Comparative Analysis of Machine Learning Models for Demand Forecasting: A Case Study on Weekly Sales Prediction

Belgin Erdoğan, Hızır İlyas Aydoğan

Istanbul Technical University
Master of Science in Big Data & Business Analytics

August 6, 2024

Abstract

In this study, we explore a comprehensive range of machine learning models for demand forecasting using historical sales data. Our primary objective is to predict weekly sales for the upcoming period by leveraging various features extracted from the data. We utilize methods including K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Artificial Neural Networks (ANN), Classification and Regression Trees (CART), Bagging, Random Forest, Gradient Boosting Machines (GBM), Extreme Gradient Boosting (XGBoost), LightGBM, and CatBoost. The performance of these models is evaluated based on their R^2 scores, with the aim of identifying the most effective model for accurate sales prediction. The study provides a comparative analysis of these models, highlighting their strengths and limitations.

Keywords: Demand Forecasting, Machine Learning, E-commerce, Sales Prediction, Time Series Analysis, Random Forest, K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Artificial Neural Networks (ANN), Gradient Boosting Machines (GBM), XGBoost, LightGBM, CatBoost, Ensemble Methods, Feature Engineering, Rug Market, Product Attributes, Inventory Management, Sales Optimization, Predictive Modeling

1. Introduction

In this study, we focus on demand forecasting within the e-commerce sector, specifically for a company that sells rugs in the U.S. market. Demand forecasting has been a major concern of operational strategy to manage the inventory and optimize the customer satisfaction level. The researchers have proposed many conventional and advanced forecasting techniques, but no one leads to complete accuracy. [5]. Accurate sales predictions are crucial for optimizing inventory management, minimizing stock-outs and overstock situations, and improving overall operational efficiency. Our analysis concentrates on the top X products, which are responsible for 80% of the company's revenue. Key features that potentially impact sales performance include the rug's washability, pile height, and country of production—attributes that are commonly searched by U.S. consumers. Additionally, the material composition plays a significant role, as customers often scrutinize product details for information about toxic elements and safety, particularly when considering rugs for households

with children or pets. The production type, whether machine-made or handmade, is another important factor influencing consumer preferences and, consequently, sales performance. By incorporating these features, our study aims to leverage various machine learning models to forecast weekly sales. Machine Learning is one of the most talked about topics in the last few years. More and more businesses want to adopt it to maintain the competitive edge; however, very few really have the right resources and the appropriate data to implement it [6]. We evaluate a diverse set of models, including K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Artificial Neural Networks (ANN), Classification and Regression Trees (CART), Bagging, Random Forest, Gradient Boosting Machines (GBM), Extreme Gradient Boosting (XGBoost), LightGBM, and CatBoost. The performance of these models is compared using R^2 scores to determine the most effective approach for accurate demand forecasting. This research not only seeks to enhance the accuracy of sales forecasts but also provides insights into the factors driving consumer choices in the rug market. In the following sections, we detail the methodology, including data preparation, feature engineering, and model training. We then present the results, comparing the performance of each model and discussing the implications of our findings.

2. Literature Research

Machine learning (ML) techniques have gained widespread adoption across diverse domains, offering solutions to challenges posed by large and intricate datasets. These techniques use complex algorithms to find relevant patterns in large and varied datasets—a task that would be nearly impossible for a human analyst to accomplish. The main goal of machine learning is inductive inference, which is the process of creating general models from particular empirical data. The constant creation of new algorithms and the increase in data availability at lower computational costs have driven advancements in this field [3].

This literature review provides a concise overview of the machine learning models used in this study, focusing on their historical development, algorithmic details, practical applications, and key strengths and limitations.

2.1 Historical Context and Development

This review encompasses the evolution of machine learning algorithms applied in demand forecasting, spanning foundational methods like K-Nearest Neighbors (KNN) and Support Vector Regression (SVR) to sophisticated ensemble techniques such as Random Forest, Gradient Boosting Machines (GBM), and LightGBM. The development of these models highlights their increasing complexity and capability to handle diverse data sets in various applications.

2.2 Algorithm Details

Each model leverages unique principles to predict future sales:

- **K-Nearest Neighbors (KNN):** Utilizes distance metrics to predict values based on the closest data points.
- **Support Vector Regression (SVR):** Focuses on margin maximization to enhance prediction accuracy.
- **Classification and Regression Trees (CART):** Constructs decision trees to make predictions based on data splits.
- **Bagging (Bootstrap Aggregation):** Combines multiple iterations of CART to reduce variance.
- **Random Forest:** An extension of Bagging, generating a multitude of decision trees and averaging their predictions. A decision tree falls under the supervised category of machine learning and uses frequency tables for making the predictions. One advantage of a decision tree is that it can handle both categorical and numerical variables. As the name suggests, it operates in sort of a tree structure and forms these rules based on various splits to finally make predictions.[6]
- **Artificial Neural Network (ANN):** Employs layers of interconnected nodes to model complex relationships in the data.
- **Gradient Boosting Machines (GBM):** Builds models sequentially to correct errors made by preceding models.

- **Extreme Gradient Boosting (XGBoost):** An optimized version of GBM, enhancing speed and performance. XGBoost Kang et al. [4] used the XGBoost hybrid model for tourism and trend prediction. They introduced location attributes and the time-lag effect of network search data to propose the hybrid model. The findings suggest that the spatiotempo- ral XGBoost composite model outperforms single forecasting approaches. There are several modifiable parameters in the XGBoost algorithm, including general, promo- tion, and learning objective parameters. They adopted the tree model to concentrate on the nonlinear interactions between the Baidu index and the number of tourists. Using the general parameters, the promotion parameters were changed according to the model chosen.
- **LightGBM:** A highly efficient gradient boosting framework that leverages tree-based learning algorithms for faster computation and higher efficiency.

2.3 Applications

These models are extensively applied in various industries, particularly in e-commerce and retail, to forecast demand. Their versatility and capability to handle large datasets make them invaluable for predicting sales trends and optimizing inventory management.

2.4 Strengths and Limitations

Each model has distinct strengths and weaknesses:

- **KNN:** Simple and intuitive but can struggle with large datasets.
- **SVR:** Effective for small datasets but computationally intensive.
- **CART:** Easy to interpret but prone to overfitting.
- **Bagging:** Reduces overfitting but increases computational cost.
- **Random Forest:** Robust and reduces variance but can be less interpretable.
- **ANN:** Capable of capturing complex patterns but requires extensive data and tuning.
- **GBM:** Highly accurate but sensitive to overfitting and requires careful tuning.
- **XGBoost:** Fast and efficient but complex to implement.
- **LightGBM:** Highly efficient but can be sensitive to overfitting with small datasets.

This literature review sets the foundation for the empirical analysis, providing context for the performance of these models in predicting e-commerce sales.

3. Model Process Steps

In the realm of demand forecasting for e-commerce, a structured and systematic approach to modeling is essential to derive accurate and actionable insights. This process involves several key stages, each contributing to the overall effectiveness of the predictive models. These stages include business and data understanding, data preparation, model building, evaluation, and the application of Principal Component Analysis (PCA).

4. Business and Data Understanding

The business model we have analyzed pertains to an e-commerce company specializing in the global sale of rugs. Key factors and metrics to consider for accurate demand forecasting include the rug's material, country of production, and washability features.

The sales data utilized in our analysis covers the entirety of the year 2023, with a breakdown by week. The following metrics were employed in our modeling process:

- **Product Title:** The name of the product as displayed on the website, for which demand forecasting is conducted.
- **Week:** The week number of the sales data, formatted as yyyy-mm.

- **Net Quantity:** The net sales quantity of the product within the respective week.
- **Sessions:** The number of visits or sessions created during that week.
- **General Product Type:** A broad category that provides insight into the type of product.
- **Detailed Product Type:** More specific information regarding the product type.
- **Country:** The country where the product is produced.
- **Material:** The material used in the production of the rug.
- **Washability:** The washability feature of the product.
- **Top Title Comments:** The number of comments associated with each product.
- **Is Special Date:** Indicates whether the week includes special dates or holidays.
- **Pile Height:** The height of the rug's pile.
- **Color Variants:** The number of color swatches available for the product.

4.0.1 Advanced EDA

- **Categorical Variables**

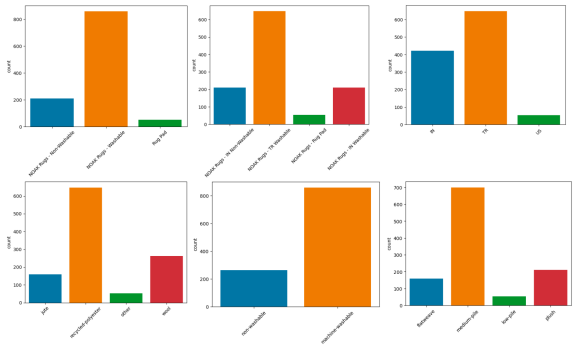

Figure 1: From left to right and top to bottom: General Product Type, Detailed Product Type, Country, Material, Washability, and Pile Height

- **Numerical Variables**

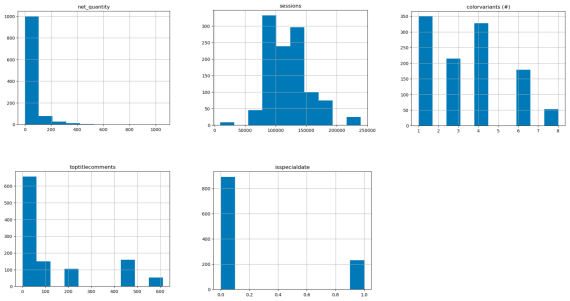

Figure 2: From left to right and top to bottom: Net Quantity, Sessions, Color Variants, Number of Comments, and Special Date

5. Data Preparation

5.1 Duplicates

```
Data Rows =  1122
Duplicate Rows =  0
Duplicate Rows % =  0.0
After drop_duplicates Data Rows = 1122
```

Figure 3: The output of duplicates

5.2 Outlier Analysis

Outlier detection shows an extreme part in many areas, such as, pattern classification grouping, and decision-making, due to the fact that it can disclose rare but essential circumstance, and find interesting /unpredicted data items [1].

In our model, we employed three distinct versions of outlier analysis: a non-outlier analysis version, a Z-Score version, and an Interquartile Range (IQR) version. After thorough evaluation, we proceeded with the Z-Score version, as it proved to be more suitable for our data. The details of these methods will be explained in the subsequent sections.

Z-Score analysis identifies outliers by measuring how many standard deviations a data point is from the mean. Typically, data points with a Z-Score

greater than 3 or less than -3 are considered outliers. The Z-Score for a data point x is calculated as:

$$Z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation.

On the other hand, IQR analysis identifies outliers by calculating the range between the first quartile (Q1) and the third quartile (Q3) and defining outliers as data points that fall below $Q1 - 1.5 \times \text{IQR}$ or above $Q3 + 1.5 \times \text{IQR}$. The IQR is calculated as:

$$\text{IQR} = Q3 - Q1$$

The main difference between the two methods is that Z-Score analysis is based on the mean and standard deviation, making it sensitive to the underlying distribution of the data, whereas IQR analysis is based on the median and quartiles, making it more robust to non-normal distributions and skewed data.

5.3 Missing Values

	Number_of_Nulls	Percentage_of_Nulls
product_title	0	%0.0
week	0	%0.0
net_quantity	0	%0.0
sessions	0	%0.0
generalproducttype	0	%0.0
detailedproducttype	0	%0.0
country	0	%0.0
material	0	%0.0
washability	0	%0.0
toptitlecomments	0	%0.0
isspecialdate	0	%0.0
pile_height	0	%0.0
colorvariants (#)	0	%0.0

Figure 4: Output of missing value results

5.4 Correlation

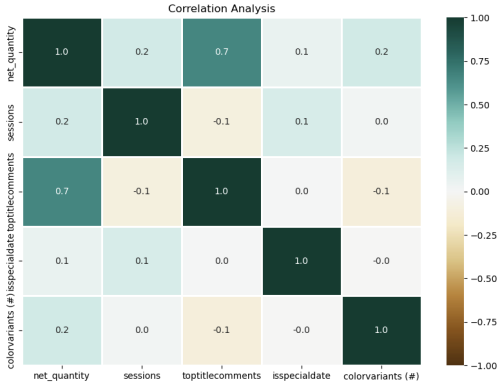

Figure 5: Correlation between features and metrics

5.5 Feature Engineering

To enhance the predictive power of our models, we performed several feature engineering steps on the dataset:

1- **Splitting the Week Column:** We split the 'week' column into 'year' and 'week' components to facilitate easier analysis and manipulation.

2- **Converting Week Number to Month:** We developed a function to convert the week number and year into a month number, ensuring a more granular temporal analysis. The function calculates the first day of the given year and adds the appropriate number of weeks to determine the month.

3- **Creating a Composite Time Column:** We created a composite column 'year_month_week' to provide a comprehensive time identifier for each record. This column is then positioned at the beginning of the DataFrame for better organization.

4- **Categorizing Session Counts:** To classify session counts into categorical variables, we used percentiles. Sessions were divided into 'Low', 'Medium', and 'High' categories based on their values relative to the 25th, 50th, and 75th percentile.

5- **Classifying Color Variants:** We categorized 'colorVariants (#)' using mean and standard deviation, creating bins that classify color variants

into groups: '1', '2', '3', and '3+'. This method allows for a simplified yet informative representation of color variants.

By implementing these feature engineering steps, we refined the dataset to better capture the underlying patterns and relationships, ultimately improving the performance of our demand forecasting models.

6. Model Preparation

6.1 Lag/Shifted Features:

In time series analysis, incorporating lagged features is crucial as it allows the model to account for temporal dependencies and patterns in the data. By using historical sales data, we can provide the model with valuable context to improve its predictive performance. We implemented lag features for `net_quantity` at intervals of 1, 2, 3, 4, 13, 26, and 52 weeks, representing previous week, month, three months, six months, and one year, respectively. This helps the model recognize seasonal trends and patterns in sales. Missing values generated from shifting were filled with zeroes to maintain data integrity.

6.2 Encoding:

In our data preprocessing, we utilized both label encoding and one-hot encoding to transform categorical variables into a format suitable for machine learning algorithms.

- **Label Encoding:** We applied label encoding to the `session_category` and `colorvariants_category` variables because these categories have an inherent order or priority. Label encoding assigns a unique numerical value to each category, preserving the ordinal relationship.

- **One-Hot Encoding:** For categorical variables without a meaningful order and with a relatively small number of unique values, we used one-hot encoding. This technique creates binary columns for each category, ensuring the model does not assume any ordinal relationship among the categories. We applied one-hot encoding to variables such as `product_title`, `generalproducttype`, `detailedproducttype`, `country`, `material`, `washability`, and `pile_height`.

- **Boolean Columns:** We converted any boolean columns to integers to maintain consistency in the dataset.

- **Dropping Non-Numeric Columns:** Finally, we removed the `year`, `month`, `week`, and `year_month_week` columns, as they were initially in object format and were no longer necessary after encoding.

6.3 Data Splitting for Model Training:

To prepare our dataset for training and evaluating our models, we split the data into training and testing sets. This ensures that we can assess the model's performance on unseen data, providing a robust evaluation metric.

- **Defining Features and Target Variable:** We first separated the target variable, `net_quantity`, from the feature set.

- **Splitting the Data:** We used the `train_test_split` function from the `sklearn.model_selection` module to divide the data. We allocated 20% of the data for testing and the remaining 80% for training. A random state of 42 was set to ensure reproducibility of the results.

- **Verifying the Split:** We verified the shapes of the training and testing sets to confirm the split proportions. The resulting shapes were 897 samples for training and 225 samples for testing.

6.4 Model Evaluation Metrics:

In our analysis, we employed two key evaluation metrics to assess the performance of our models: Root Mean Squared Error (RMSE) and R-squared (R^2). These metrics provide comprehensive insights into the accuracy and explanatory power of our predictive models.

- **Root Mean Squared Error (RMSE):** RMSE is a standard way to measure the error of a model in predicting quantitative data. It represents the square root of the average of the squared differences between the predicted and actual values.

- **R-squared (R^2):** R^2, also known as the coefficient of determination, indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It provides a measure of how well the observed outcomes are replicated by the model.

By using RMSE and R^2 as our evaluation metrics, we were able to comprehensively assess the accuracy and effectiveness of our models in predicting future sales quantities. RMSE provides insight into the average prediction error, while R^2 illustrates the proportion of variability in the sales data that our models can explain.

7. MODELS:

7.1 Classification and Regression Trees (CART)

Classification and Regression Trees (CART) are a type of decision tree algorithm used for predictive modeling. The primary advantage of CART is its ability to handle both classification and regression tasks. CART builds a binary tree structure where each internal node represents a decision based on a feature, and each leaf node represents a predicted outcome. The goal is to split the data into subsets that are as homogeneous as possible with respect to the target variable.

- **Model Performance:**
  - RMSE: 22.88
  - R-squared: 0.87

- **Tuning the CART Model:** To optimize the performance of our CART model, we performed hyperparameter tuning using GridSearchCV. This process involves searching through a specified grid of hyperparameters and selecting the combination that provides the best model performance.

1. **Parameter Grid:**

   - **max_depth:** The maximum depth of the tree. Limiting the depth can help prevent overfitting. We tested depths of `None` (unlimited), 10, 20, 30, and 40.

   - **min_samples_split:** The minimum number of samples required to split an internal node. Higher values prevent the model from learning overly specific patterns. We tested values of 2, 5, 10, and 20.

   - **min_samples_leaf:** The minimum number of samples required to be at a leaf node. Larger values can smooth the model and help prevent overfitting. We tested values of 1, 2, and 4.

   - **max_features:** The number of features to consider when looking for the best split. We tested `'auto'` (all features), `'sqrt'` (square root of the number of features), `'log2'` (logarithm of the number of features), and `None` (all features).

2. **Grid Search:** We utilized `GridSearchCV` to perform an exhaustive search over the specified parameter grid. This method evaluates each combination of parameters using cross-validation and selects the model with the best R-squared score.

3. **Best Model Selection:** After the grid search, we extracted the best model and used it to make predictions on the test set.

4. **Performance Metrics:** We evaluated the tuned model using RMSE and R-squared to compare its performance to the initial model.

- **Tuned CART Model Performance:**
  - RMSE: 22.33
  - R-squared: 0.88

By fine-tuning the hyperparameters, we achieved a slight improvement in both the RMSE and R-squared metrics, demonstrating the effectiveness of hyperparameter optimization in enhancing the model's predictive accuracy.

7.2 Bagging (Bootstrap Aggregation)

Bagging, or Bootstrap Aggregation, is an ensemble learning technique that improves the stability and accuracy of machine learning algorithms. By training multiple models on different subsets of the data and aggregating their predictions, Bagging reduces variance and enhances model performance. Each model in the ensemble is trained on a random sample of the data (with replacement), and their predictions are combined, typically by averaging, to produce a final result.

- **Model Performance:**
  - RMSE: 18.23
  - R-squared: 0.92

- **Tuning the Bagging Model:** To optimize the Bagging model, we employed GridSearchCV to find the best combination of hyperparameters. The tuning process involves adjusting the parameters of the base estimator and the Bagging ensemble itself:

1. **Parameter Grid:**
   - `estimator__max_depth`: Maximum depth of the base decision trees. We tested values of 10, 20, 30, and 40.
   - `estimator__min_samples_split`: Minimum number of samples required to split an internal node in the base decision trees. We tested values of 2, 5, 10, and 15.
   - `estimator__min_samples_leaf`: Minimum number of samples required to be at a leaf node in the base decision trees. We tested values of 1, 2, 4, and 8.
   - `n_estimators`: Number of base estimators in the Bagging ensemble. We tested values of 50, 100, and 150.
   - `max_samples`: Proportion of samples to use for fitting each base estimator. We tested values of 0.5, 0.7, and 1.0.
2. **Grid Search:** We used `GridSearchCV` to systematically explore the parameter grid and select the best model based on R-squared performance. This method was configured with 5-fold cross-validation and utilized all available CPU cores for computation.
3. **Best Model Selection:** The best model from the grid search was used to make predictions on the test set.
4. **Performance Metrics:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned Bagging Model Performance:**
  - RMSE: 17.64
  - R-squared: 0.92

The tuning process resulted in a slight improvement in RMSE, indicating enhanced performance through hyperparameter optimization.

7.3 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the average prediction (for regression) or the majority vote (for classification). This method leverages the power of multiple trees to improve predictive accuracy and control overfitting. By aggregating the results from multiple trees, Random Forest can handle large datasets and complex relationships with high accuracy.

- **Model Performance:**
  - RMSE: 18.00
  - R-squared: 0.92

- **Tuning the Random Forest Model:** To enhance the performance of the Random Forest model, we used RandomizedSearchCV for hyperparameter tuning. This approach randomly samples a set of hyperparameter combinations from a predefined distribution and evaluates them to find the best model.

1. **Parameter Distribution:**
   - `n_estimators`: Number of trees in the forest. We tested values of 100, 200, 300, and 400.
   - `max_depth`: Maximum depth of each tree. We tested depths of 3, 5, 8, 10, and 15.

- `min_samples_split`: Minimum number of samples required to split an internal node. We tested values of 2, 5, and 10.
- `min_samples_leaf`: Minimum number of samples required to be at a leaf node. We tested values of 1, 2, and 4.
- `bootstrap`: Whether to use bootstrap samples for building trees. We tested both True and False.

2. **Randomized Search:** We employed `RandomizedSearchCV` to search through the parameter distribution and identify the best hyperparameter combination. This method was configured to perform 10 iterations, using 5-fold cross-validation and leveraging all available CPU cores for computation.
3. **Best Model Selection:** We selected the best model from the random search and used it to predict on the test set.
4. **Performance Metrics:** We evaluated the tuned model using RMSE and R-squared metrics to assess its performance compared to the initial model.

- **Tuned Random Forest Model Performance:**
  - RMSE: 17.69
  - R-squared: 0.92

The tuning process led to a slight improvement in RMSE, demonstrating the effectiveness of hyperparameter optimization in refining the Random Forest model's predictive performance.

7.4 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm used for regression and classification. For regression tasks, KNN predicts the target value by averaging the values of the k-nearest neighbors in the feature space. The choice of k and distance metrics significantly impacts the model's performance.

- **Model Performance:**
  - RMSE: 18.05
  - R-squared: 0.92
- **Tuning the KNN Model:** To optimize the KNN model, we used `GridSearchCV` to find the best combination of hyperparameters. The tuning process involved exploring various values for:

1. **Parameter Grid:**
   - `n_neighbors`: The number of neighbors to consider for making predictions. We tested values from 1 to 35.
   - `weights`: The weighting function used for prediction. Options included `'uniform'` (equal weight for all neighbors) and `'distance'` (weight inversely proportional to distance).
   - `algorithm`: The algorithm used to compute the nearest neighbors. We used `'auto'`, which selects the best algorithm based on the dataset.
   - `p`: The power parameter for the Minkowski distance metric. Values of 1 (Manhattan distance) and 2 (Euclidean distance) were tested.
2. **Grid Search:** `GridSearchCV` was used to systematically explore the parameter grid and select the best model based on R-squared performance.
3. **Best Model Selection:** The best parameters from the grid search were used to retrain the KNN model.
4. **Performance Evaluation:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned KNN Model Performance:**
  - RMSE: 17.95
  - R-squared: 0.92

The tuning process led to a slight improvement in RMSE, demonstrating enhanced predictive performance through careful adjustment of hyperparameters.

7.5 Support Vector Regression (SVR)

Support Vector Regression (SVR) is a type of machine learning model used for regression tasks. It works by finding a hyperplane in a high-dimensional space that best fits the data. SVR aims to find a balance between model complexity and prediction accuracy by minimizing prediction errors within a specified margin.

- **Model Performance:**
  - RMSE: 48.23
  - R-squared: 0.42

- **Tuning the SVR Model:** To optimize the SVR model, we used a pipeline approach that included feature scaling and SVR for regression. The tuning involved adjusting several key hyperparameters:

1. **Parameter Grid:**
   - `Regularization Parameter (C):` Controls the trade-off between fitting the training data well and keeping the model simple. Tested values included 0.1, 1, 10, and 100.
   - `Kernel Coefficient (Gamma):` Defines the influence of individual training examples. Values tested were `'scale'` and `'auto'`.
   - `Kernel Type:` The function used to transform the input data. Options included `'linear'`, `'rbf'` (Radial Basis Function), and `'poly'` (polynomial).
   - `Degree of Polynomial Kernel:` Only used if the polynomial kernel is selected. Values tested ranged from 2 to 4.
2. **Grid Search:** We performed a grid search to find the best combination of hyperparameters for the SVR model.
3. **Best Model Selection:** After the grid search, we evaluated the performance of the tuned model using the specified metrics.
4. **Performance Evaluation:** The tuned model's performance was assessed using RMSE and R-squared metrics.

- **Tuned SVR Model Performance:**
  - RMSE: 18.64
  - R-squared: 0.90

The tuning improved the SVR model, resulting in a lower RMSE and higher R-squared score compared to the initial model.

7.6 Artificial Neural Network (ANN)

Artificial Neural Networks (ANNs) are computational models inspired by the human brain. They consist of interconnected nodes (neurons) organized in layers: input, hidden, and output. ANNs are capable of capturing complex patterns and relationships in data, making them powerful for regression tasks.

- **Model Performance:**
  - RMSE: 15.26
  - R-squared: 0.94

- **Tuning the ANN Model:** To optimize the ANN model, we performed a grid search to find the best hyperparameters:

1. **Parameter Grid:**
   - `Alpha:` Regularization parameter to prevent overfitting. Tested values included 0.1, 0.01, 0.02, and 0.005.
   - `Hidden Layer Sizes:` The number of neurons and layers in the hidden layers. Tested configurations were (20,20), (100,50,150), and (300,200,150).
   - `Activation Function:` Function used to introduce non-linearity. Tested options were `'relu'` (Rectified Linear Unit) and `'logistic'` (Sigmoid).
2. **Grid Search:** We performed a grid search to identify the best hyperparameters for the ANN model.
3. **Best Model Selection:** The best parameters from the grid search were used to retrain the ANN model.
4. **Performance Evaluation:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned ANN Model Performance:**
  - RMSE: 16.53
  - R-squared: 0.93

The tuned ANN model showed improved performance with a lower RMSE and higher R-squared score compared to the initial model.

7.7 Gradient Boosting Machines (GBM)

Gradient Boosting Machines (GBM) is an ensemble technique that builds models sequentially, where each model corrects errors made by the previous ones. By combining multiple weak learners (typically decision trees), GBM aims to reduce errors and improve predictive performance.

- **Model Performance:**
  - RMSE: 16.1
  - R-squared: 0.94

- **Tuning the GBM Model:** For optimizing the GBM model, we performed a grid search to identify the best hyperparameters:

1. **Parameter Grid:**
   - `n_estimators`: The number of boosting stages to be run. Tested values were 100, 200, and 500.
   - `learning_rate`: Step size at each iteration to prevent overfitting. Tested values included 0.1, 0.01, and 0.001.
   - `max_depth`: Maximum depth of the individual trees. Tested depths were 3, 5, and 8.
   - `min_samples_split`: Minimum number of samples required to split an internal node. Tested values were 2, 5, and 10.
   - `min_samples_leaf`: Minimum number of samples required to be at a leaf node. Tested values included 1, 2, and 4.
2. **Grid Search:** We performed a grid search to identify the best hyperparameters for the GBM model.
3. **Best Model Selection:** The model was retrained with the optimal parameters obtained from the grid search.
4. **Performance Evaluation:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned GBM Model Performance:**
  - RMSE: 16.56
  - R-squared: 0.93

The tuned GBM model provided a robust performance, showing a strong R-squared value and a low RMSE.

7.8 Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) is a powerful and efficient machine learning algorithm that builds upon the principles of gradient boosting. It enhances the standard boosting process with additional techniques to improve speed and performance, making it particularly effective for large datasets and complex problems.

- **Model Performance:**
  - RMSE: 18.66
  - R-squared: 0.91

- **Tuning the XGBoost Model:** The tuning process involved finding the optimal hyperparameters to enhance the model's performance:

1. **Parameter Grid:**
   - `learning_rate`: Controls the step size during training. Values tested were 0.1, 0.01, and 0.03.
   - `n_estimators`: Number of boosting rounds. Tested values included 100, 200, and 300.
   - `colsample_bytree`: Fraction of features to be used for each tree. Values were 0.5, 0.7, and 1.
   - `subsample`: Fraction of samples used for fitting individual trees. Values tested were 0.5, 0.7, and 1.

- `max_depth`: Maximum depth of the trees. Values included 3, 5, and 8.
- `min_child_weight`: Minimum sum of weights needed for a child node. Tested values were 1, 3, and 5.
- `gamma`: Minimum loss reduction required to make a further partition. Values were 0, 0.1, and 0.5.

2. **Grid Search:** After tuning, the model was retrained with the optimal parameters.
3. **Performance Evaluation:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned XGBoost Model Performance:**
  - RMSE: 16.59
  - R-squared: 0.93

The tuned XGBoost model demonstrated improved accuracy with a high R-squared value and a low RMSE.

7.9 LightGBM

LightGBM (Light Gradient Boosting Machine) is an efficient and scalable gradient boosting framework that uses a histogram-based learning method to improve training speed and reduce memory usage. It's particularly well-suited for large datasets and high-dimensional features.

- **Model Performance:**
  - RMSE: 16.75
  - R-squared: 0.93

- **Tuning the LightGBM Model:** The tuning process involved optimizing several hyperparameters to achieve better model performance:

1. **Parameter Grid:**
   - `learning_rate`: Determines the step size during training. Tested values were 0.001, 0.01, and 0.1.
   - `n_estimators`: Number of boosting iterations. Values included 100, 200, and 500.
   - `colsample_bytree`: Fraction of features used for each tree. Tested values were 0.5, 0.7, and 1.
   - `num_leaves`: Maximum number of leaves in one tree. Values tested were 20, 30, and 50.
   - `max_depth`: Maximum depth of the tree. Tested values included 3, 5, and 8.
2. **Grid Search:** After tuning, the model was retrained with the optimal parameters.
3. **Performance Evaluation:** The tuned model's performance was evaluated using RMSE and R-squared metrics.

- **Tuned LightGBM Model Performance:**
  - RMSE: 15.92
  - R-squared: 0.94

The tuned LightGBM model showed the best performance with the lowest RMSE and highest R-squared value, reflecting its effectiveness in capturing the underlying patterns in the data.

8. Evaluation

The chart presented below displays the evaluation metrics for all the models considered.

	Model	RMSE	R-squared	Tuned RMSE	Tuned R-squared
0	Classification and Regression Trees (CART)	22.88	0.87	22.33	0.88
1	Bagging (Bootstrap Aggregation)	18.23	0.92	17.64	0.92
2	Random Forest	18.00	0.92	17.69	0.92
3	K-Nearest Neighbors (KNN)	18.05	0.92	17.95	0.92
4	Support Vector Regression (SVR)	48.23	0.42	18.64	0.90
5	Artifical Neural Network (ANN)	15.26	0.94	16.53	0.93
6	Gradient Boosting Machines (GBM)	16.10	0.94	16.56	0.93
7	Extreme Gradient Boosting (XGBoost)	18.66	0.91	16.59	0.93
8	LightGBM	16.75	0.93	15.92	0.94

Figure 6: Evaluation of all models

9. Feature Importance

Feature importance provides insight into which features (or variables) have the most significant impact on the predictions made by a model. Understanding feature importance is crucial for several reasons:

1. **Model Interpretability**: It helps in understanding how different features influence the model's predictions, making the model more transparent and easier to interpret.

2. **Feature Selection**: By identifying the most influential features, you can simplify the model by removing less important features, which can improve model performance and reduce computational cost.

3. **Domain Knowledge**: It provides valuable information that can be used to gain deeper insights into the underlying data, potentially revealing new trends or relationships.

4. **Improving Model Performance**: By focusing on the most important features, you can fine-tune the model to better capture the relevant patterns in the data.

The chart below displays the feature importance for the models used.

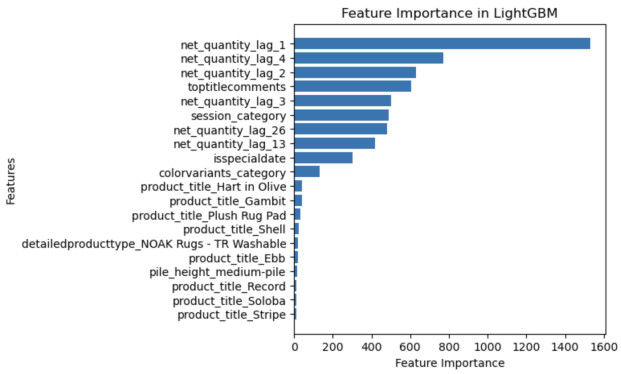

Figure 7: Future importance

It is evident that, aside from lag-related features, the number of comments holds significant importance for the models.

10. Principal Component Analysis (PCA) and Model Evaluation

PCA is one of the most fundamental tools of dimensionality reduction for extracting effective features from high- dimensional vectors of input data [2].

1. **Standardization:**
   To ensure that all features are on the same scale, we first standardize the data. This is an important step because PCA is sensitive to the scale of the input features.

2. **Initial PCA:**
   We then apply PCA to the standardized data to reduce its dimensionality. By examining the cumulative explained variance ratio, we can determine how much of the variance in the data is explained by the principal components. This is visualized by plotting the cumulative explained variance against the number of components.

3. **Selecting Principal Components:**
   To retain 95% of the variance, PCA is configured to include enough principal components to capture this proportion of the total variance. The number of components required to achieve 95% variance is calculated and plotted.

4. **Model Training and Evaluation:**
   With the reduced feature set obtained from PCA, a LightGBM model is trained and evaluated. The model's performance is assessed using metrics such as Root Mean Squared Error (RMSE) and R-squared.

5. **Hyperparameter Tuning:**
   Further optimization of the LightGBM model is performed using GridSearchCV on the PCA-transformed data to identify the best hyperparameters. The tuned model is then evaluated using the same performance metrics to determine if there is an improvement.

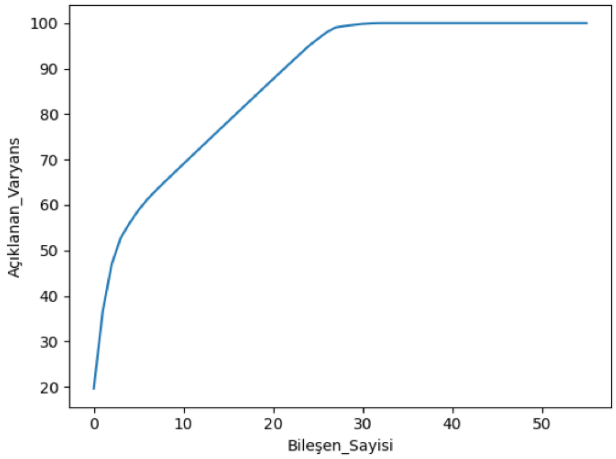

Figure 8: Explained variance

	Model	RMSE	R-squared	Tuned RMSE	Tuned R-squared
0	Classification and Regression Trees (CART)	60.38	0.65	55.27	0.71
1	Bagging (Bootstrap Aggregation)	50.80	0.75	46.34	0.79
2	Random Forest	45.25	0.80	43.78	0.82
3	K-Nearest Neighbors (KNN)	50.25	0.76	47.39	0.78
4	Support Vector Regression (SVR)	93.59	0.16	29.91	0.86
5	Artifical Neural Network (ANN)	39.55	0.85	38.24	0.86
6	Gradient Boosting Machines (GBM)	46.90	0.79	47.38	0.78
7	Extreme Gradient Boosting (XGBoost)	44.60	0.81	44.04	0.81
8	LightGBM	48.32	0.78	47.91	0.78

Figure 9: Non-outlier analysis version

11. Outlier Analysis Results

Outliers can distort statistical analyses and model performance, making their detection and treatment essential for ensuring the accuracy and robustness of the results. This analysis typically involves examining the data using various methods to determine which observations are outliers and how they might affect the overall data distribution.

In this analysis, we employ three key methods for detecting outliers:

- **Non-Outlier Results:**
  This refers to the baseline dataset where no specific outlier detection has been applied. It provides a reference point to compare the impact of outlier detection methods.

By comparing the results obtained from these methods with the non-outlier dataset, we can evaluate how each method affects the data distribution and assess their impact on model performance.

- **Z-Score Method::**
  The Z-score method standardizes data points by calculating how many standard deviations away each observation is from the mean. Observations with Z-scores exceeding a certain threshold (commonly ± 3) are flagged as outliers. This method is useful for identifying outliers in data that follows a roughly normal distribution.

	Model	RMSE	R-squared	Tuned RMSE	Tuned R-squared
0	Classification and Regression Trees (CART)	22.88	0.87	22.33	0.88
1	Bagging (Bootstrap Aggregation)	18.23	0.92	17.64	0.92
2	Random Forest	18.00	0.92	17.69	0.92
3	K-Nearest Neighbors (KNN)	18.05	0.92	17.95	0.92
4	Support Vector Regression (SVR)	48.23	0.42	18.64	0.90
5	Artifical Neural Network (ANN)	14.88	0.94	17.33	0.92
6	Gradient Boosting Machines (GBM)	16.10	0.94	16.56	0.93
7	Extreme Gradient Boosting (XGBoost)	18.66	0.91	16.59	0.93
8	LightGBM	16.75	0.93	15.92	0.94

Figure 10: Z-Score outlier analysis version

- **Interquartile Range (IQR) Method:**
  The IQR method is based on the statistical spread of the middle 50% of the data. It identifies outliers by calculating the range between the first quartile (Q1) and the third quartile (Q3) and then defining outliers as points lying beyond a specified factor of the IQR (typically 1.5 times the IQR). This method is particularly effective for datasets with skewed distributions or non-normal data.

	Model	RMSE	R-squared	Tuned RMSE	Tuned R-squared
0	Classification and Regression Trees (CART)	20.19	0.90	22.15	0.88
1	Bagging (Bootstrap Aggregation)	18.14	0.92	17.52	0.92
2	Random Forest	17.90	0.92	17.50	0.92
3	K-Nearest Neighbors (KNN)	17.94	0.92	17.80	0.92
4	Support Vector Regression (SVR)	47.66	0.42	18.53	0.90
5	Artifical Neural Network (ANN)	15.20	0.94	16.89	0.93
6	Gradient Boosting Machines (GBM)	15.75	0.94	16.49	0.93
7	Extreme Gradient Boosting (XGBoost)	18.22	0.92	16.41	0.93
8	LightGBM	16.63	0.93	15.83	0.94

Figure 11: IQR outlier analysis version

12. Conclusion

In this analysis, we evaluated various machine learning models and their performance on our dataset, assessing metrics such as RMSE and R-squared. The models tested included K-Nearest Neighbors (KNN), Support Vector Regression (SVR), Artificial Neural Networks (ANN), Gradient Boosting Machines (GBM), Extreme Gradient Boosting (XGBoost), and LightGBM.

LightGBM demonstrated the best performance with the lowest RMSE and highest R-squared, indicating its superior predictive accuracy. The application of Principal Component Analysis (PCA) improved model efficiency by reducing dimensionality while maintaining performance levels.

Outlier analysis highlighted the impact of outliers on model accuracy, with results showing how different methods (Z-score and IQR) can influence the identification and treatment of outliers.

Overall, these insights underscore the importance of model selection, hyperparameter tuning, and robust outlier detection in achieving reliable and accurate predictive results.

References

[1] Peruri Venkata Anusha, Ch. Anuradha, Patnala S.R. Chandra Murty, and Ch. Surya Kiran. Detecting outliers in high dimensional data sets using z-score methodology. International Journal of Innovative Technology and Exploring Engineering (IJITEE), 9(1), November 2019.

[2] Annie George. Anomaly detection based on machine learning: Dimensionality reduction using pca and classification using svm. International Journal of Computer Applications, 47(21), June 2012.

[3] MD Tanvir Islam, Eftekhar Hossain Ayon, and Bishnu Padh Ghosh. Revolutionizing retail: A hybrid machine learning approach for precision demand forecasting and strategic decision-making in global commerce. Journal of Computer Science and Technology Studies, 2023.

[4] J. Kang, X. Guo, L. Fang, X. Wang, and Z. Fan. Integration of internet search data to predict tourism trends using spatial-temporal xgboost composite model. International Journal of Geographical Information Science, 36(2):236–252, 2021.

[5] Arnab Mitra, Arnav Jain, Avinash Kishore, and Pravin Kumar. A comparative study of demand forecasting models for a multi-channel retail company: A novel hybrid machine learning approach. Journal of Forecasting, September 2022. © The Author(s), under exclusive licence to Springer Nature Switzerland AG 2022.

[6] Pramod Singh. Machine Learning with PySpark - With Natural Language Processing and Recommender Systems. 2018.