Name: M. Bilal

ID: F2021266574

Liver Predictor Project Report

1. Introduction

The "Liver Predictor" project aims to develop and evaluate machine learning models to predict liver disease based on various health indicators. By analyzing a dataset containing features related to patient health and liver function, the goal is to create predictive models that can accurately classify individuals as either having liver disease or not. This project seeks to identify the most effective machine learning algorithms for this task, providing a tool that could potentially aid in early diagnosis and intervention. This report details the dataset preparation, model training, evaluation, and performance analysis of different models.

2. Data Preparation

2.1. Loading the Dataset

The dataset was loaded from a CSV file using pandas for initial inspection and processing.

```
import pandas as pd

# Load the dataset
train = pd.read_csv("Liver_disease_data.csv")
train.head()
```

Details:

- The dataset comprises health indicators and a target variable indicating liver disease status.
- The `head()` function displays the first few rows, allowing us to check the data's structure.

2.2. Data Inspection

We examined the dataset for missing values and data types.

```
for column in train:
    print(f"Column {column}: {train[column].isnull().sum()} null values/type:",
train[column].dtype)
```

Details:

- Identifying missing values and data types helps in deciding on appropriate preprocessing steps.

2.3. Data Preprocessing

Scaling:

Features were scaled using `MinMaxScaler` to normalize values between 0 and 1.

```
from sklearn import preprocessing

# Convert DataFrame to numpy array
train = train.values

# Scale features
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(train)
train = pd.DataFrame(x_scaled)

# Define features and target
X = train.loc[:, 0:9]
Y = train.loc[:, 10]
```

Details:

- Scaling ensures that all features contribute equally to model training.

Data Splitting:

The dataset was divided into training and testing sets.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25,
random_state=42)
```

Details:

- 25% of the data was set aside for testing, ensuring a fair evaluation of model performance.

3. Model Training and Evaluation

3.1. XGBoost Classifier

Training and Evaluation:

```
import xgboost as xgb
from sklearn.metrics import confusion_matrix, f1_score, precision_score,
recall_score
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Initialize and train the model
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
```

```
# Making predictions
y_pred = model.predict(X_test)
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
```

```
sns.heatmap(cm, annot=True, fmt='g', linewidth=0.1, linecolor='black',
xticklabels=['0', '1'], yticklabels=['0', '1'], cmap="Blues")
plt.title("Confusion Matrix for XGBoost")
plt.show()

# Performance Metrics
print("XGBoost Classifier")
print(f"F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Precision: {precision_score(y_test, y_pred, average='macro')}")
print(f"Recall: {recall_score(y_test, y_pred, average='macro')}")
```

Details:

- XGBoost is efficient and performs well with complex datasets.
- Metrics such as F1 Score, Precision, and Recall quantify the model's performance.

3.2. LightGBM Classifier

Training and Evaluation:

```
from lightgbm import LGBMClassifier

# Initialize and train the model
lgbm = LGBMClassifier(metric='auc')
lgbm.fit(X_train, y_train)

# Making predictions
y_pred = lgbm.predict(X_test)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='g', linewidth=0.1, linecolor='black',
xticklabels=['0', '1'], yticklabels=['0', '1'], cmap="Blues")
plt.title("Confusion Matrix for LightGBM")
plt.show()

# Performance Metrics
print("LightGBM Classifier")
print(f"F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Precision: {precision_score(y_test, y_pred, average='macro')}")
print(f"Recall: {recall_score(y_test, y_pred, average='macro')}")
```

Details:

- LightGBM is known for speed and accuracy, especially on large datasets.

3.3. CatBoost Classifier

Training and Evaluation:

```
from catboost import CatBoostClassifier

# Initialize and train the model
```

```
catc = CatBoostClassifier(learning_rate=0.01, verbose=0)
catc.fit(X_train, y_train)
```

```
# Making predictions
y_pred = catc.predict(X_test)
```

```
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='g', linewidth=0.1, linecolor='black',
xticklabels=['0', '1'], yticklabels=['0', '1'], cmap="Blues")
plt.title("Confusion Matrix for CatBoost")
plt.show()
```

```
# Performance Metrics
print("CatBoost Classifier")
print(f"F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Precision: {precision_score(y_test, y_pred, average='macro')}")
print(f"Recall: {recall_score(y_test, y_pred, average='macro')}")
```

Details:

- CatBoost handled the dataset well and achieved the best performance metrics, making it the preferred model.
- CatBoost's ability to manage categorical features and its robust performance led to its selection.

3.4. Neural Network using TensorFlow

Building and Training the Model:

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

```
# Define the neural network model
model = Sequential()
model.add(Dense(10, kernel_initializer='normal', input_dim=10,
activation='relu'))
model.add(Dense(8, kernel_initializer='normal', activation='relu'))
model.add(Dense(4, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal', activation='sigmoid'))
```

```
# Compile the model
model.compile(loss='binary_crossentropy',
optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
metrics=['accuracy'])
```

```
# Callbacks
lr_reducer = ReduceLROnPlateau(monitor='val_accuracy', factor=0.2, patience=2,
min_lr=10e-7, cooldown=1, verbose=1)
early_stopper = EarlyStopping(monitor='val_accuracy', min_delta=0,
patience=10, verbose=1)
callbacks = [lr_reducer, early_stopper]
```

```
# Train the model
history = model.fit(X_train, y_train, batch_size=16, epochs=200,
validation_split=0.15, callbacks=callbacks)

# Making predictions
y_pred = model.predict(X_test)
y_pred = np.rint(y_pred).astype(int)

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred, labels=[0, 1])
sns.heatmap(cm, annot=True, fmt='g', linewidth=0.1, linecolor='black',
xticklabels=['0', '1'], yticklabels=['0', '1'], cmap="Blues")
plt.title("Confusion Matrix for Neural Network")
plt.show()

# Performance Metrics
print("Neural Network")
print(f"F1 Score: {f1_score(y_test, y_pred, average='macro')}")
print(f"Precision: {precision_score(y_test, y_pred, average='macro')}")
print(f"Recall: {recall_score(y_test, y_pred, average='macro')}")
```

Details:

- The neural network model demonstrated the capacity to learn complex patterns but required careful tuning and training.

4. Results and Analysis

4.1. Model Performance Summary

The performance metrics for each model are as follows:

XGBoost Classifier

- **F1 Score:** [0.8767126344445932]
- **Precision:** [0.8764214046822743]
- **Recall:** [0.8770435054493478]

LightGBM Classifier

- **F1 Score:** [0.8835688048260939]
- **Precision:** [0.8841545352743561]
- **Recall:** [0.8830735215293908

CatBoost Classifier ***(Best Performer)***

- **F1 Score:** [0.9214325328134715
- **Precision:** [0.9232127592567598]
- **Recall:** [0.920169287118099]

CatBoost outperformed other models in terms of F1 Score, Precision, and Recall, demonstrating its superior ability to handle the liver disease prediction task effectively.

4.2. Confusion Matrix Insights

The confusion matrices for each model were visualized to analyze the true positive, true negative, false positive, and false negative rates. This visualization helps in understanding the model's performance in more detail.

- **XGBoost Classifier Confusion Matrix:**
    - **True Positives (TP):** Correctly identified cases of liver disease.
    - **True Negatives (TN):** Correctly identified non-cases of liver disease.
    - **False Positives (FP):** Incorrectly identified non-cases as liver disease.
    - **False Negatives (FN):** Incorrectly identified liver disease cases as non-disease.
- **LightGBM Classifier Confusion Matrix:**
    - The confusion matrix for LightGBM shows similar metrics, with visual indicators of model performance.
- **CatBoost Classifier Confusion Matrix:**
    - CatBoost's confusion matrix provides insights into how well it performs in classifying cases versus non-cases.
- **Neural Network Confusion Matrix:**
    - The neural network's confusion matrix reveals its performance in predicting liver disease cases and non-cases.

5. Conclusion

The "Liver Predictor" project successfully implemented and evaluated various machine learning models for liver disease prediction. The CatBoost Classifier emerged as the most effective model, providing the best balance of F1 Score, Precision, and Recall.

Future Work:

- **Hyperparameter Tuning:** Further optimization to enhance model performance.
- **Feature Engineering:** Incorporate additional features for improved accuracy.
- **Cross-Validation:** Use k-fold cross-validation for robust performance metrics.
- **Ensemble Methods:** Combine models to leverage their strengths.
- **Model Interpretability:** Apply techniques like SHAP or LIME for better understanding of model predictions.
- **Data Augmentation:** Expand dataset to improve model generalization.

Link:

https://github.com/Malik-12-21/Liver-predictor