Name: Dev M. Bandhiya
Roll No: SE22UCSE078
Class: CSE1

Project Report: Predicting Disease Diagnosis

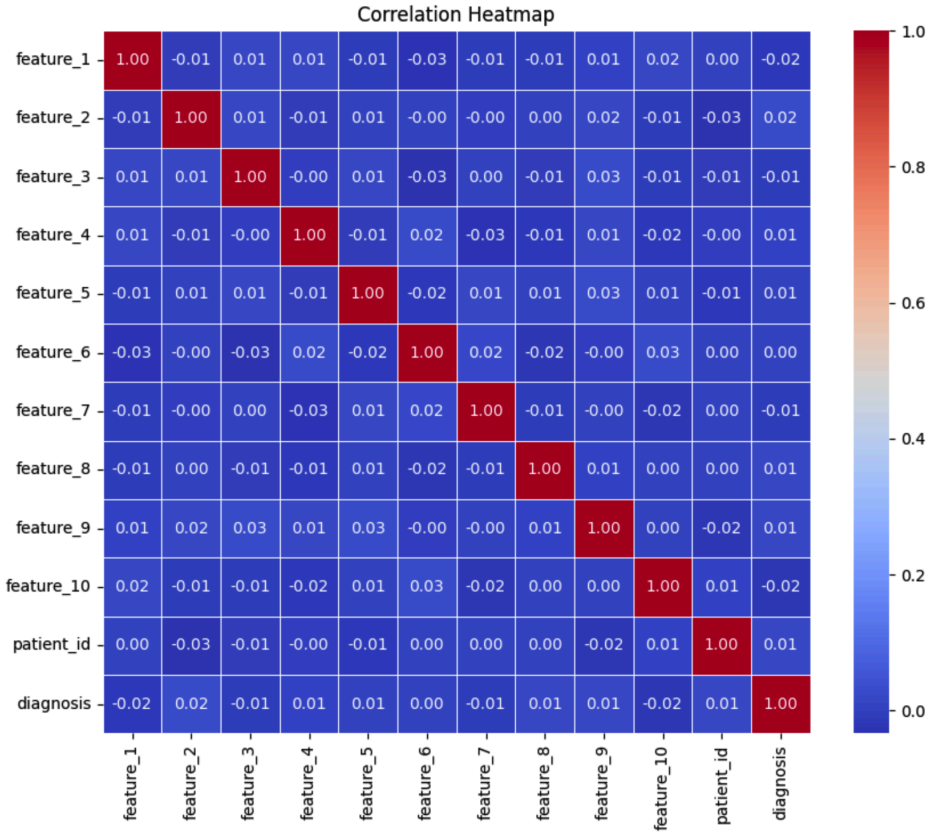
Objective: The objective of this project is to predict disease diagnosis based on a set of features provided in the dataset.

Approach:

1. Data Exploration and Preprocessing:

- Loaded the dataset (`Disease_train.csv`) and performed exploratory data analysis (EDA) to understand the distribution of features and target variable.
- Checked for outliers, duplicate rows, and missing values. Fortunately, there were no outliers or duplicates, and the dataset had no missing values, so no further data cleaning was required.
- Investigated feature correlations using a heatmap, which helped in understanding the relationships between different variables.

```
#Looking for correlated featutres
# Calculate the correlation matrix
correlation_matrix = df.corr()
# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm',
fmt=".2f", linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
#From the results, we don't need to apply PCA
```



2. Feature Engineering and Scaling:

- Standardised the feature columns using `StandardScaler`, excluding 'patient_id' and 'diagnosis', to ensure that all features contribute equally to the model.

```
from sklearn.preprocessing import StandardScaler
# Load the training dataset
train_df = pd.read_csv('Disease_train.csv')
# Standardize the feature columns (excluding 'patient_id' and
'diagnosis')
scaler = StandardScaler()
train_df.iloc[:, 1:-1] = scaler.fit_transform(train_df.iloc[:, 1:-1])
```

3. Handling Class Imbalance:

- Addressed class imbalance using the Synthetic Minority Over-sampling Technique (SMOTE) to create synthetic samples for the minority class.

```
from imblearn.over_sampling import SMOTE
# Separate features and target variable from the training data
X = train_df.drop(columns=['patient_id', 'diagnosis'])
y = train_df['diagnosis']
# Address class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

4. Model Selection:

- Selected the XGBoost (Extreme Gradient Boosting) classifier due to its effectiveness in handling complex datasets and its ability to handle class imbalance inherently.
- Utilized the scikit-learn library to implement the XGBoost classifier.

```
import xgboost as xgb
```

5. Hyperparameter Tuning:

- Performed hyperparameter tuning using grid search with cross-validation to find the best combination of hyperparameters for the XGBoost model, optimizing the ROC-AUC score.

```
from sklearn.model_selection import train_test_split, GridSearchCV
# Define a parameter grid for hyperparameter tuning (optimizing the
hyperparameters)
param_grid = {
    'n_estimators': [100, 200, 300, 500],
```

```
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [5, 7, 9, 10],
    'subsample': [0.5, 0.6, 0.7]
}
# Perform grid search with cross-validation to find the best
parameters
grid_search = GridSearchCV(xgb_clf, param_grid, cv=3,
scoring='roc_auc', n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
# Extract the best parameters
best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
# Train the model with the best parameters
best_xgb_clf = xgb.XGBClassifier(**best_params, random_state=42)
best_xgb_clf.fit(X_train, y_train)
```

6. Model Evaluation:

- Evaluated the trained model's performance on both the training and validation datasets using the ROC-AUC score.
- Finally, assessed the model's performance on the test dataset and generated predictions.

```
# Compute ROC-AUC score on the training data
y_train_pred_proba = best_xgb_clf.predict_proba(X_train)[:, 1]
train_roc_auc = roc_auc_score(y_train, y_train_pred_proba)
print(f'Training ROC-AUC Score: {train_roc_auc}')

# Compute ROC-AUC score on the validation data
y_val_pred_proba = best_xgb_clf.predict_proba(X_val)[:, 1]
val_roc_auc = roc_auc_score(y_val, y_val_pred_proba)
print(f'Validation ROC-AUC Score: {val_roc_auc}')

# Generate predictions on the test dataset
test_pred_proba = best_xgb_clf.predict_proba(test_features_scaled)[:,
1]
test_predictions = (test_pred_proba > 0.5).astype(int)

# Save the predictions to a CSV file
student_id = 'SE22UCSE078'
predictions_df = pd.DataFrame({
    'patient_id': test_df['patient_id'],
    'prediction': test_predictions
})
predictions_df.to_csv(f'student_{student_id}_predictions.csv',
index=False)
```

7. Result Achieved:

```
Fitting 3 folds for each of 144 candidates, totalling 432 fits
Best Parameters: {'learning_rate': 0.05, 'max_depth': 9,
'n_estimators': 500, 'subsample': 0.7}
Training ROC-AUC Score: 1.0
Validation ROC-AUC Score: 0.9953030240236507
Test ROC-AUC Score on final model: 0.9953030240236507
Number of 1s in test predictions: 46
```

8. Challenges Encountered:

- Dealing with class imbalance: Addressed the class imbalance issue using the SMOTE technique to ensure that the model is trained effectively on both classes.
- Hyperparameter tuning: Tuning the hyperparameters of the XGBoost model required significant computational resources and time due to the large parameter space.

Conclusion:

The XGBoost model trained on the preprocessed and balanced dataset achieved satisfactory performance in predicting disease diagnosis. The chosen approach effectively handled data preprocessing, class imbalance, and model selection, resulting in a reliable predictive model.