**A. Lending Underwriting Models Comparison**

**1. Model Identifier**

Name: Lending Underwriting Models Comparison

Version: 1.0

Date: 4/5/2024

**2. Objective**

**Purpose of the Model**: To compare the performance of three different underwriting models (E1, E2, E3) in predicting loan default probabilities.

**Business Problem Addressed:** Enhancing the accuracy of loan default predictions to optimize lending decisions and minimize risk.

**3. Development Team**

Developer: Akshara Venkatesh

**4. Data Sources**

**Data Description:**

**Type of data:** Structured (loan applications, financial histories)

**Sources:** Internal databases

**Data Collection Methods:**

**Data Preprocessing:**

* **Cleaning steps:** Removal of outliers, handling missing values
* **Feature engineering:** Creation of credit score bands, income verification flags

Deep Feature Synthesis (DFS) is an automated feature engineering technique that enhances data analysis by creating new features from relational databases through operations like sums and averages, capturing complex patterns to improve predictions, such as loan defaults, by revealing deeper insights into financial behaviors.

* **Data augmentation techniques:**

**5. Model Architecture**

**Type of Model:**

Binomial Generalized Linear Model, Deep Feature Synthesis + Binomial Generalized Linear Model , Distributed Random Forests

**Framework Used:**

H2O.ai

**Key Parameters:**

The H2O.ai settings include the following key configurations:

* **Initialization:** The H2O server is started with `h2o.init()` which initializes the H2O machine learning environment.
* **Data Conversion:** DataFrames are converted to H2OFrame using `h2o.H2OFrame()`, making them compatible with H2O's machine learning tools.
* **Data Preparation:** The target column is converted to a factor (categorical variable) for classification purposes using `asfactor()` method. The data is then split into training and test sets using a 50-50 split ratio.
* **Model Training with AutoML:**
  + H2OAutoML Configuration: AutoML is configured with a maximum of 20 models, a seed for reproducibility, and a runtime limit of 3600 seconds.
  + Algorithm Selection: Only the Distributed Random Forest (DRF) or Generalized Linear Models (GLM) are included in different runs, as specified by the `include\_algos` parameter.
  + Training: AutoML is run on the training data with specified features and target columns.
* **Model Evaluation:**
  + The best model from AutoML (`automl.leader`) is selected.
  + The performance of this model is evaluated on the test data, and metrics such as accuracy, F1 score, and recall are calculated.
* **Advanced Analysis:** For certain models like GLM, variable importance and partial dependence plots are generated to understand feature influence on the model prediction.

**Algorithms/Techniques Used:** AutoML for model selection and parameter tuning

**6. Training**

Environment:

Hardware specifications

Software versions: H2O 3.32.1.3, Python 3.12.2

Training Data Split:

Training set: 50%

Test set:50%

Hyperparameters: Managed by H2O AutoML

Optimization Techniques: Auto-tuning by H2O AutoML

**7. Performance**

Evaluation Metrics: Accuracy

Results Summary: Comparative analysis shows Model E2 outperforms others in terms of Accuracy.

Comparison with Previous Versions: First version of comparative analysis.

Plots:

Variable importance plots

**8. Implementation**

Deployment Environment: Local environment

Integration Points:

Interfaces/Endpoints:

**9. Monitoring and Maintenance**

Monitoring Strategy:

Performance metrics to track: AUC, accuracy

Alert thresholds: AUC drop below 0.75, accuracy under 70%

Scheduled Maintenance:

Model Updates:

Frequency:

Process:

**10. Ethical Considerations**

Bias and Fairness:

Impact Assessment:

**11. Documentation and References**

Technical Documentation:

User Documentation:

Relevant Papers/Research:

**B. Collections Effectiveness Model using Graph Attention Neural Network**

**1.Model Identifier:**

Name: Collections Effectiveness Model using Graph Attention Neural Network

Version: 1.0

Date: 4/5/2024

**2. Objective:**

Purpose of the Model: To optimize debt collection strategies by identifying key accounts that may default on payments.

Business Problem Addressed: Minimizes revenue loss and enhances collection efforts by targeting specific high-risk accounts.

**3. Development Team:**

Developer: Akshara Venkatesh

**4. Data Sources:**

Data Description:

Type of data: Structured (Includes columns like Account Number, Customer Name, Loan Amount, Outstanding Balance, Payment Due Date, Last Payment Date, Last Payment Amount, Days Past Due, Collection Status, Risk Category)

Sources: Internal databases

Data Collection Methods:

Data Preprocessing:

Cleaning steps: Handling Missing Values

Feature engineering:

Data augmentation techniques:

**5.Model Architecture:**

Type of Model: Graph Attention Networks (GAT)

Framework Used: PyTorch

Key Parameters:

* Tensor Operations:

Tensors are manipulated for model inputs, which suggests the use of `torch.tensor` for creating tensor objects from data arrays.

* Model Architecture Elements:

Graph Attention Network (GAT): This type of neural network is used, which indicates the utilization of specific layers and functions associated with graph-based neural networks in PyTorch.

Loss Function: A mean squared error (MSE) loss is mentioned, implying the use of `torch.nn.MSELoss()` for calculating the loss between the predicted and actual values. There is a user warning related to mismatched target and input sizes, which emphasizes attention to dimensionality in model inputs and outputs.

* Backpropagation and Optimization:

While specific optimizer details aren't provided in the printed outputs, typical PyTorch implementations would involve defining an optimizer from `torch.optim`, such as `SGD` or `Adam`, to update model weights based on computed gradients.

* Debugging and Error Handling:

Algorithms/Techniques Used:Attention mechanisms to determine influential accounts based on feature importance.

**6. Training:**

Environment:

Hardware specifications:

Software versions: 2.3.0+cpu

Hyperparameters:

* Learning Rate : 0.01
* Number of Epochs : 100
* Loss Function : Mean Squared Error Loss (MSE Loss)

Optimization Techniques:

* Optimizer: Adam

**7. Performance:**

Evaluation Metrics: Root Mean Squared Error

Results Summary: model's predictions match the actual data points in terms of the model's

objective, which is minimizing the days past due prediction error.

Comparison with Previous Versions: First Version of Optimisation of collections.

Graphs/Charts:

**8. Implementation:**

Deployment Environment:

Integration Points:

Interfaces/Endpoints:

**9. Monitoring and Maintenance:**

Monitoring Strategy:

Performance metrics to track: Root Mean Squared Error

Scheduled Maintenance:

Model Updates:

**10. Ethical Considerations:**

Bias and Fairness:

Impact Assessment:

**11. Documentation and References:**

Technical Documentation:

User Documentation:

Relevant Papers/Research:

**C. : Hyperpersonalisation Recommendation System**

**1. Model Identifier:**

Name: Hyperpersonalisation Recommendation System

Version: 1.0

Date:4/5/2024

**2. Objective:**

Purpose of the Model: To provide personalized recommendations using Graph Attention Networks (GAT).

Business Problem Addressed:Enhancing customer experience through tailored product and service recommendations.

**3. Development Team:**

Developer: Akshara

**4. Data Source:**

Data Description:

Type of data: Structured synthetic data including customer types, industries, locations, products, transactions, risk ratings, and regulatory requirements.

Sources: Internal Databases

Data Collection Methods:

Data Preprocessing:

Cleaning steps:

Feature engineering: Creating node features from raw attributes like customer demographics, transaction history, or product usage patterns.

Encoding categorical variables into numeric formats.

Constructing derived features, such as aggregating transaction amounts over certain periods or computing the frequency of service usage.

Data augmentation techniques:

**5. Model Architecture:**

Type of Model: Graph Attention Networks (GAT)

Framework Used: PyTorch

Key Parameters:

Model Definition and Evaluation:

The model is likely using a Graph Attention Network (GAT) architecture, typically defined in PyTorch using custom classes or modules that inherit from `torch.nn.Module`.

The model is set to evaluation mode with `model.eval()` during inference to ensure layers like dropout are disabled.

Data Handling:

Node features and edge indices are processed and utilized, likely represented as `torch.Tensor`.

The model uses these tensors for making predictions: `predictions = model(graph.x, edge\_index)`.

Prediction and Probability Calculation:

Predictions are converted to probabilities using softmax: `probabilities = torch.softmax(predictions, dim=1)`.

The top recommendations are extracted using `torch.topk`, which is a function to find the `k` largest elements.

No Gradient Calculation During Inference:

Gradient calculation is disabled during the inference using `torch.no\_grad()`, which is critical for reducing memory usage and improving computation speed during model evaluation.

Utilities and Torch Operations:

PyTorch operations like slicing tensors and manipulating tensor dimensions.

There might be additional functions related to the management of graph data and its features.

Algorithms/Techniques Used: GAT leverages the attention mechanism to weight the influence of different nodes in a graph, which is particularly useful for recommendation systems.

**6. Training:**

Environment:

Hardware specifications:

Software versions: 2.3.0+cpu

Hyperparameters:

Number of layers in GAT: The model is defined with a variable `num\_layers` which determines the number of GAT convolution layers.

Hidden dimension: Each GAT layer has a `hidden\_dim` parameter specifying the size of the hidden layer.

Output dimension: The output dimension of the model is specified by `output\_dim`.

Learning rate: The learning rate for the Adam optimizer is set to `0.01`.

Epochs: The model is trained for a fixed number of epochs (`10` epochs).

Optimization Techniques:

Optimizer: Adam optimizer for training the model.

Loss function: A cross-entropy loss (`nn.CrossEntropyLoss()`) is used as the criterion for training the model.

Training loop: The model training involves a standard loop where for each epoch, gradients are set to zero, the model output is computed, the loss is calculated, backpropagation is performed, and the optimizer updates the model parameters.

**7. Performance:**

Evaluation Metrics: Accuracy, Precision, Recall, F1 Score

Results Summary: Graph Attention Network (GAT) model to generate product recommendations for

various customers

Comparison with Previous Versions: First Version of Optimisation of collections.

Graphs/Charts:

**8. Implementation:**

Deployment Environment:

Integration Points:

Interfaces/Endpoints:

**9. Monitoring and Maintenance:**

Monitoring Strategy:

Scheduled Maintenance:

Model Updates:

Frequency:

Process:

**10. Ethical Considerations:**

Bias and Fairness:

Impact Assessment:

**11. Documentation and References:**

Technical Documentation:

User Documentation:

Relevant Papers/Research: