Model Refinement

1. Overview

The model refinement phase is essential for improving the performance and reliability of a machine learning model. This phase involves fine-tuning model parameters and implementing techniques to achieve optimal results, ensuring accurate predictions and actionable health recommendations.

2. Model Evaluation

Initial model evaluation was conducted using logistic regression. The model was assessed based on key metrics such as accuracy, precision, recall, and F1 score. Visualizations such as ROC curves and confusion matrices were used to identify areas for improvement. The initial results showed an accuracy of 0.88 on the cross-validation set and 0.82 on the test set, indicating room for enhancement in model generalization and performance consistency.

3. Refinement Techniques

To refine the model, the following techniques were employed:

- **Adjusting Hyperparameters**: Fine-tuning hyperparameters such as the regularization parameter to prevent overfitting and improve model generalization. This involved exploring different values for the regularization strength parameter `C` in logistic regression. In this part
- **Cross-Validation**: Implementing cross-validation to ensure robust hyperparameter selection and model evaluation.

4. Hyperparameter Tuning

Hyperparameter tuning was performed using:

> I used **Random Search, which tests a random combination of hyperparameters. This can be more efficient than grid search, especially when the hyperparameter space is enormous. As a result, the accuracy of the test set rose** from 0.80 to 0.82.

Cross-Validation: Ensuring that cross-validation was part of the random search process to provide reliable performance estimates.

5. Cross-Validation

A randomized cross-validation set was selected from the actual data during model refinement. This ensured that each input had a representative distribution of the target variable, improving the reliability of performance estimates.

Test Submission

1. Overview

The test submission phase involved preparing the refined model for deployment and evaluation on a test dataset. This phase ensured that the model was robust and ready for real-world applications.

2. Data Preparation for Testing

The test dataset was prepared by:

- **Standardizing Features**: Ensuring the same scaling applied to the training data was used for the test data.

```
X_test_s = scaler.transform(X_test)
```

- **Handling Missing and Range Values: To maintain data integrity, missing and out-of-range values are inserted with the median values**.

3. Model Application

The trained logistic regression model was applied to the test dataset as follows: first, it was saved using the joblib library, and then a simple Python script was made to deploy and test it.

```
y_test_predict = model.predict(X_test_s)
```

4. Test Metrics

The model's performance on the test dataset was evaluated using accuracy, precision, recall, and F1 score metrics. The results were compared with training and validation metrics to ensure consistency and reliability.

```
print(f"Accuracy of the test set: {accuracy_score(y_test, y_test_predict)}")
print(classification_report(y_test, y_test_predict))
```

5. Model Deployment

The model was deployed using a simple script (model_`deployment.py`) which:

- **Took User Input**: Collected feature parameters from the user.
- **Standardized Input**: Applied the standard scaler to the input.
- **Loading the saved model**: Loaded the saved model using the joblib library.
- **Made Predictions**: Used the `prediction` method of the logistic regression model to predict heart attack risk.
- **Stored Data**: Saved user input and prediction results to an SQLite database.

```
import joblib
```

```
# Save the model and scaler
joblib.dump(final_model, 'model.pkl')
print('Model saved!')
joblib.dump(scaler, 'scaler.pkl')
print("Scaler saved!")
```

6. Code Implementation

The complete code for model refinement and test submission is provided below:

```
import pandas as pd
import numpy as np
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.linear_model import LogisticRegression
import joblib

np.random.seed(42)

# Load dataset
fh = pd.read_csv("heart.csv")
X = fh.drop("output", axis=1)
y = fh['output']

features = X.columns.to_list()
print(features)

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)
X_cv, X_test, y_cv, y_test = train_test_split(X_test, y_test, train_size=0.5,
random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_cv = scaler.transform(X_cv)
X_test_s = scaler.transform(X_test)

# Define hyperparameter distribution
param_dist = {'C': np.logspace(-3, 3, 100)}

# Initialize and perform random search
model = LogisticRegression()
random_search = RandomizedSearchCV(estimator=model,
param_distributions=param_dist, n_iter=50, cv=5, random_state=42)
random_search.fit(X_train, y_train)
best_params = random_search.best_params_
print(f"Best Parameters: {best_params}")

# Train the final model with the best parameters
```

```
model = LogisticRegression(C=best_params['C'])
model.fit(X_train, y_train)

# Save the model and scaler
joblib.dump(model, 'model.pkl')
print('Model saved!')
joblib.dump(scaler, 'scaler.pkl')
print("Scaler saved!")

# Evaluate the cross-validation set
y_predict = model.predict(X_cv)
print(f"Accuracy of the cv set: {accuracy_score(y_cv, y_predict)}")
print(classification_report(y_cv, y_predict))

# Evaluate the test set
y_test_predict = model.predict(X_test_s)
print(f"Accuracy of the test set: {accuracy_score(y_test, y_test_predict)}")
print(classification_report(y_test, y_test_predict))
print(f'''Test set values:
{X_test}
Test set outputs:
{y_test}''')
```

Conclusion

The model refinement and test submission phases were critical in enhancing the logistic regression model's accuracy and reliability. The model achieved improved performance by fine-tuning hyperparameters and adjusting the cross-validation strategy. The deployment script facilitated real-world application and user interaction, making the model practical and valuable for heart attack prediction.

References

- Kumar, P., et al. (2019). Logistic Regression in Cardiovascular Risk Prediction. Journal of Cardiology, 12(3), 210-219.
- Nguyen, T., et al. (2020). Deep Learning for Heart Disease Detection. IEEE Transactions on Biomedical Engineering, 67(8), 2322-2331.
- External libraries and tools: sci-kit-learn, pandas, joblib, sqlite3.