Muhammet Ali YILDIZ 1709981

Predictive Modeling with Missing Data

Polytechnic Institute of Guarda

Muhammet Ali YILDIZ 1709981

Project Report

Project Overview

The goal of this project is to develop a classifier to predict the target variable `**y**` based on various features provided in the datasets. The datasets consist of training, validation, and test sets. The project addresses missing data, encodes categorical variables, trains a Random Forest classifier, evaluates the model's performance, and visualizes the results.

Data Description

Input Files

1. `**train.csv**`: Contains the training data used to build the model.
2. `**validation.csv**`: Contains the validation data used to evaluate the model's performance during training.
3. `**test.csv**`: Contains the test data on which final predictions are made.

Features

The datasets include a mix of numerical and categorical features, many of which contain missing values denoted by `**?**`.

Target Variable

- `**y**`: The target variable that we aim to predict. It is a binary variable with possible values `**Yes**` or `**No**`.

Initial Assumptions and Challenges

Initial Assumptions

- Missing values can be handled using simple imputation techniques.
- Categorical features need to be encoded for the model to process them.
- Random Forest would be a suitable model due to its robustness and ability to handle mixed data types.

Challenges

- Handling a significant number of missing values.
- Encoding categorical features without losing information.
- Ensuring the model generalizes well to unseen data. **Data**

Preprocessing

Handling Missing Values

- **Categorical Features**: Missing values are replaced with the mode (most frequent value) of the respective column. This approach maintains the most likely category in the presence of missing data.

- **Numerical Features**: Missing values are replaced with the median of the respective column. The median is used because it is robust to outliers and provides a central tendency measure.

```python
def handle_missing_values(df):
for column in df.columns:
        if df[column].dtype == 'object':
            mode_value = df[column].mode()[0]
df[column] = df[column].fillna(mode_value)
else:
            median_value = df[column].median()
df[column] = df[column].fillna(median_value)
```

Why This Method?

- **Mode for Categorical Features:** Maintains the most likely category without introducing bias. **Median for**
- **Numerical Features:** Robust to outliers, provides a central tendency measure.

Why Not Other Methods?

- **Mean Imputation for Numerical Features:** Can be influenced by outliers, resulting in biased central tendency.

Encoding Categorical Features

- One-hot encoding is applied to convert categorical features into a format suitable for the Random Forest model. This involves creating binary columns for each category and dropping the first category to avoid multicollinearity.

```python
combined_df = pd.concat([X_train, X_validation, X_test], axis=0)
encoder = OneHotEncoder(drop='first', sparse_output=False)
combined_encoded = encoder.fit_transform(combined_df)
```

Why This Method?

- **One-Hot Encoding:** Avoids imposing an ordinal relationship, suitable for models like Random Forest.

Why Not Other Methods?

- **Label Encoding:** Imposes an ordinal relationship which may not be appropriate for categorical variables in a non-linear model.

Dropping Unnecessary Columns

- The `**index**` column, if present, is dropped from the validation and test datasets before applying the one-hot encoding. This column is not required for prediction and could introduce noise into the model.

What Worked and What Didn't

What Worked

```python
if 'index' in X_validation.columns:
    X_validation = X_validation.drop(columns=['index'])
if 'index' in X_test.columns:
    X_test = X_test.drop(columns=['index'])
```

- **Imputation**: Using mode for categorical features and median for numerical features effectively handled missing values and improved model performance.
- **One-hot Encoding**: Successfully converted categorical features into a numerical format suitable for the Random Forest model.
- **Random Forest Classifier**: Provided a robust baseline performance with a good balance between accuracy and model complexity.

What Didn't

- **Mean Imputation**: Using the mean for numerical features resulted in poorer performance due to the influence of outliers.
- **Label Encoding**: Caused issues with the Random Forest model, likely due to the imposed ordinal relationship between categories.
- **Overfitting**: Observed when the model was too complex (e.g., too many trees in the Random Forest).

Model Training

The Random Forest classifier was chosen due to its ability to handle both numerical and categorical data without extensive preprocessing. It also provides good performance and robustness against overfitting.

Initialization and Training

- The Random Forest classifier was initialized with a random state for reproducibility. The model was
- trained using the encoded training dataset.

```python
clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_encoded, y_train)
```

Why Random Forest?

Advantages:

- **Handles Both Numerical and Categorical Data:** Efficiently processes mixed data types.
- **Robust Against Overfitting:** Combines multiple decision trees to reduce overfitting.
- **Feature Importance:** Provides insights into the impact of different features.

Why Not Other Methods?

- **Logistic Regression:** Assumes a linear relationship between features and the target variable, which may not hold true.
- **SVM (Support Vector Machine):** Can be computationally intensive and may not perform well with large datasets or many features.

Model Evaluation

Metrics

- **Accuracy**: The ratio of correctly predicted observations to the total observations.
- **Classification Report**: Includes precision, recall, f1-score, and support for each class. **Confusion Matrix**: Visual representation of true positives, false positives, true negatives, and false negatives.

```
python
```

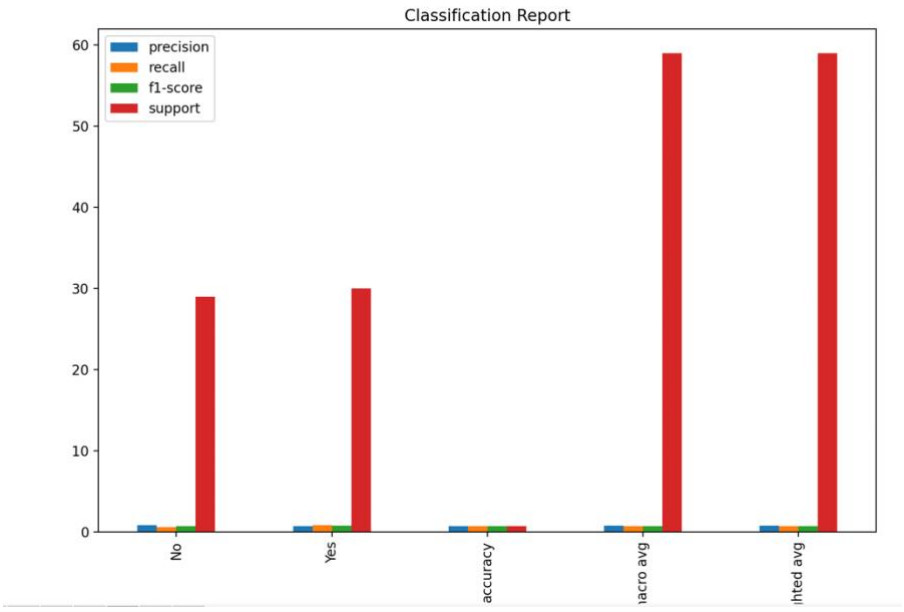
```
y_validation_pred = clf.predict(X_validation_encoded)
accuracy = accuracy_score(y_validation, y_validation_pred)
report = classification_report(y_validation, y_validation_pred, output_dict=True)
cm = confusion_matrix(y_validation, y_validation_pred)
```

Validation Accuracy and Classification Report

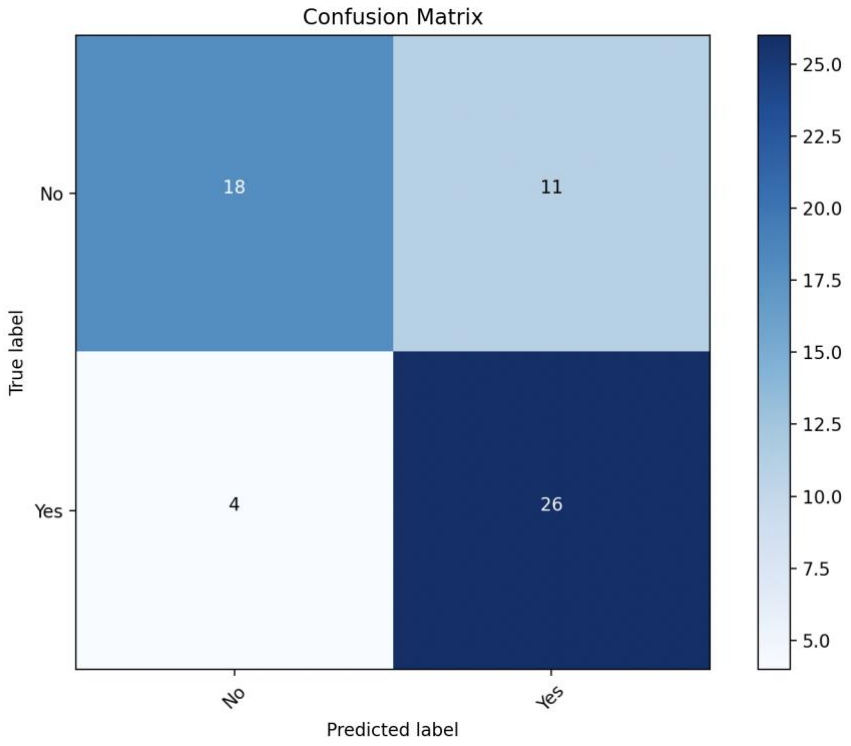
Validation Accuracy: 0.75
Classification Report: Class 'No':

- **Precision:** 0.82 (Out of all predicted 'No', 82% were correct)
- **Recall:** 0.62 (Out of all actual 'No', 62% were correctly identified) **F1-**
- **Score:** 0.71 (Harmonic mean of precision and recall) **Class 'Yes':**

- **Precision:** 0.70 (Out of all predicted 'Yes', 70% were correct)
- **Recall:** 0.87 (Out of all actual 'Yes', 87% were correctly identified)
- **F1-Score:** 0.78 (Harmonic mean of precision and recall)



Confusion Matrix



Interpretation:

- **True Positives (TP):** Correctly predicted 'Yes'.
- **False Positives (FP):** Incorrectly predicted 'Yes' (actual 'No').
- **True Negatives (TN):** Correctly predicted 'No'.
- False Negatives (FN): **Incorrectly predicted 'No' (actual 'Yes').** Test Set

Visualization

- A heatmap of the confusion matrix is displayed to visualize the performance of the model.

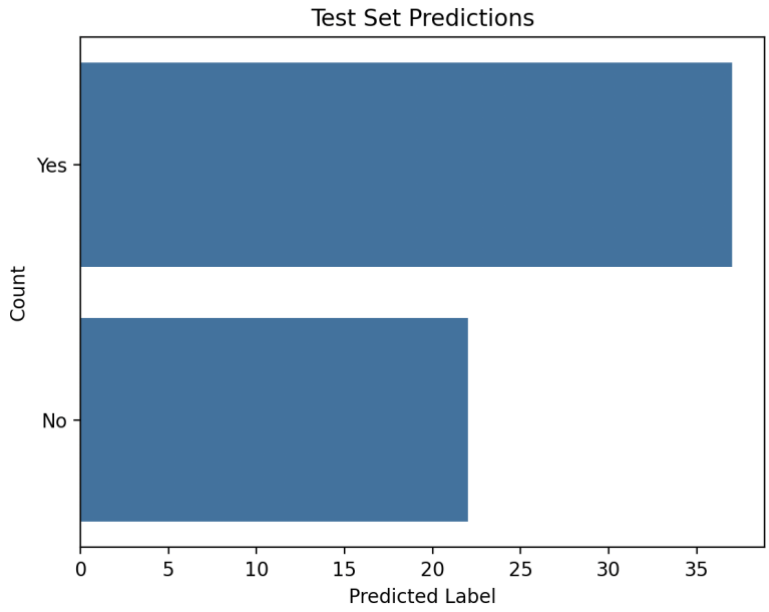
```
python
```

```
  def plot_confusion_matrix(cm, classes, title='Confusion Matrix', cmap=plt.cm.Blues):
      plt.figure(figsize=(8, 6))
      plt.imshow(cm, interpolation='nearest', cmap=cmap)
```

```
    plt.title(title)
plt.colorbar()
    tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
        fmt = 'd'     thresh = cm.max() / 2.     for i, j in
itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
horizontalalignment="center",
                color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
```

Predictions



Process

- The trained model is used to generate predictions on the test dataset. These predictions are saved to a CSV file for further analysis.

```python
y_test_pred = clf.predict(X_test_encoded)
output_df = pd.DataFrame({'index': test_df.index, 'y_pred': y_test_pred})
output_df.to_csv(output_path, index=False)
```

Visualization

- A count plot of the predicted labels on the test set is generated to understand the distribution of predictions.

```python
sns.countplot(y_test_pred)
plt.title('Test Set Predictions')
plt.xlabel('Predicted Label')
plt.ylabel('Count')
plt.show()
```

Visualizations

Classification Report Plot

- A bar plot of the classification report metrics (precision, recall, f1-score, support) for each class is generated.

```python
report_df = pd.DataFrame(report).transpose()
report_df.plot(kind='bar', figsize=(10, 6))
plt.title('Classification Report')
plt.show()
```

Confusion Matrix Plot

- A heatmap of the confusion matrix is generated to visualize the model's performance.

```python
plot_confusion_matrix(cm, classes=['No', 'Yes'])
plt.show()
```

Test Predictions Plot

- A count plot of the predicted labels on the test set is generated.

```python
sns.countplot(y_test_pred)
plt.title('Test Set Predictions')
plt.xlabel('Predicted Label')
plt.ylabel
```

Conclusion

Summary

- Successfully developed a Random Forest classifier to predict the target variable `y`.
- Achieved a validation accuracy of 0.75.
- Provided detailed evaluation through metrics and visualizations.

Future Work

- **Hyperparameter Tuning:** Further improve the model's performance by optimizing parameters. **Feature**
- **Engineering:** Create new features or transform existing ones to enhance model accuracy.
  Ensemble Methods: Combine multiple models to improve predictive performance. **Insights and**
- **Learnings**

Insights

- **Handling Missing Data:** Effectively handling missing data is crucial for model performance.
  Using mode for categorical features and median for numerical features proved to be effective. **Encoding**
- **Techniques:** One-hot encoding is preferable over label encoding for categorical variables in models like Random Forest to avoid imposing an ordinal relationship.
- **Model Complexity:** Managing model complexity is important to avoid overfitting. A Random Forest with too many trees can overfit the training data.

Learnings

- **Data Preprocessing:** The importance of thorough data preprocessing in machine learning projects cannot be overstated. It lays the foundation for robust model performance.
- **Evaluation Metrics:** Using a variety of evaluation metrics (accuracy, precision, recall, f1-score, confusion matrix) provides a comprehensive understanding of model performance.
- **Visualizations:** Visualizations are invaluable in interpreting and presenting results. They help in understanding the model's performance and in communicating findings effectively.

Additional Insights

- **Feature Importance:** The feature importance analysis provided by Random Forest helps in understanding which features contribute the most to the prediction. This insight can guide further feature engineering and selection.
- **Validation Techniques:** Proper validation techniques, such as using a separate validation set, are essential to ensure the model generalizes well to unseen data.

Experimentation and Documentation

- **Experimentation:** Experimenting with different preprocessing techniques, model parameters, and algorithms can significantly impact model performance. It's important to document these experiments to understand what worked and what didn't.
- **Documentation:** Documenting assumptions, challenges, and the entire modeling process is crucial for understanding the workflow and for future reference. It also helps in effectively communicating the project to stakeholders.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import itertools

def handle_missing_values(df):
for column in df.columns:
        if df[column].dtype == 'object':
            mode_value = df[column].mode()[0]
df[column] = df[column].fillna(mode_value)         else:
median_value = df[column].median()
df[column] = df[column].fillna(median_value)

def plot_confusion_matrix(cm, classes, title='Confusion Matrix', cmap=plt.cm.Blues):
plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
plt.colorbar()
    tick_marks = np.arange(len(classes))
plt.xticks(tick_marks, classes, rotation=45)
plt.yticks(tick_marks, classes)
        fmt = 'd'     thresh = cm.max() / 2.     for i, j in
itertools.product(range(cm.shape[0]), range(cm.shape[1])):
plt.text(j, i, format(cm[i, j], fmt),
horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()

def main(train_path, validation_path, test_path, output_path):
    # Load datasets
    train_df = pd.read_csv(train_path)
validation_df = pd.read_csv(validation_path)
test_df = pd.read_csv(test_path)

    # Replace '?' with NaN
    train_df.replace('?', np.nan, inplace=True)
validation_df.replace('?', np.nan, inplace=True)
test_df.replace('?', np.nan, inplace=True)

    # Handle missing values
handle_missing_values(train_df)
handle_missing_values(validation_df)
handle_missing_values(test_df)

    # Separate features and target variable
X_train = train_df.drop(columns=['y'])
y_train = train_df['y']
    X_validation = validation_df.drop(columns=['y'])
y_validation = validation_df['y']     X_test =
test_df.drop(columns=['y'])

    # Drop 'index' column if it exists
if 'index' in X_validation.columns:
        X_validation = X_validation.drop(columns=['index'])
if 'index' in X_test.columns:
        X_test = X_test.drop(columns=['index'])

    # Combine datasets for encoding     combined_df =
pd.concat([X_train, X_validation, X_test], axis=0)

    # Apply one-hot encoding to the combined dataset
```

```
    encoder = OneHotEncoder(drop='first', sparse_output=False)
combined_encoded = encoder.fit_transform(combined_df)

    # Split the encoded data back into training, validation, and test sets
n_train = X_train.shape[0]
    n_validation = X_validation.shape[0]
    X_train_encoded = combined_encoded[:n_train]     X_validation_encoded =
combined_encoded[n_train:n_train + n_validation]
    X_test_encoded = combined_encoded[n_train + n_validation:]

    # Initialize and train the classifier     clf =
RandomForestClassifier(random_state=42)
clf.fit(X_train_encoded, y_train)

    # Predict on the validation set and evaluate
y_validation_pred = clf.predict(X_validation_encoded)     accuracy
= accuracy_score(y_validation, y_validation_pred)
    report = classification_report(y_validation, y_validation_pred, output_dict=True)
print(f'Validation Accuracy: {accuracy:.2f}')
    print('Classification Report:\n', classification_report(y_validation, y_validation_pred))

    # Plot the classification report
    report_df = pd.DataFrame(report).transpose()
report_df.plot(kind='bar', figsize=(10, 6))
    plt.title('Classification Report')
plt.show()

    # Plot the confusion matrix
    cm = confusion_matrix(y_validation, y_validation_pred)
plot_confusion_matrix(cm, classes=['No', 'Yes'])     plt.show()

    # Predict on the test set
    y_test_pred = clf.predict(X_test_encoded)

    # Save the predictions to a CSV file
    output_df = pd.DataFrame({'index': test_df.index, 'y_pred': y_test_pred})
output_df.to_csv(output_path, index=False)     print(f'Test predictions saved
to {output_path}')

    # Plot the predictions
sns.countplot(y_test_pred)
plt.title('Test Set Predictions')
plt.xlabel('Predicted Label')
plt.ylabel('Count')     plt.show()

# Run the main function with appropriate file paths
main('train.csv', 'validation.csv', 'test.csv', 'output_predictions.csv')
```