CT-3 Mini Project MLSC - PA2312052010010 – Result and Inference

Inference document heart disease prediction using MLSC concepts

1. Import the necessary Libraries

Data Preparation

2.1 Read the Data

loaded a dataset named "heart_disease_data.csv" into a Pandas DataFrame and displayed the first five rows using the **head()** function

2.2 Check the Data Type

2.3 Remove Insignificant Variables, if its applicable.

2.4 Distribution of Variables

2.5 Missing Value Treatment

Rows and Columns:

The dataset contains 303 rows and an unspecified number of columns.

Data Information:

The info() method provides information about the dataset's columns, data types, and non-null values.

Missing Values:

The dataset has been checked for missing values using the isnull().sum() method.

Statistical Measures:

Descriptive statistics have been generated for the dataset using the describe() method. This includes measures like mean, median, minimum, maximum, and quartiles for numerical columns.

Target Variable Distribution:

The distribution of the target variable target has been printed using value_counts(). This indicates the count of each unique value in the target variable.

2.6 Dummy Encode the Categorical Variables

2.7 Scale the Data

2.8 Train-Test Split

Dataset Splitting:

The code splits the dataset into two parts: training set and testing set.

Features (X) are obtained by dropping the 'target' column from the original dataset (df), indicating that all other columns are considered as features.

The target variable (y) is extracted from the 'target' column of the dataset.

Train-Test Split:

The train_test_split function from sklearn.model_selection is used to split the dataset.

The test_size parameter is set to 0.2, indicating that 20% of the data will be used for testing, while 80% will be used for training.

The random_state parameter is set to 42 for reproducibility, ensuring that the same random split is obtained each time the code is run.

Output:

The code prints the shapes of the training and testing sets, indicating the number of samples and features in each set.

Inference:

The dataset is split into training and testing sets to prepare for model training and evaluation.

The training set (X_train, y_train) will be used to train machine learning models.

The testing set (X_test, y_test) will be used to evaluate the performance of the trained models on unseen data.

By splitting the dataset, you can ensure that the model's performance is assessed on data it hasn't seen during training, helping to gauge its ability to generalize to new data.

This splitting is a crucial step in the machine learning workflow, allowing you to develop models that can make accurate predictions on unseen data.

3. Logistic Regression (Full Model)

Logistic Regression Model Training:

The code imports LogisticRegression from sklearn.linear_model.

An instance of the Logistic Regression model (logistic_model) is created.

The model is trained using the training data (X_train and y_train) with the fit() method.

Model Prediction:

The trained model is used to make predictions on both the training and testing datasets (X_train and X_test), resulting in logistic_train_preds and logistic_test_preds, respectively.

Model Evaluation:

The accuracy of the model is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in logistic_train_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

DecisionTreeClassifier(1 Mark)

Decision Tree Classifier Model Training:

The code imports DecisionTreeClassifier from sklearn.tree.

An instance of the Decision Tree classifier model (decision_tree_model) is created.

The model is trained using the training data (X_train and y_train) with the fit() method.

Model Prediction:

The trained model is used to make predictions on both the training and testing datasets (X_train and X_test), resulting in decision_tree_train_preds and decision_tree_test_preds, respectively.

Model Evaluation:

The accuracy of the model is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in decision_tree_train_accuracy.

The testing accuracy is printed and stored in decision_tree_test_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

Use BaggingClassifier and evaluate the train accuracy and Test accuracy(1 Mark)

Bagging Classifier Initialization:

The code imports BaggingClassifier from sklearn.ensemble.

A base classifier is initialized as a Decision Tree classifier (DecisionTreeClassifier()).

The Bagging Classifier is initialized with the base classifier, specifying the number of base estimators (n_estimators=10) and setting the random state for reproducibility.

Model Training:

The Bagging Classifier is trained on the training data (X_train and y_train) using the fit() method.

Model Prediction:

Predictions are made on both the training and testing datasets (X_train and X_test) using the trained Bagging Classifier, resulting in bagging_train_preds and bagging_test_preds, respectively.

Model Evaluation:

The accuracy of the Bagging Classifier is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in bagging_train_accuracy.

The testing accuracy is printed and stored in bagging_test_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies of the Bagging Classifier.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

Use RandomForestClassifier and evaluate the train accuracy and Test accuracy(1 Mark)

Random Forest Classifier Initialization:

The code imports RandomForestClassifier from sklearn.ensemble.

A Random Forest Classifier is initialized with the specified number of estimators (n_estimators=100) and setting the random state for reproducibility.

Model Training:

The Random Forest Classifier is trained on the training data (X_train and y_train) using the fit() method.

Model Prediction:

Predictions are made on both the training and testing datasets (X_train and X_test) using the trained Random Forest Classifier, resulting in rf_train_preds and rf_test_preds, respectively.

Model Evaluation:

The accuracy of the Random Forest Classifier is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in rf_train_accuracy.

The testing accuracy is printed and stored in rf_test_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies of the Random Forest Classifier.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

AdaBoostClassifier and evaluate the train accuracy and Test accuracy(1 Mark)

AdaBoost Classifier Initialization:

The code imports AdaBoostClassifier from sklearn.ensemble.

A base classifier is initialized as a Decision Tree classifier with limited depth (max_depth=1).

The AdaBoost Classifier is initialized with the base classifier, specifying the number of estimators (n_estimators=100) and setting the random state for reproducibility.

Model Training:

The AdaBoost Classifier is trained on the training data (X_train and y_train) using the fit() method.

Model Prediction:

Predictions are made on both the training and testing datasets (X_train and X_test) using the trained AdaBoost Classifier, resulting in ada_train_preds and ada_test_preds, respectively.

Model Evaluation:

The accuracy of the AdaBoost Classifier is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in ada_train_accuracy.

The testing accuracy is printed and stored in ada_test_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies of the AdaBoost Classifier.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

Use GradientBoostingClassifier and evaluate the train accuracy and Test accuracy(1 Mark)

Gradient Boosting Classifier Initialization:

The code imports GradientBoostingClassifier from sklearn.ensemble.

A Gradient Boosting Classifier is initialized with the specified number of estimators (n_estimators=100) and setting the random state for reproducibility.

Model Training:

The Gradient Boosting Classifier is trained on the training data (X_train and y_train) using the fit() method.

Model Prediction:

Predictions are made on both the training and testing datasets (X_train and X_test) using the trained Gradient Boosting Classifier, resulting in gb_train_preds and gb_test_preds, respectively.

Model Evaluation:

The accuracy of the Gradient Boosting Classifier is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in gb_train_accuracy.

The testing accuracy is printed and stored in gb_test_accuracy.

Visualization:

A bar plot is created to visualize the training and testing accuracies of the Gradient Boosting Classifier.

The training accuracy is represented by a blue bar, and the testing accuracy is represented by an orange bar.

Use VotingClassifier and evaluate the train accuracy and Test accuracy(1 Mark)

Voting Classifier Initialization:

The code imports VotingClassifier from sklearn.ensemble.

Three base classifiers are initialized: Logistic Regression (LogisticRegression), Decision Tree (DecisionTreeClassifier), and Support Vector Machine (SVC).

The Voting Classifier is initialized with these base classifiers (logistic_regression_clf, decision_tree_clf, svc_clf) and the voting strategy set to 'hard'.

Model Training:

The Voting Classifier is trained on the training data (X_train and y_train) using the fit() method.

Model Prediction:

Predictions are made on both the training and testing datasets (X_train and X_test) using the trained Voting Classifier, resulting in voting_train_preds and voting_test_preds, respectively.

Model Evaluation:

The accuracy of the Voting Classifier is evaluated on both the training and testing datasets using the accuracy_score metric from sklearn.metrics.

The training accuracy is printed and stored in voting_train_accuracy.

The testing accuracy is printed and stored in voting_test_accuracy.

Visualization:

There's a mistake in the visualization part of the code, where it's printing the training and testing accuracies for the RandomForestClassifier instead of the VotingClassifier. The correct code should visualize the training and testing accuracies of the VotingClassifier.

Compare the Logistic Regression,DecisionTreeClassifier,Bagging Classifier,Random Forest,AdaBoost, Gradient Boosting, Votingand visualize Train and Test Accuracy(3 Mark)

The provided inferences offer a detailed analysis of the comparative performance of various classifiers based on train and test accuracies. Here's a summary distilled from the inferences:

1. **Logistic Regression:**
    - Similar performance between train and test sets suggests no overfitting.
    - Lower accuracy compared to ensemble methods indicates its limitations.
2. **Decision Tree:**
    - Higher accuracy on the training set suggests overfitting.
    - Tendency to overfit the training data is a common trait of decision tree classifiers.
3. **Bagging Classifier:**
    - Bagging with decision trees reduces overfitting and improves generalization.
    - Test accuracy is slightly higher than that of a single decision tree, demonstrating the effectiveness of ensemble methods.
4. **Random Forest:**
    - Random Forests introduce randomness in feature selection, enhancing generalization.
    - Test accuracy is competitive and higher compared to Bagging and Decision Tree, showcasing the advantages of randomness in ensemble methods.
5. **AdaBoost:**
    - Focuses on hard-to-classify instances, leading to improved overall performance.
    - Exhibits strong performance on both train and test sets, highlighting its effectiveness in addressing classification challenges.
6. **Gradient Boosting:**
    - Provides high accuracy, as evidenced by higher accuracy on both train and test sets.
    - Sequential training of weak learners contributes to improved accuracy, making it a powerful ensemble method.
7. **Voting Classifier:**
    - Combines multiple models' predictions, resulting in improved accuracy.
    - Leverages the strengths of diverse models to enhance predictive power and generalization.
8. **Overall Comparison:**

- Ensemble methods like Random Forest, AdaBoost, Gradient Boosting, and Voting Classifier outperform individual classifiers like Logistic Regression and Decision Tree.
- They mitigate overfitting and enhance predictive power by leveraging diverse models.
- Gradient Boosting shows the highest accuracy on both train and test sets, indicating its effectiveness in this scenario.

Comparison Graph: