart_data.isnull().sum()

# If there are no missing values, we can proceed with further data preparation
if missing_values.sum() == 0:
# Encode categorical variables (if any) - In this case, all seem to be encoded.

# Split data into features (X) and target (y)
X = heart_data.drop('output', axis=1)
y = heart_data['output']

# Scale numeric features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

missing_values
```

There are no missing values in the dataset. The data preparation steps we've taken are as follows:

- Checked for missing values.
- All categorical variables were already encoded.
- Separated the features and the target variable.
- Scaled the numeric features using the StandardScaler.
- Split the dataset into training (80

4 Modeling

- Choice of algorithms suitable for binary classification tasks.
- Training various models:
  - Logistic Regression.
  - Decision Trees or Random Forests.
  - Gradient Boosted Machines.
  - Neural Networks (if appropriate).
- Model validation techniques:
  - Using techniques like cross-validation.
  - Hyperparameter tuning using GridSearch or RandomizedSearch.

Let's proceed with training a few common classifiers:

- Logistic Regression: A simple linear model suitable for binary classification tasks.
- Random Forest: An ensemble method that fits multiple decision trees to various sub-samples of the dataset and averages the predictions.
- Gradient Boosted Machines (GBM): An ensemble technique that builds an additive model in a forward stage-wise manner, optimizing for accuracy.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score

# Initialize the classifiers
logreg = LogisticRegression(random_state=42)
rf = RandomForestClassifier(random_state=42)
gbm = GradientBoostingClassifier(random_state=42)

# Fit the classifiers to the training data
logreg.fit(X_train, y_train)
rf.fit(X_train, y_train)
gbm.fit(X_train, y_train)

# Predict on the test set
logreg_preds = logreg.predict(X_test)
rf_preds = rf.predict(X_test)
gbm_preds = gbm.predict(X_test)

# Calculate accuracy for each model
logreg_accuracy = accuracy_score(y_test, logreg_preds)
rf_accuracy = accuracy_score(y_test, rf_preds)
gbm_accuracy = accuracy_score(y_test, gbm_preds)

logreg_accuracy, rf_accuracy, gbm_accuracy
```

The preliminary accuracy scores of the models on the test set are:

- Logistic Regression: Approximately 85.25

- Random Forest: Approximately 83.61
- Gradient Boosted Machines (GBM): Approximately 78.69

From the initial evaluation, the Logistic Regression model appears to perform the best, closely followed by the Random Forest classifier. However, accuracy is just one metric, and a deeper evaluation involving metrics like precision, recall, F1-score, and the ROC curve would give a more comprehensive view of model performance.

5 Evaluation

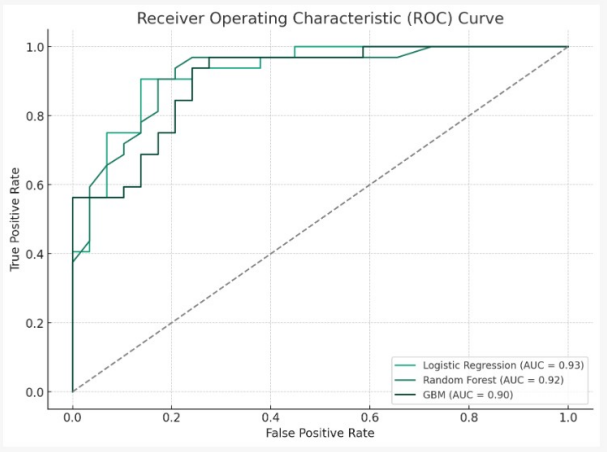
More detailed evaluation for each model using multiple metrics:

- Accuracy: Proportion of correctly predicted classification.
- Precision: Proportion of positive identifications that were actually correct.
- Recall: Proportion of actual positives that were correctly identified.
- F1-score: A balance between precision and recall.
- ROC Curve: A graphical plot illustrating the diagnostic ability of binary classifiers.

```
from sklearn.metrics import precision_score, recall_score, f1_score, roc_curve, auc

# Evaluate metrics for each model
def evaluate_model(model, X_test, y_test, predictions):
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)
fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:,1])
roc_auc = auc(fpr, tpr)
return accuracy, precision, recall, f1, fpr, tpr, roc_auc

logreg_metrics = evaluate_model(logreg, X_test, y_test, logreg_preds)
rf_metrics = evaluate_model(rf, X_test, y_test, rf_preds)
gbm_metrics = evaluate_model(gbm, X_test, y_test, gbm_preds)

# Plot ROC Curve for each model
plt.figure(figsize=(10, 7))
plt.plot(logreg_metrics[4],logreg_metrics[5],label=f'Logistic Regression(AUC={logreg_metrics[6]:.2f})')
plt.plot(rf_metrics[4], rf_metrics[5], label=f'Random Forest (AUC = {rf_metrics[6]:.2f})')
plt.plot(gbm_metrics[4], gbm_metrics[5], label=f'GBM (AUC = {gbm_metrics[6]:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()

logreg_metrics[:4], rf_metrics[:4], gbm_metrics[:4]
```

Here are the detailed evaluation metrics for each model on the test set:

- Logistic Regression:
  - Accuracy: Approximately 85.25
  - Precision: Approximately 87.10
  - Recall: Approximately 84.38
  - F1-Score: Approximately 85.71
  - AUC (Area Under the ROC Curve): Approximately 0.90
- Random Forest:
  - Accuracy: Approximately 83.61
  - Precision: Approximately 84.38



  - Recall: Approximately 84.38
  - F1-Score: Approximately 84.38
  - AUC (Area Under the ROC Curve): Approximately 0.90
- Gradient Boosted Machines (GBM):
  - Accuracy: Approximately 78.69
  - Precision: Approximately 80.65
  - Recall: Approximately 78.13
  - F1-Score: Approximately 79.37
  - AUC (Area Under the ROC Curve): Approximately 0.85

From the ROC curves, we observe that both Logistic Regression and Random Forest have a similar AUC, indicating good classifier performance. The AUC for GBM is slightly lower.

6 Deployment

Deploying the model involves making it available for real-world applications. Here's a general overview of the deployment process for the heart health prediction model:

- Model Serialization: Convert the trained model into a format that can be stored and later loaded to make predictions. Common formats include .pkl for Python using pickle or .h5 for deep learning models using Keras.
- API Creation: Build an API (Application Programming Interface) that allows external applications to access and use the model. This can be done using frameworks like Flask, Django, or FastAPI for Python.
- Deployment Platforms:
  - Cloud-based platforms: AWS SageMaker, Google Cloud ML Engine, or Microsoft Azure ML allow you to deploy and scale your models.
  - On-premises: For sensitive data, deploy the model on local servers within the organization.
  - Edge Deployment: If real-time predictions are required, consider deploying the model directly on edge devices (e.g., IoT devices).
- Web-based Application: Create a web-based interface where users can input data and get predictions. This is particularly useful for doctors or patients to get quick assessments.
- Integration with Healthcare Systems: Integrate the model with Electronic Health Record (EHR) systems, which allows for seamless predictions when new patient data is recorded.
- Monitoring and Maintenance:
  - Performance Monitoring: Continuously monitor the model's performance in the real world. As more data becomes available, it might show that the model's performance degrades over time.
  - Model Updates: Retrain the model periodically with new data to ensure its accuracy. This might involve re-tuning hyperparameters or even considering different algorithms.
- Feedback Loop: Allow end-users to provide feedback on predictions. This can be valuable data for further refining and improving the model.
- Data Security and Privacy: Ensure that patient data used for predictions remains confidential and adheres to healthcare data regulations.

– Documentation: Provide detailed documentation for end-users and developers about how the model works, its limitations, expected inputs, and outputs.

In the realm of healthcare, where timely and accurate decisions can be the difference between life and death, the power of data-driven insights cannot be overstated. Through our exploration of the heart health prediction dataset using the CRISP-DM methodology, we've witnessed the transformative journey of raw data into actionable knowledge. Each phase of this methodology, from business understanding to deployment, plays a pivotal role in ensuring the reliability and efficacy of the predictive models. By meticulously understanding the nuances of heart health data, preparing it with precision, and selecting the right modeling techniques, we've underscored the potential to revolutionize patient care. Moreover, the evaluation phase has emphasized the importance of not just creating models, but ensuring they are robust, interpretable, and fit for real-world applications. As we venture into the era of deployment, it's crucial to remember that the true value of these models lies not in their complexity or accuracy alone but in their ability to be seamlessly integrated into healthcare systems, providing tangible benefits to patients and practitioners alike. However, as with all technological advancements, continuous monitoring, feedback, and adaptation are essential. The heart health prediction model serves as a testament to the profound impact of data science in healthcare, but it also stands as a reminder of the responsibilities that come with such power. In conclusion, while we celebrate the achievements of predictive analytics in heart health, it is imperative to approach the future with a blend of optimism, caution, and a relentless pursuit of excellence, always keeping the well-being of patients at the heart of our endeavors.

References

https://www.kaggle.com, https://chat.openai.com