STATS 3DA3

Homework Assignment 6

Ashley Chen, Jasmine Ho, JC Abanto

2025-04-16

Question

1)

Our target variable measure the severity of heart disease which can be defined as binary classification problem. Our goal would then to be able to predict the probability of heart disease. This means we can carry out a logistic regression and a random forest classifier to predict the presence and absence of heart disease.

2)

```python
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler

heart_disease = fetch_ucirepo(id=45)

X = heart_disease.data.features
y = heart_disease.data.targets

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X)
```

3)

We have 13 features and 1 target variable. Starting with our features, we have age (age in years), sex (1 = male; 0 = female), cp (1: typical angina, 2: atypical angina, 3: non-anginal pain, 4: asymptomatic), trestbps (resting blood pressure in mm Hg on admission to the hospital), chol (serum cholestoral in mg/dl), fbs (fasting blood sugar $>$ 120 mg/dl where 1 = true; 0 = false), restecg (resting electrocardiographic results), exang (exercise induced angina), oldpeak (ST depression induced by exercise relative to rest), slope (1 = upsloping, 2 = flat, 3 = downsloping), ca (number of major vessels colored by floursopy), thal (3 = normal; 6 = fixed defect; 7 = reversable defect). Finally our target variable, 'num', is the diagnosis of heart disease.

```
print(f"Observations in X: {len(X)}")
print(f"Summary of X:\n{X.describe()}")
print(f"Summary of y:\n{y.describe()}")
```

We find that the average age of patients is 54.4 years old, with a standard deviation of 9.1 years.

```
print(f"Data Types of X: \n{X.dtypes}")
print(f"\nData Types of y: \n{y.dtypes}")
```

All of our data types in X are numerical but some representing categorical variables.

4)

```
print(f"y before transformation: {y['num'].value_counts()}")
y['num'] = y['num'].apply(lambda x: 1 if x > 0 else 0)
print(f"y before transformation: {y['num'].value_counts()}")
```

5)

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

corr_matrix = X.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)
plt.title("Correlation Heatmap")
plt.show()
```

From this correlation matrix, we can conclude that thalach (max heart rate) has a strong negative correlation (-0.39) with age and oldpeak. We can assume that younger individuals tend to have a higher heart rate, while those with more severe heart disease (higher oldpeak) have lower thalach. We also found that ca (number of major vessels) has a strong positive correlation (0.36) with age. We can say that older individuals are more likely to have more blocked vessels.

6)

```
X = X.dropna()
y = y.loc[X.index]

print(f"Length of X after transformations: {len(X)}")
print(f"Length of y after transformations: {len(y)}")
```

7)

```
import pandas as pd
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Assume X is your original DataFrame
categorical_columns = ['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'thal']
X_cleaned = X.drop(columns=categorical_columns)

# Scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_cleaned)
```

```
k_range = range(2,10)
silhouette_scores = []

for k in k_range:
    kmeans = KMeans(n_clusters = k, n_init = 20, random_state = 0)
    cluster_labels = kmeans.fit_predict(X_scaled)
    silhouette_avg = silhouette_score(X_scaled, cluster_labels)
    silhouette_scores.append(silhouette_avg)

plt.figure(figsize=(9, 5))
plt.plot(k_range, silhouette_scores, marker='o')
plt.ylabel("Silhouette Score")
plt.xlabel("Number of Clusters (k)")
plt.title("k Values Silhouette Score")
plt.show()
```

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

kmeans = KMeans(n_clusters=2, random_state=42)
clusters = kmeans.fit_predict(X_scaled)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='viridis')
plt.title("K-Means Clustering (k=2)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label="Cluster")
plt.show()

X_scaled_df = pd.DataFrame(X_scaled, columns=X_cleaned.columns)
X_scaled_df['cluster'] = clusters
```

8)

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1)
```

9)

We are going to use logistic regression and random forest. Logistic regression is suitable for this assignment because it predicts a binary outcome, in which case, the target variable is 0 or 1. Random forest is also a good classifier to use because it has high predictive accuracy and does not depend on linear relationships, which is good for this dataset as there are both numerical and categorical variables.

10)

We are going to use accuracy and F1 scores to compare the classifier performance between logistic regression and random forest. We can create the confusion matrix from the predictions to calculate the accuracy and F1 scores. Accuracy scores are calculated by the number of correct predictions over the total number of predictions.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Where:

$$TP = \text{True Positives}$$

$$TN = \text{True Negatives}$$

$$FP = \text{False Positives}$$

$$FN = \text{False Negatives}$$

To obtain the F1 score, we will need to use precision and recall that are derived from the confusion matrix. Once we calculate that, the F1 score can be calculated by:

$$\text{Precision} = \frac{TP}{TP + FP}$$
$$\text{Recall} = \frac{TP}{TP + FN}$$
$$F_1 = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Using these metrics, we can compare the overall accuracy and balance between the two classifiers to determine which classifier most optimal for predicting heart disease.

11)

```
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

log_reg = LogisticRegression(solver='liblinear', max_iter=1000, random_state=42)

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] # test this out to see if accuracy improves
}

grid_search = GridSearchCV(estimator=log_reg, param_grid=param_grid, cv=5, scoring='accuracy',
grid_search.fit(X_train_scaled, y_train)
```

```
print("Best parameters found: ", grid_search.best_params_)
print("Best cross-validation accuracy: {:.4f}".format(grid_search.best_score_))

best_log_reg = grid_search.best_estimator_
y_pred = best_log_reg.predict(X_test_scaled)
y_pred2 = best_log_reg.predict(X_train_scaled)

accuracy = accuracy_score(y_test, y_pred)
train_accuracy = accuracy_score(y_train, y_pred2)
cm = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred, target_names=['No Heart Disease', 'Heart Diseas

print(f"Test set accuracy: {accuracy:.4f}")
print(f"Train set accuracy: {train_accuracy:.4f}")
print("Confusion Matrix:")
print(cm)
print("Classification Report:")
print(report)
```

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report



rf_clf = RandomForestClassifier(random_state=42)
rf_param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'criterion': ['gini', 'entropy']
}

rf_grid_search = GridSearchCV(estimator=rf_clf, param_grid=rf_param_grid, cv=5, scoring='accura
rf_grid_search.fit(X_train_scaled, y_train)

print("Random Forest Best Parameters:", rf_grid_search.best_params_)
print("Random Forest Best Cross-Validation Accuracy:", rf_grid_search.best_score_)

best_rf_clf = rf_grid_search.best_estimator_
rf_y_pred = best_rf_clf.predict(X_test_scaled)

rf_accuracy = accuracy_score(y_test, rf_y_pred)
rf_cm = confusion_matrix(y_test, rf_y_pred)
rf_report = classification_report(y_test, rf_y_pred, target_names=[f'Class {i}' for i in np.un

print(f"Random Forest Test Accuracy: {rf_accuracy:.4f}")
print("Random Forest Confusion Matrix:")
print(rf_cm)
print("Random Forest Classification Report:")
print(rf_report)
```

References

Stack Overflow. (2014, February 16). Fine-tuning parameters in logistic regression. Stack Overflow. https://stackoverflow.com/questions/21816346/fine-tuning-parameters-in-logistic-regression

Stack Overflow. (2016, July 17). Random Forest Hyperparameter Tuning - scikit-learn using GridSearchCV. Stack Overflow. https://stackoverflow.com/questions/35164310/random-forest-hyperparameter-tuning-scikit-learn-using-gridsearchcv