Lab Course Machine Learning

Exercise Sheet 5

Prof. Dr. Dr. Lars Schmidt-Thieme
Jung Min Choi

HiWi: Harish Malik

Submission deadline : November 21, 2024

General Instructions

1. Data should be normalized.
2. Train to Test split should be 80-20 / with Validaiton 70-15-15
3. Convert any non-numeric values to numeric values. For example you can replace a country name with an integer value or more appropriately use one-hot encoding.

1 KNN Imputation and Classification (10 points)

In this problem, we aim to implement data imputation using the K-Nearest Neighbors (KNN) algorithm. The idea is to replace missing values in a dataset by calculating the average of their K-Nearest Neighbors, where K serves as a hyperparameter. Additionally, we will use a KNN classifier to classify the data and evaluate its performance.

1. **[2 Points]** Download the *DodgerLoopGame* datasets from the following link: `https://www.timeseriesclassification.com/description.php?Dataset=DodgerLoopGame`. Load the train and test datasets, and display the number of `NaN` values in each dataset.
2. **[3 Points]** Use the `KNNImputer` from `sklearn.impute` to replace missing values in the train and test datasets. Perform grid search to tune the hyperparameter K (number of neighbors) used by the imputer. Use the same optimal K for all features. You can also use the `dist` function from `scipy.spatial.distance` to compute pairwise distances.
3. **[5 Points]** Implement a K-Nearest Neighbors classifier **without using `sklearn`**. The classifier should use majority voting and Euclidean distance as the distance metric. Perform grid search to find the optimal K for the classifier that maximizes accuracy on the validation set.

2 Decision Tree Regression Implementation (10 points)

In this problem, you will implement a Decision Tree Regression model from scratch using the Residual Sum of Squares (RSS) as the split quality criterion. You will also evaluate its performance on a given dataset. You can split the dataset only into train/test datasets.

1. **[2 Points]** Define a class for a tree node. Each node should store:
   - A split feature and threshold value (for non-leaf nodes).
   - A predicted value (for leaf nodes).
2. **[4 Points]** Implement a recursive function to build the decision tree using RSS as the quality criterion. Use a stopping criterion based on either a maximum depth or a minimum number of samples in a node.
3. **[3 Points]** Write a function to make predictions for new data instances using the trained decision tree.
4. **[1 Points]** Evaluate the model's performance on a test dataset by calculating Mean Squared Error (MSE).

Dataset: Use the **Iris dataset:** Target attribute class {Iris Setosa, Iris Versicolour, Iris Virginica}.
`https://archive.ics.uci.edu/ml/datasets/Iris`

```
TreeNode Class:
    - Attributes:
        - predicted_value: Predicted value for leaf nodes
        - split_feature: Index of feature used for splitting (non-leaf
          nodes)
        - threshold: Threshold value used for splitting
        - left: Reference to the left child node
        - right: Reference to the right child node

DecisionTreeRegressor Class:
    - Methods:
        1. fit(X, y):
            - Build the decision tree by calling _build_tree() with
              input X and target y.

        2. _build_tree(X, y, depth=0):
            - Create a TreeNode.
            - If stopping criteria are met (min samples or max depth):
                - Set the node's predicted_value to the mean of y.
                - Return the node.
            - Find the best feature and threshold by calling
              _find_best_split().
            - If no valid split is found:
                - Set the node's predicted_value to the mean of y.
                - Return the node.
            - Set the node's split_feature and threshold to the selected
               values.
            - Split X and y into left and right subsets based on the
              threshold.
            - Recursively call _build_tree() for the left and right
              subsets to construct child nodes.
            - Attach the child nodes to the current node's left and
              right attributes.
            - Return the node.

        3. _find_best_split(X, y):
            - Initialize best_rss to infinity, best_feature, and
              best_threshold to None.
            - For each feature in X:
                - Iterate through unique threshold values in the feature
                  .
                - Split y into left and right subsets based on the
                  threshold.
                - Skip the split if either subset is empty.
                - Calculate RSS for the split by calling _calculate_rss
                  ().
                - Update best_rss, best_feature, and best_threshold if
                  the current RSS is lower.
            - Return the best_feature, best_threshold, and best_rss.

        4. _calculate_rss(y_left, y_right):
            - Compute the variance of y_left and y_right.
            - Return the sum of the variances as RSS.

        5. predict(X):
            - For each sample in X:
                - Call _predict_sample() to traverse the tree from the
                  root node and return the predicted value.
            - Return an array of predicted values.

        6. _predict_sample(sample, node):
```

```
- If the node is a leaf (predicted_value is not None):
    - Return the node's predicted_value.
- Otherwise:
    - Compare the sample's value at split_feature with the
      node's threshold.
    - Traverse to the left or right child node based on the
      comparison.
    - Recursively call _predict_sample() for the child node.
```