Deakin University

Machine Learning

OnTrack Submission

# Task5.2D

*Submitted By:*
Sathiyanarayanan Senthil Kumar
s223789819
2024/05/20 03:41

*Tutor:*
Ming Liu

May 20, 2024

Cell 01

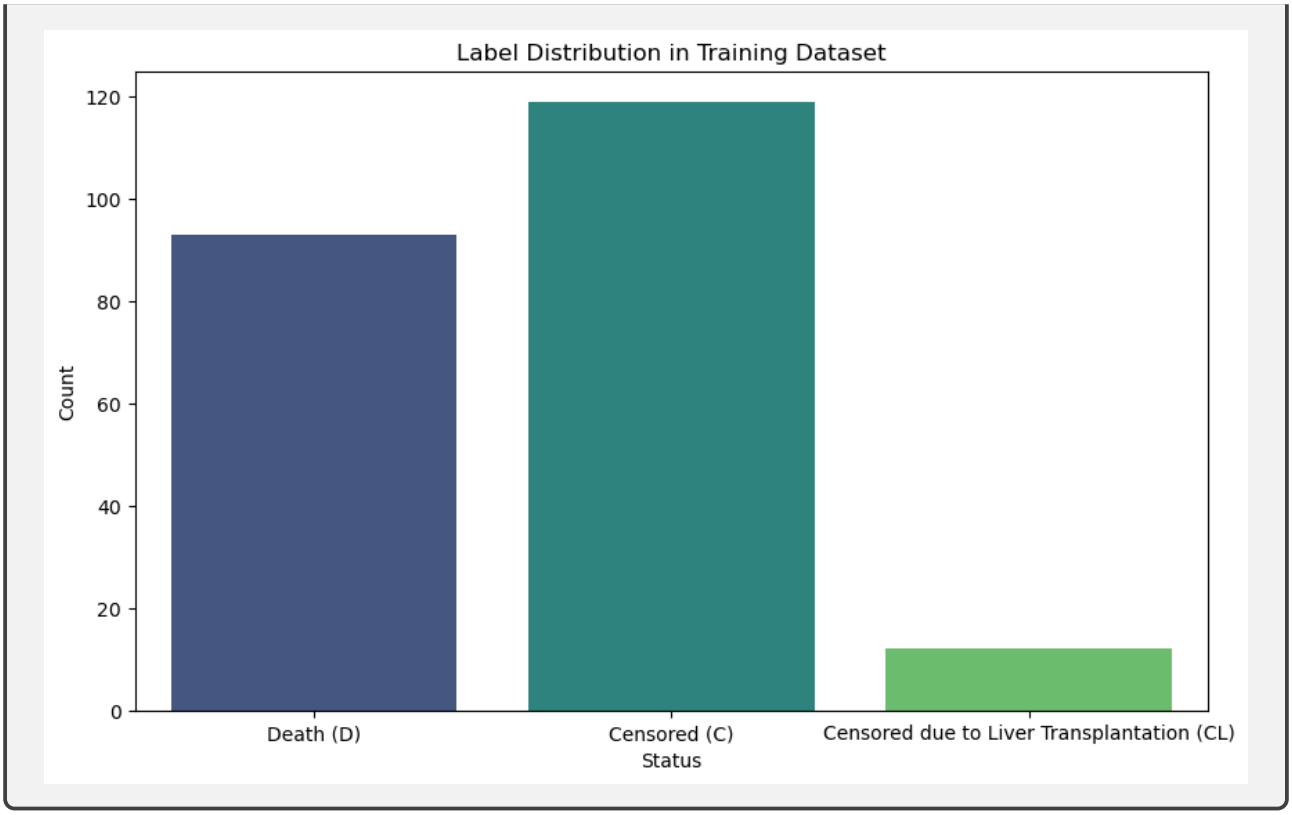
```
# Ans1 - Loading and exploring the datasets
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import seaborn as sns
import warnings

# (a) Loading the datasets
train_data = pd.read_csv('Data/cirrhosis_train.csv')
test_data = pd.read_csv('Data/cirrhosis_test.csv')

# Displaying dataset sizes
train_size = train_data.shape
test_size = test_data.shape

print(f"Train dataset size:{train_size}")
print(f"Test dataset size:{test_size}")

#############################################################################################

# (b) Displaying feature types and checking for missing values
print("\nTrain dataset info:")
train_info = train_data.info()
missing_train = train_data.isnull().sum()
print("\nMissing values in Train dataset")
print(missing_train[missing_train > 0])
print("\nTest dataset info:")
test_info = test_data.info()
missing_test = test_data.isnull().sum()
print("\nMissing values in Test dataset\n")
print(missing_test[missing_test > 0])

#############################################################################################

# (c) Identifying columns with missing values
train_id = train_data['trainID']
train_data = train_data.drop('trainID', axis=1)
test_id = test_data['testID']
test_data = test_data.drop('testID', axis=1)

missing_train_columns = train_data.columns[train_data.isnull().any()]
missing_test_columns = test_data.columns[test_data.isnull().any()]

# Filling missing values in the training set
for column in missing_train_columns:
    train_data[column].fillna(train_data[column].mean(), inplace=True)

# Filling missing values in the test set
for column in missing_test_columns:
    test_data[column].fillna(test_data[column].mean(), inplace=True)


# Ensuring 'Status' column is treated as a categorical column
train_data['Status'] = train_data['Status'].astype('category')

#############################################################################################

# (d) Label encoding for 'Status' column
status_mapping = {'D': 0, 'C': 1, 'CL': 2}
train_data['Status'] = train_data['Status'].map(status_mapping)
test_data['Status'] = test_data['Status'].map(status_mapping)

# List of other categorical columns to encode (excluding 'Status')
```

```
categorical_columns = train_data.select_dtypes(include=['object', 'category']).columns.tolist()
categorical_columns.remove('Status')

# Applying one-hot encoding for other categorical features
train_data = pd.get_dummies(train_data, columns=categorical_columns)
test_data = pd.get_dummies(test_data, columns=categorical_columns)

# Aligning the train and test dataframes by the columns
train_data, test_data = train_data.align(test_data, join='left', axis=1, fill_value=0)

print("\nInfo: Label encoding applied to 'Status' and one-hot encoding applied to other
↪  categorical features.")

##############################################################################################

# (e) Label distribution based on the training data
# Counts in status
label_distribution = train_data['Status'].value_counts()
print("\nLabel distribution in the training dataset:")
print(label_distribution)

# Plotting the label distribution
warnings.filterwarnings("ignore", category=FutureWarning, module="seaborn.categorical")
plt.figure(figsize=(10, 6))
status_order = [0, 1, 2]
sns.countplot(x='Status', data=train_data, palette='viridis', order = status_order)
plt.title('Label Distribution in Training Dataset')
plt.xlabel('Status')
plt.ylabel('Count')
plt.xticks(ticks=[0, 1, 2], labels=['Death (D)', 'Censored (C)', 'Censored due to Liver
↪  Transplantation (CL)'])
plt.show()
```

---

```
Train dataset size:(224, 20)
Test dataset size:(88, 20)

Train dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 224 entries, 0 to 223
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   trainID        224 non-null    int64
 1   N_Days         224 non-null    int64
 2   Status         224 non-null    object
 3   Drug           224 non-null    object
 4   Age            224 non-null    int64
 5   Sex            224 non-null    object
 6   Ascites        224 non-null    object
 7   Hepatomegaly   224 non-null    object
 8   Spiders        224 non-null    object
 9   Edema          224 non-null    object
 10  Bilirubin      224 non-null    float64
 11  Cholesterol    201 non-null    float64
 12  Albumin        224 non-null    float64
 13  Copper         222 non-null    float64
 14  Alk_Phos       224 non-null    float64
 15  SGOT           224 non-null    float64
 16  Tryglicerides  200 non-null    float64
 17  Platelets      221 non-null    float64
 18  Prothrombin    224 non-null    float64
 19  Stage          224 non-null    int64
```

```
dtypes: float64(9), int64(4), object(7)
memory usage: 35.1+ KB

Missing values in Train dataset
Cholesterol      23
Copper            2
Tryglicerides    24
Platelets         3
dtype: int64

Test dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88 entries, 0 to 87
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   testID         88 non-null     int64
 1   N_Days         88 non-null     int64
 2   Status         0 non-null      float64
 3   Drug           88 non-null     object
 4   Age            88 non-null     int64
 5   Sex            88 non-null     object
 6   Ascites        88 non-null     object
 7   Hepatomegaly   88 non-null     object
 8   Spiders        88 non-null     object
 9   Edema          88 non-null     object
 10  Bilirubin      88 non-null     float64
 11  Cholesterol    83 non-null     float64
 12  Albumin        88 non-null     float64
 13  Copper         88 non-null     int64
 14  Alk_Phos       88 non-null     float64
 15  SGOT           88 non-null     float64
 16  Tryglicerides  82 non-null     float64
 17  Platelets      87 non-null     float64
 18  Prothrombin    88 non-null     float64
 19  Stage          88 non-null     int64
dtypes: float64(9), int64(5), object(6)
memory usage: 13.9+ KB

Missing values in Test dataset

Status           88
Cholesterol       5
Tryglicerides     6
Platelets         1
dtype: int64

Info: Label encoding applied to 'Status' and one-hot encoding applied to other categorical
↪  features.

Label distribution in the training dataset:
Status
1    119
0     93
2     12
Name: count, dtype: int64
```


Label Distribution in Training Dataset
Count
120
100
80
60
40
20
0
Death (D)
Censored (C)
Censored due to Liver Transplantation (CL)
Status

Cell 02

```
# Ans 2 - Supervised Machine Learning Models
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier


# Splitting the data
X = train_data.drop('Status', axis=1)
y = train_data['Status']

# Standardizing the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Function to evaluate the models
def evaluate_model(model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)
    report = classification_report(y_val, y_pred, zero_division=0)
    return accuracy, report

# Model 1 - Logistic Regression
# Creating and evaluating Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)
accuracy_lr, report_lr = evaluate_model(log_reg)
print("Logistic Regression - Accuracy:", accuracy_lr)
print("Logistic Regression - Classification Report:\n", report_lr)

# Model 2 - Random Forest with Hyperparameter Tuning
# Defining the parameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initializing GridSearchCV with more cross-validation folds
rf_model = RandomForestClassifier(random_state=42)
grid_search_rf = GridSearchCV(estimator=rf_model, param_grid=param_grid_rf, cv=5, n_jobs=-1,
↪  verbose=2)

# Fitting GridSearchCV
grid_search_rf.fit(X_train, y_train)

# Getting the best parameters from GridSearchCV
best_params_rf = grid_search_rf.best_params_
print("Best parameters for Random Forest:", best_params_rf)

# Evaluating Random Forest with the best parameters
rf_best_model = RandomForestClassifier(**best_params_rf, random_state=42)
accuracy_rf_best, report_rf_best = evaluate_model(rf_best_model)
print("Random Forest (Best) - Accuracy:", accuracy_rf_best)
print("Random Forest (Best) - Classification Report:\n", report_rf_best)

# Model 3 - Gradient Boosting
# Creating and evaluating Gradient Boosting model
gb_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
```

```python
accuracy_gb, report_gb = evaluate_model(gb_model)
print("Gradient Boosting - Accuracy:", accuracy_gb)
print("Gradient Boosting - Classification Report:\n", report_gb)

# (b) Design Decisions for Each ML Model Used
# Logistic Regression: Simple and interpretable, useful as a baseline model.
# Random Forest: Can handle large datasets and capture non-linear relationships.
# Gradient Boosting: Builds an ensemble of weak learners to improve model performance, often
↪  yields high accuracy.

# (c) Logistic Regression: Hyperparameters Optimized: None.
# Random Forest: Hyperparameters Optimized: GridSearchCV
# Gradient Boosting: Hyperparameters Optimized: n_estimators (set to 100)

# (d) Oversampling, undersampling, class weights

# (e) from the result, the best model is Gradient Boosting
```

---

```
Logistic Regression - Accuracy: 0.8
Logistic Regression - Classification Report:
               precision    recall  f1-score   support

           0       0.81      0.72      0.76        18
           1       0.82      0.88      0.85        26
           2       0.00      0.00      0.00         1

    accuracy                           0.80        45
   macro avg       0.54      0.54      0.54        45
weighted avg       0.80      0.80      0.80        45

Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best parameters for Random Forest: {'max_depth': None, 'min_samples_leaf': 1,
↪  'min_samples_split': 10, 'n_estimators': 200}
Random Forest (Best) - Accuracy: 0.8222222222222222
Random Forest (Best) - Classification Report:
               precision    recall  f1-score   support

           0       0.82      0.78      0.80        18
           1       0.82      0.88      0.85        26
           2       0.00      0.00      0.00         1

    accuracy                           0.82        45
   macro avg       0.55      0.55      0.55        45
weighted avg       0.80      0.82      0.81        45

Gradient Boosting - Accuracy: 0.8444444444444444
Gradient Boosting - Classification Report:
               precision    recall  f1-score   support

           0       0.93      0.78      0.85        18
           1       0.83      0.92      0.87        26
           2       0.00      0.00      0.00         1

    accuracy                           0.84        45
   macro avg       0.59      0.57      0.57        45
weighted avg       0.85      0.84      0.84        45
```

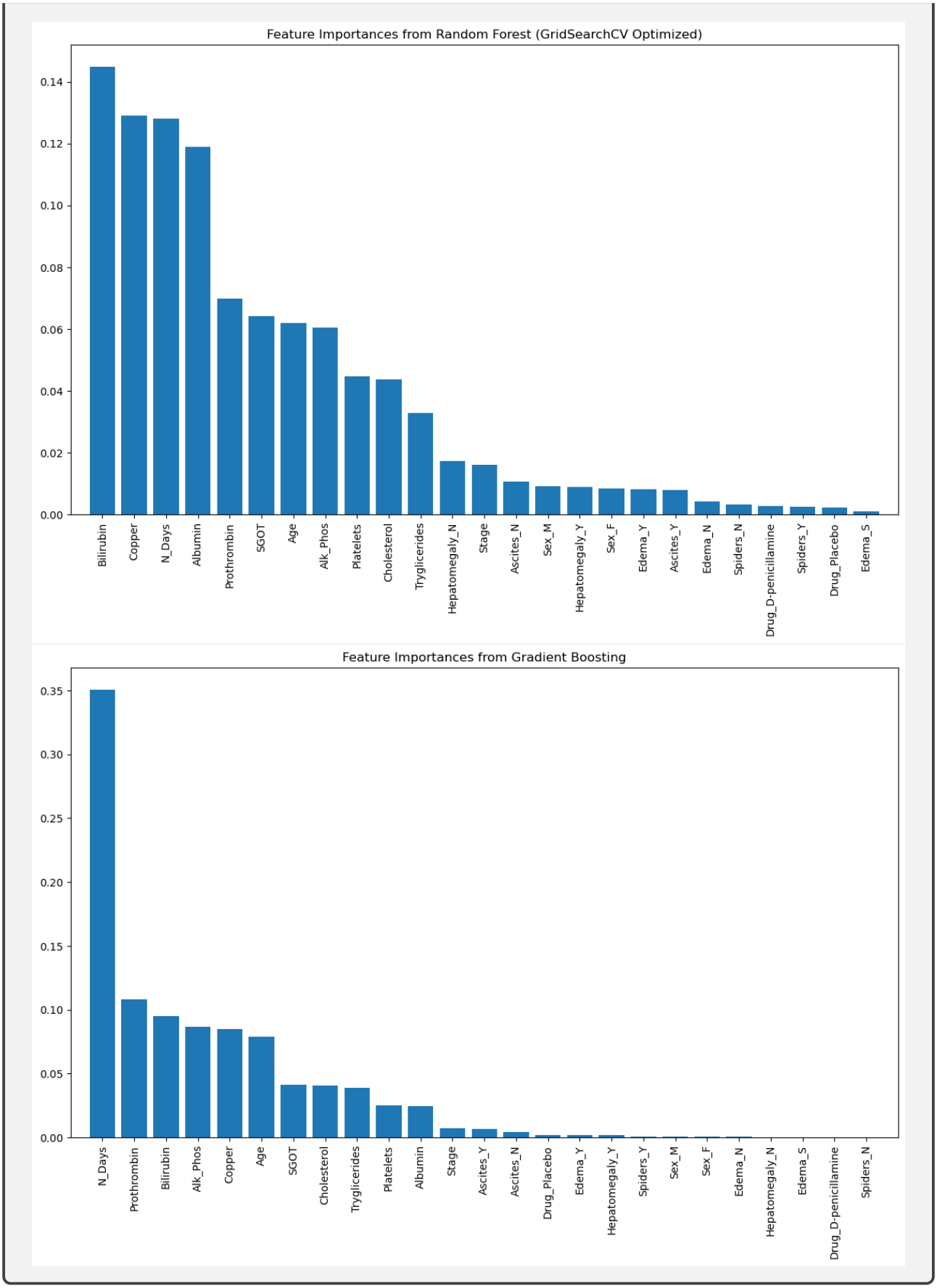
Cell 03

```
# Ans 3 - Best model
# Using the best model (Gradient Boosting) for prediction on the test set
best_model = GradientBoostingClassifier(n_estimators=100, random_state=42)
best_model.fit(X, y)

# Preprocessing the test set
X_test = test_data.drop(['Status'], axis=1)

# Standardising
X_test = scaler.transform(X_test)


# Making predictions on the test set
y_test_pred = best_model.predict(X_test)

status_mapping = {0: 'D', 1: 'C', 2: 'CL'}
y_test_pred_mapped = [status_mapping[pred] for pred in y_test_pred]


# Saving the predictions
predictions = pd.DataFrame({'testID': test_id, 'Status': y_test_pred_mapped})
predictions.to_csv('test_predictions.csv', index=False)

print("Predictions saved to 'test_predictions.csv'.")
```

```
Predictions saved to 'test_predictions.csv'.
```

Cell 04

```
# Ans4 - Feature importance
# Feature Importance from Random Forest
feature_importances_rf = rf_best_model.feature_importances_
features = train_data.drop('Status', axis=1).columns
indices_rf = np.argsort(feature_importances_rf)[::-1]

plt.figure(figsize=(14, 8))
plt.title("Feature Importances from Random Forest (GridSearchCV Optimized)")
plt.bar(range(X_train.shape[1]), feature_importances_rf[indices_rf], align="center")
plt.xticks(range(X_train.shape[1]), features[indices_rf], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()


# Feature Importance from Gradient Boosting
feature_importances_gb = gb_model.feature_importances_
indices_gb = np.argsort(feature_importances_gb)[::-1]

plt.figure(figsize=(14, 8))
plt.title("Feature Importances from Gradient Boosting")
plt.bar(range(X_train.shape[1]), feature_importances_gb[indices_gb], align="center")
plt.xticks(range(X_train.shape[1]), features[indices_gb], rotation=90)
plt.xlim([-1, X_train.shape[1]])
plt.show()
```


Feature Importances from Random Forest (GridSearchCV Optimized)
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Bilirubin
Copper
N_Days
Albumin
Prothrombin
SGOT
Age
Alk_Phos
Platelets
Cholesterol
Tryglicerides
Hepatomegaly_N
Stage
Ascites_N
Sex_M
Hepatomegaly_Y
Sex_F
Edema_Y
Ascites_Y
Edema_N
Spiders_N
Drug_D-penicillamine
Spiders_Y
Drug_Placebo
Edema_S
Feature Importances from Gradient Boosting
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
N_Days
Prothrombin
Bilirubin
Alk_Phos
Copper
Age
SGOT
Cholesterol
Tryglicerides
Platelets
Albumin
Stage
Ascites_Y
Ascites_N
Drug_Placebo
Edema_Y
Hepatomegaly_Y
Spiders_Y
Sex_M
Sex_F
Edema_N
Hepatomegaly_N
Edema_S
Drug_D-penicillamine
Spiders_N

# Report on Machine Learning for Predicting Survival Status of Liver Cirrhosis Patients

## Introduction:

This report outlines the process and results of developing machine learning models to predict the survival status of patients with liver cirrhosis. The dataset, sourced from a Mayo Clinic study on primary biliary cirrhosis (PBC) of the liver, consists of clinical features used to predict survival states: death (D), censored (C), and censored due to liver transplantation (CL).

## Question 1: Data Exploration and Pre-processing:

### (a) Dataset Size:

The dataset comprises:

- **Training Set:** 224 rows and 20 columns
- **Test Set:** 88 rows and 20 columns.

```
Train dataset size:(224, 20)
Test dataset size:(88, 20)
```

### (b) Feature Types and Missing Values:

**Training Set:**

```
Train dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 224 entries, 0 to 223
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   trainID        224 non-null    int64
 1   N_Days         224 non-null    int64
 2   Status         224 non-null    object
 3   Drug           224 non-null    object
 4   Age            224 non-null    int64
 5   Sex            224 non-null    object
 6   Ascites        224 non-null    object
 7   Hepatomegaly   224 non-null    object
 8   Spiders        224 non-null    object
 9   Edema          224 non-null    object
 10  Bilirubin      224 non-null    float64
 11  Cholesterol    201 non-null    float64
 12  Albumin        224 non-null    float64
 13  Copper         222 non-null    float64
 14  Alk_Phos       224 non-null    float64
 15  SGOT           224 non-null    float64
 16  Tryglicerides  200 non-null    float64
 17  Platelets      221 non-null    float64
 18  Prothrombin    224 non-null    float64
 19  Stage          224 non-null    int64
dtypes: float64(9), int64(4), object(7)
memory usage: 35.1+ KB
```

```
Missing values in Train dataset
Cholesterol      23
Copper            2
Tryglicerides    24
Platelets         3
dtype: int64
```

- Contains a mix of categorical and numerical features.
- **Missing values in columns:** Cholesterol (23), Copper (2), Triglycerides (24), and Platelets (3).

**Test Set:**

```
Test dataset info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 88 entries, 0 to 87
Data columns (total 20 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   testID         88 non-null     int64
 1   N_Days         88 non-null     int64
 2   Status         0 non-null      float64
 3   Drug           88 non-null     object
 4   Age            88 non-null     int64
 5   Sex            88 non-null     object
 6   Ascites        88 non-null     object
 7   Hepatomegaly   88 non-null     object
 8   Spiders        88 non-null     object
 9   Edema          88 non-null     object
 10  Bilirubin      88 non-null     float64
 11  Cholesterol    83 non-null     float64
 12  Albumin        88 non-null     float64
 13  Copper         88 non-null     int64
 14  Alk_Phos       88 non-null     float64
 15  SGOT           88 non-null     float64
 16  Tryglicerides  82 non-null     float64
 17  Platelets      87 non-null     float64
 18  Prothrombin    88 non-null     float64
 19  Stage          88 non-null     int64
dtypes: float64(9), int64(5), object(6)
memory usage: 13.9+ KB
```

```
Missing values in Test dataset

Status           88
Cholesterol       5
Tryglicerides     6
Platelets         1
dtype: int64
```

- Contains similar types of features as the training set.
- **Missing values in columns:** Status (88), Cholesterol (5), Triglycerides (6), and Platelets (1).

**(c) Handling Missing Values:**

Missing values were filled using the mean value of each respective column to maintain data integrity.

### (d) Encoding Categorical Features:

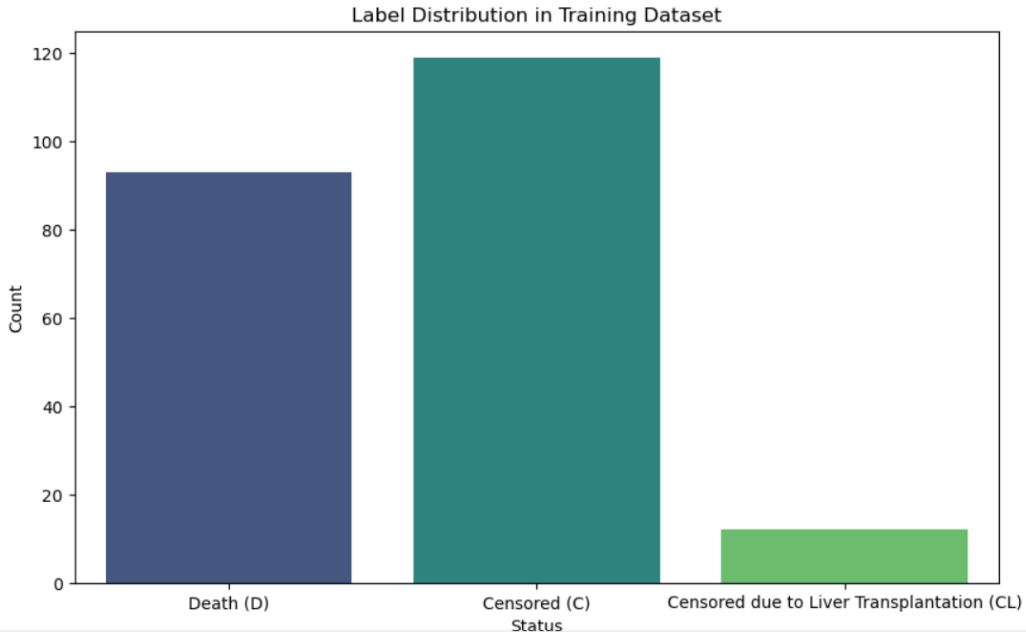
```
Info: Label encoding applied to 'Status' and one-hot encoding applied to other categorical features.
```

- The 'Status' column was label-encoded (D: 0, C: 1, CL: 2).
- Other categorical features were one-hot encoded to convert them into numerical values.

### (e) Label Distribution:

The label distribution in the training set was found to be imbalanced:

- **Death (D):** 93
- **Censored (C):** 119
- **Censored due to Liver Transplantation (CL):** 12




## Question 2: Supervised Machine Learning Models:

### (a) Model Creation and Evaluation:

Three machine learning models were developed and evaluated: Logistic Regression, Random Forest, and Gradient Boosting.

### Model 1 - Logistic Regression:

```
Logistic Regression - Accuracy: 0.8
Logistic Regression - Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.72      0.76        18
           1       0.82      0.88      0.85        26
           2       0.00      0.00      0.00         1

    accuracy                           0.80        45
   macro avg       0.54      0.54      0.54        45
weighted avg       0.80      0.80      0.80        45
```

- **Accuracy:** 80%
- **Classification Report:** Precision, recall, and F1-scores varied across classes with the highest performance for 'Censored' and lowest for 'Censored due to Liver Transplantation'.

**Model 2 - Random Forest (with Hyperparameter Tuning):**

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Best parameters for Random Forest: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
Random Forest (Best) - Accuracy: 0.8222222222222222
Random Forest (Best) - Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.78      0.80        18
           1       0.82      0.88      0.85        26
           2       0.00      0.00      0.00         1

    accuracy                           0.82        45
   macro avg       0.55      0.55      0.55        45
weighted avg       0.80      0.82      0.81        45
```

- **Best Parameters:** {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 200}
- **Accuracy:** 82.22%
- **Classification Report:** Balanced performance across 'Death' and 'Censored', but poor performance for 'Censored due to Liver Transplantation'.

**Model 3 - Gradient Boosting:**

```
Gradient Boosting - Accuracy: 0.8444444444444444
Gradient Boosting - Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.78      0.85        18
           1       0.83      0.92      0.87        26
           2       0.00      0.00      0.00         1

    accuracy                           0.84        45
   macro avg       0.59      0.57      0.57        45
weighted avg       0.85      0.84      0.84        45
```

- **Accuracy:** 84.44%
- **Classification Report:** Highest overall accuracy with better precision and recall for the majority classes, though still struggling with minority class 'Censored due to Liver Transplantation'.

**Explanation and Justification:**
The results indicate that the Logistic Regression model might be underfitted as it shows lower accuracy and struggles with capturing complex relationships in the data. On the other hand, the Random Forest and Gradient Boosting models exhibit relatively higher accuracy and balanced performance, suggesting they are not overfitted. Cross-validation and hyperparameter tuning were employed to mitigate overfitting. However, the poor performance for the minority class ('Censored due to Liver Transplantation') in all models indicates potential challenges with data imbalance rather than overfitting.

**(b) Model Design Decisions:**

**(i) Logistic Regression**

**Design Decision:** Logistic Regression was selected for its simplicity and interpretability.

**Justification:**

- **Simplicity:** It is straightforward to implement and understand.
- **Baseline Performance:** Provides a quick benchmark to compare against more complex models.
- **Efficiency:** Computationally efficient and effective for linear relationships between features.

**(ii) Random Forest**

**Design Decision**: Random Forest was chosen for its robustness and ability to handle complex data structures. Hyperparameter tuning was performed using GridSearchCV.

**Justification:**

- **Complex Interactions:** Captures non-linear relationships and feature interactions.
- **Feature Importance:** Identifies important features contributing to predictions.
- **Robustness:** Reduces overfitting by averaging multiple decision trees.
- **Optimization:** GridSearchCV optimized parameters such as the number of trees and tree depth, enhancing model performance.

**(iii) Gradient Boosting**

**Design Decision**: Gradient Boosting was selected for its strong performance and iterative learning process.

**Justification:**

- **High Accuracy:** Combines multiple weak learners to form a strong model.
- **Bias-Variance Trade-off:** Iteratively minimizes errors, balancing bias and variance.
- **Flexibility:** Adapts well to various types of data and distributions.
- **Optimization:** Setting n_estimators to 100 improved performance, with potential for further tuning.

**(c) Hyperparameter Optimization**

**(i) Logistic Regression**

- **Hyperparameters Optimized**: None
- **Explanation:** Logistic Regression was used as a baseline model to provide a straightforward and interpretable benchmark. Given its simplicity, no hyperparameter optimization was performed.

**(ii) Random Forest**

- **Hyperparameters Optimized:**
  - n_estimators: Number of trees in the forest.
  - max_depth: Maximum depth of each tree.
  - min_samples_split: Minimum number of samples required to split an internal node.
- **min_samples_leaf:** Minimum number of samples required to be at a leaf node.
- **Explanation:** GridSearchCV was used to optimize these parameters to improve model performance and prevent overfitting. The optimization aimed to balance the model's complexity and accuracy by finding the best combination of tree count and tree depth, as well as the minimum number of samples for splitting and leaf nodes.

**(iii) Gradient Boosting**

- **Hyperparameters Optimized**: n_estimators: Number of boosting stages.
- **Explanation:** The number of estimators was set to 100 based on common practice to enhance model performance. Gradient Boosting builds an ensemble of weak learners iteratively and optimizing n_estimators help in achieving a balance between underfitting and overfitting. Further hyperparameter tuning, such as learning rate and max depth, could be explored to fine-tune the model further.

**(d) Handling Label Imbalance**

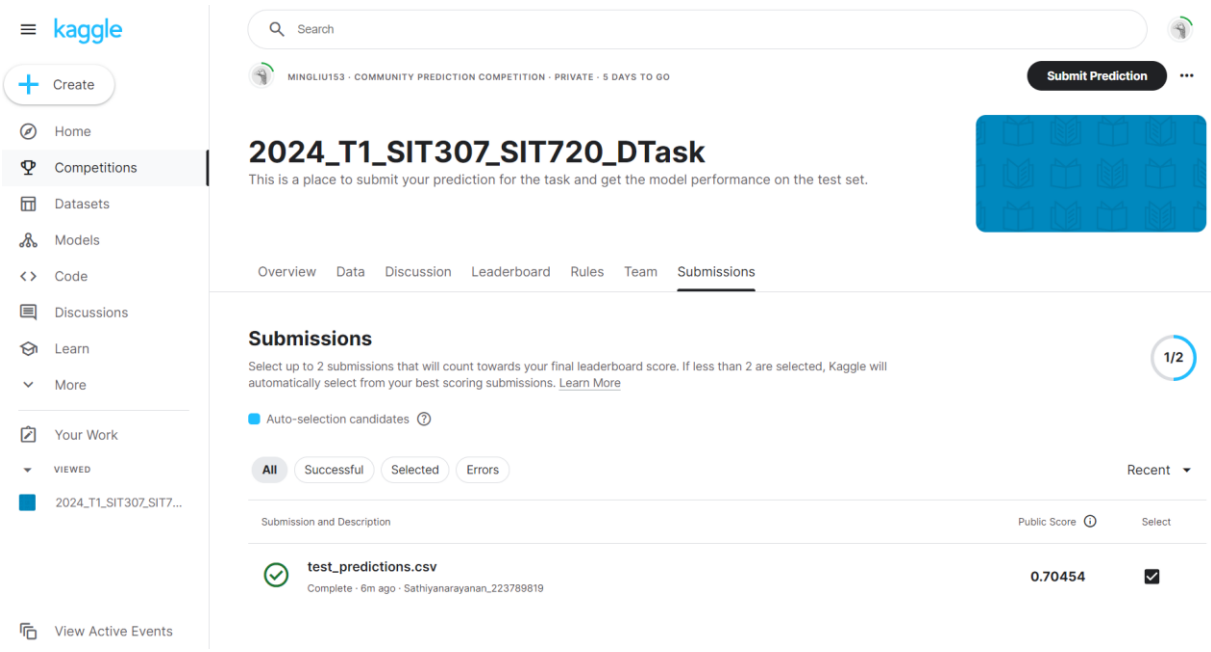
To address label imbalance, we can use oversampling techniques like SMOTE (Synthetic Minority Over-sampling Technique) to increase minority class instances or undersampling to reduce majority class instances. Alternatively, adjusting class weights in the model can help the algorithm pay more attention to the minority class. Lastly, using ensemble methods like balanced random forests can also mitigate the impact of label imbalance.

**(e) Model Recommendation**

I recommend the Gradient Boosting model due to its highest accuracy (84.44%) and balanced performance across most classes. Despite some challenges with the minority class, it consistently outperformed Logistic Regression and Random Forest in our evaluations. Its ability to iteratively correct errors and handle complex data structures makes it the best choice for this task.

## Question 3: Prediction on Test Set:

Using the best model (Gradient Boosting), predictions were made on the test set. The predicted status labels were mapped back to their categorical values and saved to 'test_predictions.csv'. The Kaggle submission score was **0.70454**, indicating a decent performance of the model in predicting the survival status.

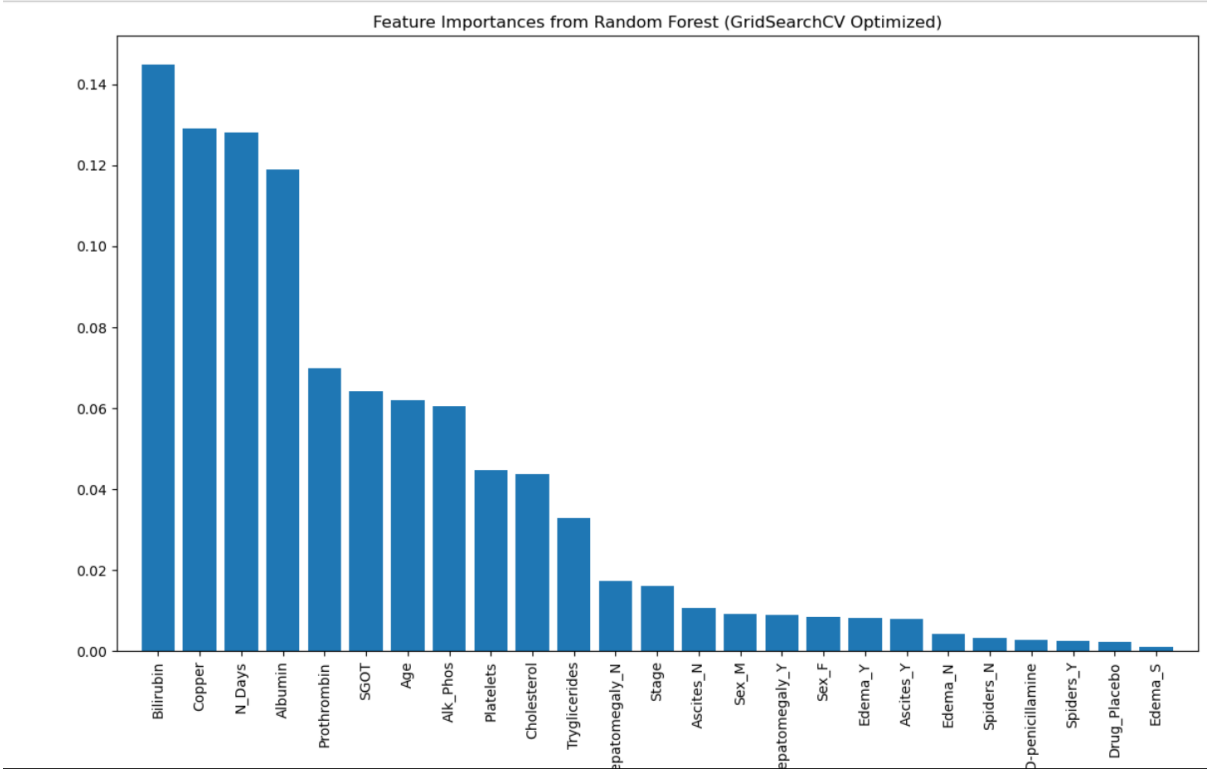



**Question 4: Feature Importance Analysis:**

Feature importance was analyzed using both Random Forest and Gradient Boosting models:

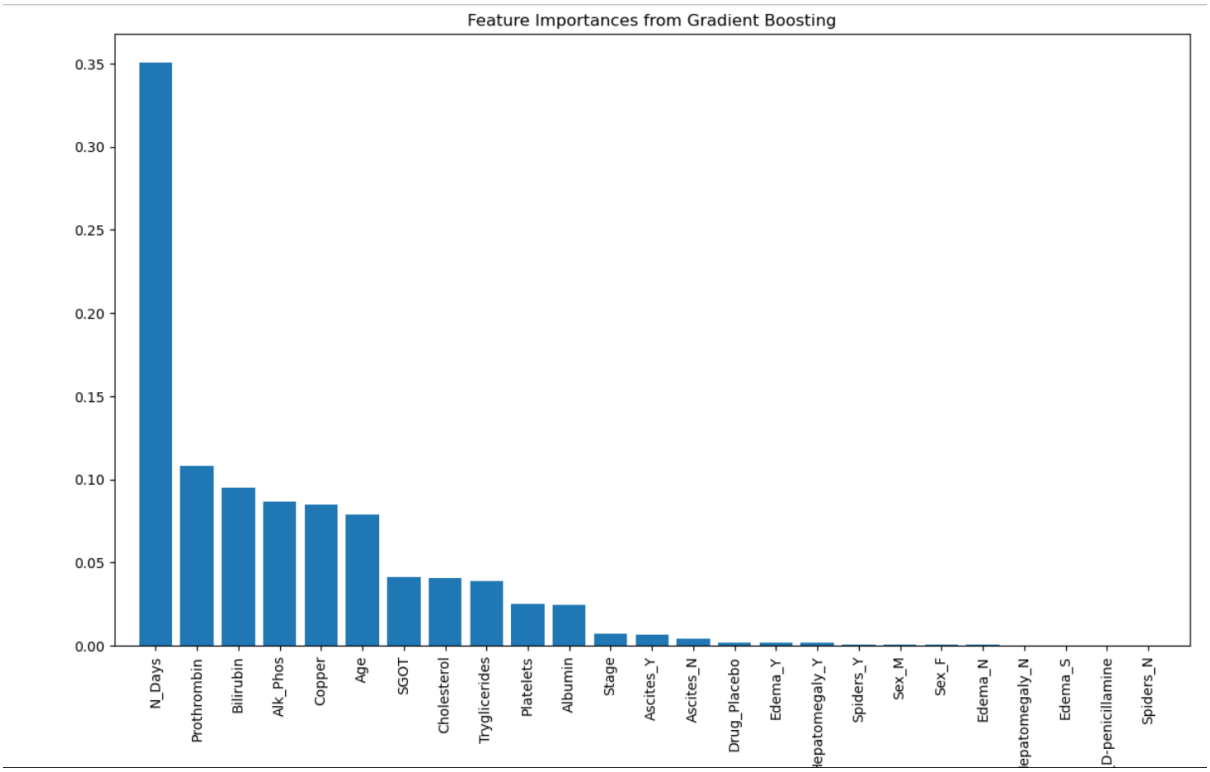
**(i) Random Forest:**

- Top important features included Bilirubin, Copper, and N_Days, among others.
- Visualized importance showed that features like Bilirubin and Copper had higher importance compared to others.



**(ii) Gradient Boosting:**

- N_Days emerged as the most important feature by a significant margin.
- Other important features included Prothrombin and Bilirubin.




## Conclusion:

The analysis and modelling efforts provide valuable insights into the factors influencing the survival status of patients with liver cirrhosis. The Gradient Boosting model emerged as the best performer, and feature importance analysis highlighted key clinical features that could guide future research and treatment strategies.