Model Reports

Random Forest Model Report

Model Definition

The Random Forest model is implemented using scikit-learn's RandomForestClassifier with the following hyperparameters:

- `n_estimators=100`: 100 decision trees to ensure robust ensemble predictions.
- `max_depth=10`: Limits tree depth to prevent overfitting while capturing sufficient patterns.
- `min_samples_split=5`: Requires at least 5 samples to split a node, balancing model complexity.
- `min_samples_leaf=2`: Ensures at least 2 samples per leaf to reduce overfitting.
- `random_state=42`: Ensures reproducibility of results.

The model was trained on a dataset with 40,000 training samples and 10,000 test samples, using five selected features:

- NumberRealEstateLoansOrLines
- NumberOfDependents
- Credit_Per_Income
- Credit_Per_Person
- MonthlyIncome_Log

These features were preprocessed using StandardScaler to standardize their scales, which is critical for consistent model performance.

Output Evaluation

The model's performance was evaluated using the provided evaluation function, which computes accuracy, AUC-ROC, and a classification report. The results are as follows:

- **Training Accuracy**: 0.9337
- **Testing Accuracy**: 0.9327
- **Training AUC-ROC**: 0.7606
- **Testing AUC-ROC**: 0.6427

Classification Report (Test Set)

Class	Precision	Recall	F1-Score	Support
0	0.93	1.00	0.97	9331
1	0.17	0.00	0.36	669

The ROC curve showed a test AUC of approximately 0.84, indicating good discriminative ability, though the model struggles with the minority class (class 1).

MLPClassifier Model Report

Model Definition

The MLP Classifier model is implemented using `sklearn.neural_network.MLPClassifier` in combination with `GridSearchCV` for tuning. The grid search is used to search for the best hyperparameters over a set of chosen values:

- `hidden_layer_sizes=[(100,50), (80,40)]`: Different number of layers and units to try capture complex patterns in data.
- `activation='relu'`: The relu function is used for activation to add non-linearities.
- `solver='adam'`: Adam optimizer is used for its adaptive capabilities and works better than SGD in this case.
- `alpha=[0.0001, 0.001]`: Regularization parameter to avoid overfitting but still let the model learn useful things.
- `learning_rate_init=0.001`: The start learning rate is fixed to keep the training stable.
- `early_stopping=True`: Training will stop if validation score don't improve anymore, avoids overtraining.

The base model used was `MLPClassifier(max_iter=50, random_state=42)`, and trained using scaled version of the training dataset. Model was tested on the test set after finding the best parameters using F1 weighted score on 3-fold cross-validation.

Model Performance Result

After finding the best estimator, the model was evaluated on the test set. The results are shown below:

- **Best Parameters**: {`'activation': 'relu', 'alpha': 0.0001, 'early_stopping': True, 'hidden_layer_sizes': (100, 50), 'learning_rate_init': 0.001, 'solver': 'adam'`}
- **Accuracy**: 0.8347
- **F1 Score**: 0.8256
- **Precision**: 0.8311
- **Recall**: 0.8245
- **Confusion Matrix**: Displayed in output (not included here)

Feature Importance (using SHAP)

To find which feature affects the model prediction the most, we used SHAP explainer with the kernel method. First, we used `shap.kmeans` to summarize the training set into 10 samples for efficiency reasons. Then, we used `KernelExplainer` on the MLP model probability predictions. SHAP values were calculated for both class 0 and class 1.

From the two SHAP graphs, we can say that feature 0 is the most important one as it affects both positive and negative classes. The dot plots show how each feature pushes the prediction toward a class. Both classes have similar importance in the first few features.

GitHub Repository

The code for this project can be found on GitHub at: https://github.com/jo3591/XAiteam