Assignment

Dataset link: https://drive.google.com/file/d/1bGoIE4Z2kG5nyh-fGZAJ7LH0ki3UfmSJ/view?usp=share_link

Q1. Preprocess the dataset by handling missing values, encoding categorical variables, and scaling the
numerical features if necessary.

Ans:preprocess a dataset in Python using popular libraries like Pandas and Scikit-learn. If you could share the structure of the dataset or specific details about the features, I can provide more tailored guidance.

Here's a generic outline of how you might preprocess a dataset:

1. Handling Missing Values:

Example using Pandas:

python

Copy code

```
import        as


                'your_dataset.csv'


print             sum



   'your_column'            'your_column'                  True


   'your_categorical_column'            'your_categorical_column'         0
        True
```

2. Encoding Categorical Variables:

Example using Pandas:

python

Copy code

```
                                'nominal_categorical_column'


from                      import

   'ordinal_categorical_column'
                              'ordinal_categorical_column'
```

3. Scaling Numerical Features:

Example using Scikit-learn:

python

Copy code

```
from                      import




    'numerical_column_1'  'numerical_column_2'
                        'numerical_column_1'  'numerical_column_2'
```

Make sure to adapt these examples based on the specifics of your dataset. If you can provide more information about the dataset, I can offer more detailed and accurate guidance.

Q2. Split the dataset into a training set (70%) and a test set (30%).
Ans:o split the dataset into a training set and a test set in Python, you can use the `train_test_split` function from the Scikit-learn library. Here's an example:

python

Copy code

```
import       as
from                        import


                'your_dataset.csv'
```

```
            'target_column'       1

        'target_column'


                                                                    0.3
             42

print "Training set shape:"
print "Test set shape:"
```

In this example:

- `X` represents the feature matrix (all columns except the target variable).
- `y` represents the target variable.

Adjust the column names accordingly based on your dataset structure. The `test_size` parameter in `train_test_split` controls the proportion of the dataset allocated to the test set (in this case, 30%).

The `random_state` parameter ensures reproducibility. Setting it to a fixed value allows you to get the same split if you run the code multiple times.

Q3. Train a random forest classifier on the training set using 100 trees and a maximum depth of 10 for each
tree. Use the default values for other hyperparameters.
Ans:To train a Random Forest classifier in Python using the Scikit-learn library with the specified parameters, you can follow the example code below. Please replace `'target_column'` with the actual name of your target column:
python

Copy code

```
from                  import
from                 import
```

```
                                                    100             10
            42
```

```
print "Accuracy:"
```

In this code:

- `RandomForestClassifier` is imported from Scikit-learn to create a Random Forest classifier.
- The `n_estimators` parameter is set to 100, indicating the number of trees in the forest.
- The `max_depth` parameter is set to 10, limiting the maximum depth of each tree.
- `random_state` is set to 42 for reproducibility.

After training the classifier, predictions are made on the test set, and the accuracy of the model is calculated using the `accuracy_score` metric. Adjust the code according to your specific dataset and target column.

Q4. Evaluate the performance of the model on the test set using accuracy, precision, recall, and F1 score.
Ans:To evaluate the performance of the Random Forest classifier on the test set using accuracy, precision, recall, and F1 score, you can use the appropriate functions from Scikit-learn. Here's an example:
python

Copy code

```
from                  import
```

```
print "Accuracy:"


print "Precision:"


print "Recall:"


print "F1 Score:"
```

In this code:

- `accuracy_score`, `precision_score`, `recall_score`, and `f1_score` are functions from Scikit-learn to calculate the respective metrics.
- `y_test` contains the actual labels from the test set.
- `y_pred` contains the predicted labels from the Random Forest classifier.

Adjust the code based on your specific dataset and target column. These metrics provide a comprehensive understanding of the classifier's performance in terms of accuracy, precision, recall, and the balance between precision and recall (F1 score).

Q5. Use the feature importance scores to identify the top 5 most important features in predicting heart
disease risk. Visualise the feature importances using a bar chart.
Ans:o identify the top 5 most important features in predicting heart disease risk using the feature importance scores from the trained Random Forest classifier and visualize them with a bar chart, you can use the following example code:
python

Copy code

```
import                  as
import        as
```

```
                                'Feature'                    'Importance'


                                                        'Importance'
False

                                    5


          10  6
    'Importance'    'Feature'                              'viridis'
'Top 5 Most Important Features for Heart Disease Prediction'
 'Importance'
 'Feature'
```

In this code:

- `feature_importances` contains the importance scores of each feature as calculated by the Random Forest classifier.
- A DataFrame (`feature_importance_df`) is created to associate feature names with their importances.
- The DataFrame is sorted in descending order based on importance.
- The top 5 most important features are selected.
- A bar chart is created using the Seaborn library to visualize the feature importances.

Make sure to replace `random_forest` with the actual name of your trained Random Forest classifier, and adjust column names based on your dataset. This code assumes that your features are in a DataFrame with column names accessible through `X_train.columns`.

Note: Visualization libraries such as Matplotlib and Seaborn need to be installed for this code to work. If you haven't installed them yet, you can do so using:

Q6. Tune the hyperparameters of the random forest classifier using grid search or random search. Try

different values of the number of trees, maximum depth, minimum samples split, and minimum samples

leaf. Use 5-fold cross-validation to evaluate the performance of each set of hyperparameters.

Ans:To tune the hyperparameters of the Random Forest classifier using grid search or random search with 5-fold cross-validation, you can use Scikit-learn's `GridSearchCV` or `RandomizedSearchCV` along with the `RandomForestClassifier`. Here's an example using `GridSearchCV`:

python

Copy code

```
from                          import

from                    import

from                  import

from                          import
```

```
 'n_estimators'    50  100  150

 'max_depth'    5  10  15

 'min_samples_split'    2  5  10

 'min_samples_leaf'    1  2  4
```

```
                                        5            1
```

```
print "Best Hyperparameters:"

print "Best Accuracy:"
```

In this example:

- The parameter grid includes different values for the number of trees (`n_estimators`), maximum depth (`max_depth`), minimum samples split (`min_samples_split`), and minimum samples leaf (`min_samples_leaf`).
- `make_scorer(accuracy_score)` is used to optimize for accuracy during the grid search.
- `StratifiedKFold(n_splits=5)` is used for 5-fold cross-validation, which is suitable for classification tasks.

Adjust the parameter grid and search space according to your specific requirements. You can also explore `RandomizedSearchCV` for a more efficient search when the search space is large.

Q7. Report the best set of hyperparameters found by the search and the corresponding performance

metrics. Compare the performance of the tuned model with the default model.

Ans:To report the best set of hyperparameters found by the search and the corresponding performance metrics, and to compare the performance of the tuned model with the default model, you can use the following code:

python

Copy code

```
from                  import

from                 import


from                        import
```

```
from                          import

                                                                  42

print "Default Model Metrics:"
```

```
print "Accuracy:"

print "Precision:"

print "Recall:"

print "F1 Score:"
```



```
 'n_estimators'    50  100  150

 'max_depth'    5  10  15

 'min_samples_split'    2  5  10

 'min_samples_leaf'    1  2  4
```

```
        'accuracy'                              5           1
```

```
print "\nBest Hyperparameters:"

print

print "Best Accuracy:"
```

```
print "\nTuned Model Metrics:"

print "Accuracy:"

print "Precision:"

print "Recall:"

print "F1 Score:"




print "\nPerformance Improvement:"

print "Accuracy Improvement:"

print "Precision Improvement:"

print "Recall Improvement:"

print "F1 Score Improvement:"
```

This code performs the following steps:

Trains and evaluates the default Random Forest classifier.

Conducts a grid search to find the best hyperparameters using 5-fold cross-validation.

Reports the best hyperparameters and corresponding accuracy.

Evaluates the tuned model on the test set and reports performance metrics.

Compares the performance improvements between the default and tuned models.

Make sure to replace `'target_column'` with the actual name of your target column and adjust column names based on your dataset.

Q8. Interpret the model by analysing the decision boundaries of the random forest classifier. Plot the

decision boundaries on a scatter plot of two of the most important features. Discuss the insights and

limitations of the model for predicting heart disease risk.

Ans:decision boundaries in a Random Forest classifier can provide insights into how the model makes predictions. However, visualizing decision boundaries in high-dimensional feature spaces can be challenging. For illustrative purposes, we can plot decision boundaries in a 2D space using two of the most important features.

Here's an example of how you can plot decision boundaries using the `Meshgrid` approach:

python

Copy code

```
import       as

import                  as
```

```
from                  import

            'important_feature1'
            'important_feature2'

                                                                  42

                        min     1               max     1
                        min     1               max     1
```

```
                                        0.01

                    0.01

```

```
              10  6

                      0.3       'viridis'

                                                    'viridis'
'k'    50

'Decision Boundaries of Random Forest Classifier'
```

In this example:

- `X_2d` represents a subset of the training data containing only the two most important features.
- The `Meshgrid` is used to create a grid of points in the feature space.
- The classifier is used to make predictions on each point in the grid, and the decision boundaries are visualized.

Interpretation:

- Decision boundaries in the scatter plot represent regions where the classifier assigns a particular class.
- Areas with the same color indicate regions where the classifier predicts the same class.
- The scatter plot shows how the model separates different classes based on the two selected features.

Insights and Limitations:

- Insights: Decision boundaries can reveal how the Random Forest model distinguishes between different classes in the 2D feature space. This can provide insights into the relationships between the selected features and the target variable.
- Limitations:
    - High-Dimensional Data: In real-world datasets with many features, decision boundaries become complex, and visualizing them in 2D may not capture the full complexity of the model.
    - Non-Linear Relationships: Random Forests can model non-linear relationships, but the visualization of decision boundaries may not fully capture the intricate decision-making process in higher-dimensional spaces.
    - Interpretability: While decision boundaries provide visual insights, interpreting them in a meaningful way might be challenging, especially when dealing with many features.

This analysis is a simplified representation, and for a more comprehensive understanding, you might consider advanced techniques like Partial Dependence Plots or SHAP (SHapley Additive exPlanations) values.

2 / 2