Low-Level Design (LLD) for BigMart Sales Prediction Project

---

1. Introduction

This document provides a detailed technical design for building a machine learning model to predict sales for BigMart products across different outlets. It includes step-by-step information on the implementation, data handling, feature engineering, model training, and evaluation.

---

2. Input Specifications

2.1 Data Sources: - CSV file containing sales data for 2013 with product and store features.

2.2 Data Attributes: - **Product Features:** - `Item_Identifier`: Unique identifier for each product. - `Item_Weight`: Weight of the product. - `Item_Fat_Content`: Categorical feature indicating low or regular fat. - `Item_Visibility`: Percentage of total display area allocated to the product. - `Item_Type`: Category to which the product belongs. - `Item_MRP`: Maximum Retail Price of the product.

- **Store Features:**
  - `Outlet_Identifier`: Unique identifier for each outlet.
  - `Outlet_Establishment_Year`: The year when the outlet was established.
  - `Outlet_Size`: Size of the outlet (small, medium, large).
  - `Outlet_Location_Type`: Categorical feature indicating outlet location (urban, rural).
  - `Outlet_Type`: Type of outlet (e.g., supermarket, grocery store).
- **Target Variable:**
  - `Item_Outlet_Sales`: Sales for each product at a particular outlet.

---

3. Functional Requirements

3.1 Data Preprocessing: - **Objective:** Clean and preprocess data to make it suitable for machine learning.

Steps: - **Handling Missing Values:** - `Item_Weight`: Fill missing values using the median. - `Outlet_Size`: Fill missing values using the mode.

- **Outlier Detection:**
  - Identify outliers in numerical features like `Item_Visibility`. Cap outliers above 99th percentile.

- **Feature Transformation:**
    - Create `Outlet_Age` by subtracting `Outlet_Establishment_Year` from the current year (2013).
    - Binning `Item_Visibility` into low, medium, and high categories to reduce skewness.

3.2 Exploratory Data Analysis (EDA): - **Objective:** Understand the distribution of data and relationships between variables.

Steps: - **Univariate Analysis:** - Plot histograms for `Item_Outlet_Sales`, `Item_MRP`, and `Item_Weight`. - Analyze categorical features using bar plots (`Item_Type`, `Outlet_Type`).

- **Bivariate Analysis:**
    - Scatter plots to visualize `Item_MRP` vs. `Item_Outlet_Sales`.
    - Box plots to check sales distribution across `Outlet_Type` and `Outlet_Location_Type`.
    - Correlation heatmaps to find relationships between numerical variables.

Tools:
Use `matplotlib` and `seaborn` for visualizations.

3.3 Feature Engineering: - **Objective:** Create meaningful features that improve the model's performance.

Steps: - **Create Derived Features:** - `Outlet_Age`: Current year (2013) minus `Outlet_Establishment_Year`. - `Item_Category`: Group `Item_Type` into broader categories (e.g., Food, Non-Food).

- **Interaction Features:**
    - Create interaction terms between `Item_Type` and `Outlet_Type` to capture product-outlet relationships.

Tools:
`pandas` and `numpy`.

3.4 Categorical Encoding: - **Objective:** Convert categorical variables into numeric form for modeling.

Steps: - **Label Encoding:** Use `LabelEncoder` for ordinal features such as `Outlet_Size`. - **One-Hot Encoding:** Apply `OneHotEncoder` or `pd.get_dummies()` on nominal variables like `Item_Fat_Content`, `Outlet_Type`, `Item_Type`, etc.

Tools:
`sklearn.preprocessing.LabelEncoder` and `pandas.get_dummies()`.

3.5 Model Training: - **Objective:** Train various machine learning models to predict sales.

Steps: - **Split Data:** - Split data into training (70%) and testing (30%) sets using `train_test_split`.

- **Baseline Model:**
  - **Linear Regression:**
    - Train a basic linear regression model using `sklearn.linear_model.LinearRegression`.
    - Evaluate using `mean_absolute_error` and `r2_score`.
- **Regularized Models:**
  - **Ridge and Lasso Regression:** Train Ridge and Lasso regression models with hyperparameter tuning.
- **Non-linear Models:**
  - **Random Forest:** Train a `RandomForestRegressor` to capture non-linear relationships and feature interactions.
  - **XGBoost:** Train an `XGBRegressor` for better performance with optimized hyperparameters.

Tools:
`sklearn.linear_model`, `sklearn.ensemble.RandomForestRegressor`, `xgboost.XGBRegressor`.

3.6 Model Evaluation: - **Objective:** Evaluate the model's performance on test data.

Steps: - Calculate metrics: - **Mean Absolute Error (MAE):** Use `mean_absolute_error` to measure prediction accuracy. - **R-squared (R^2):** Use `r2_score` to explain the variance captured by the model.

- Perform cross-validation to assess model stability.

Tools:
`sklearn.metrics.mean_absolute_error`, `sklearn.metrics.r2_score`, `sklearn.model_selection.cross_val_score`.

3.7 Model Tuning: - **Objective:** Optimize model hyperparameters to improve performance.

Steps: - **GridSearchCV:** Use grid search to tune hyperparameters such as `n_estimators` and `max_depth` for RandomForest and XGBoost. - **Feature Selection:** Apply feature importance analysis (e.g., based on Random Forest or XGBoost results) to eliminate less relevant features.

Tools:
`sklearn.model_selection.GridSearchCV`.

3.8 Final Model Selection: - **Objective:** Select the best-performing model based on evaluation metrics.

Steps: - Compare all models (Linear Regression, Ridge, Lasso, RandomForest, XGBoost) based on MAE and R^2. - Select the model with the lowest error and highest R^2 score for final deployment.

---

4. Non-Functional Requirements

4.1 Performance: - The system should be capable of predicting sales within a reasonable time (less than 2 seconds for each prediction). - The model training should complete within a few minutes for the given dataset.

4.2 Scalability: - The solution should handle increased data volume (e.g., more stores and products) without significant degradation in performance.

4.3 Usability: - The final model should be easy to deploy and use in future sales forecasting tasks.

---

5. Deployment Details

5.1 Model Saving: - Save the final model using `joblib` or `pickle` for future use.

Example: `python import joblib joblib.dump(best_model, 'final_sales_model.pkl')`

5.2 Model Inference: - Load the saved model and use it to predict sales for new data.

Example: `python model = joblib.load('final_sales_model.pkl') predictions = model.predict(new_data)`

---

6. Risks and Mitigation

Risk	**Mitigation**
Missing or inconsistent data	Use imputation and data validation techniques
Model overfitting	Use regularization (Ridge, Lasso) and cross-validation

Risk	Mitigation
Long training times for large datasets	Use optimized algorithms like XGBoost and RandomForest
Poor model performance	Perform extensive feature engineering and hyperparameter tuning

---

7. Testing and Validation

7.1 Unit Testing: - Test individual components like data preprocessing, feature engineering, and model training using unit tests.

7.2 Integration Testing: - Ensure that the data preprocessing pipeline and model training work cohesively by testing the full workflow.

7.3 Performance Testing: - Evaluate the model's performance using validation data and ensure that the prediction time is within acceptable limits.

---

8. Conclusion

The BigMart Sales Prediction project involves building a robust machine learning model to predict product sales across stores. This Low-Level Design document outlines each step involved in data preprocessing, feature engineering, model training, evaluation, and deployment. By following this plan, the project will deliver a scalable, efficient solution for forecasting sales.