Shared Bicycle Rentals Prediction Project Report

Kim Ki Pyo

04 April 2025

1 Introduction

Bike-sharing systems have gained significant popularity in urban environments, offering a convenient and eco-friendly transportation alternative. Understanding the factors influencing bike rental demand is crucial for optimizing resource allocation and improving service efficiency. This project employs a deep learning-based approach to predict bike-sharing demand by leveraging historical rental data combined with various meteorological and temporal features. By analyzing the relationships between independent variables and rental counts, this study aims to construct a robust predictive model capable of capturing complex demand patterns. This project involves thorough data analysis, preprocessing, and model development, ensuring an effective and reliable bike share rentals forecasting system.

2 Dataset Description

The dataset contains records of a shared bicycle rental service. It includes various factors that may influence rental demand, such as date, weather conditions, seasonality, and holidays. The target variable 'cnt' represents the total number of bicycles rented on a given day. This dataset will be used to analyze patterns and build predictive models for future rental demand, assisting the company in optimizing its operations and maximizing profit. Table 1 shows the columns included in the dataset along with brief descriptions of each feature.

Table 1: Description of dataset features

Column	**Description**
instant	Record index (ID)
yr	Year (0: 2018, 1: 2019)
mnth	Month (1 to 12)
dteday	Date (DD-MM-YYYY)
season	1: Spring, 2: Summer, 3: Fall, 4: Winter
holiday	Whether the day is a holiday (1) or not (0)
workingday	1 if not weekend or holiday, 0 otherwise 1
weathersit	1: Clear, Few clouds, Partly cloudy
	2: Mist + Cloudy, Broken clouds, Few clouds
	3: Light Snow or Rain + Scattered clouds
	4: Heavy Rain/Snow + Thunderstorm + Fog
temp	Temperature in Celsius
atemp	"feels like" temperature
hum	Relative humidity
windspeed	Wind speed
cnt	Total number of bike rentals

3 Data Analysis

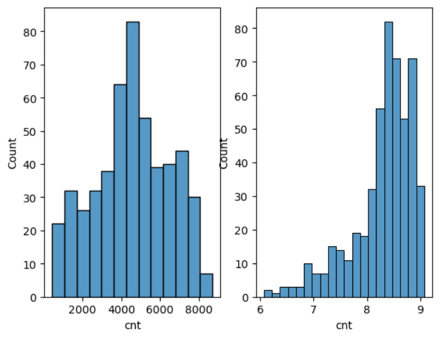


Figure 1: Distribution of target values

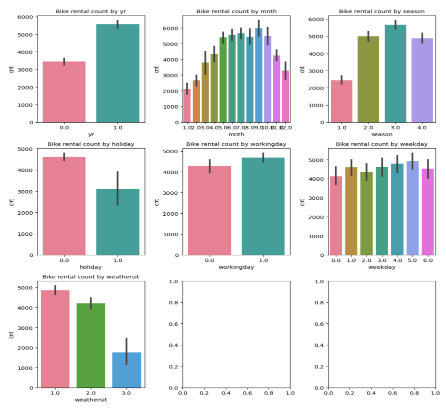


Figure 2: Distribution of "cnt" across categorical features

To analyze the distribution of the target variable (cnt), we compare its raw distribution to its log-transformed counterpart. The left plot in Figure 1 represents the original distribution, while the right plot illustrates the log-transformed version. Upon inspection, the original distribution already exhibits a relatively symmetric shape resembling a normal distribution. Given this observation, applying a log transformation is deemed unnecessary, as it does not provide a significant advantage in normalizing the target variable's distribution.

Figure 2 visualizes the distribution of “cnt” across categorical features such as “yr”, “mnth”, “season”, “holiday”, “workingday”, “weekday”, and “weathersit”. The analysis reveals that “weekday” and “workingday” have relatively uniform distributions of rental counts, suggesting that bike rental demand does not significantly fluctuate based on the day of the week or whether it is a working day. On the other hand, features like “yr”, “mnth”, “season” and “weathersit” exhibit more noticeable variations in rental counts, indicating their stronger influence on demand patterns.

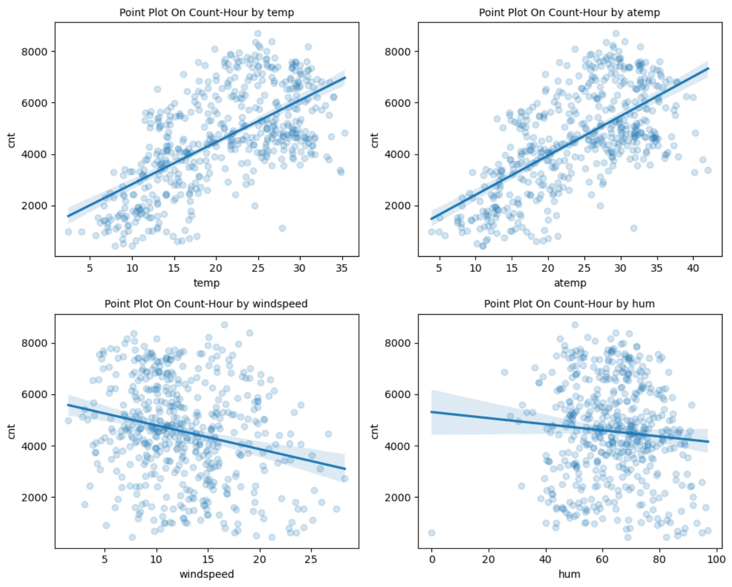


Figure 3: Scatter plot and Regression line of continuous variables and target

Additionally, the relationship between continuous variables (“temp”, “atemp”, “windspeed”, and “hum”) and the target variable is examined using scatter plots with regression lines in Figure 3. The weather-related features show varying degrees of correlation with bike rental demand. Higher values of “temp” and “atemp” generally correspond to increased bike rentals, while “hum” and “windspeed” display weaker, more complex relationships with demand. Since “temp” and “atemp” are closely related and may exhibit strong multicollinearity, further analysis—such as Variance Inflation Factor (VIF) calculations—is recommended to determine whether both should be retained in the model. These steps help identify key factors influencing bike rental demand while ensuring that the dataset is well-prepared for predictive modeling.

4 Preprocessing

The preprocessing steps include:

- Converting 'dteday' to datetime format and extracting temporal features.
- Handling missing values using heuristic methods and CatBoost-based imputation.
- Feature selection using correlation analysis and Variance Inflation Factor (VIF).
- One-hot encoding for categorical variables.

This project started by efficiently applying a series of preprocessing steps to ensure data quality and consistency. Initially, the date column 'dteday' is converted from a string format to Python's datetime format using pandas. This conversion not only standardizes the date representation but also facilitates the extraction of additional features such as the year, month, and weekday, which are crucial for capturing temporal patterns in bike sharing behavior.

To address missing values, several key features are handled systematically. For instance, the missing values in the 'yr' column are filled by checking the corresponding year extracted from the 'dteday' field—assigning a value of 1 for 2019 and 0 for 2018. Similarly, missing values in the 'mnth' and 'weekday' columns are replaced with the month and weekday obtained from the datetime conversion, respectively. This step ensures that basic temporal information is complete and accurate.

A more sophisticated approach is taken for the 'holiday' and 'workingday' columns. A custom function is implemented to fill missing values by considering the weekday of the date. If both fields are missing, the function assigns a value of 0 for holidays and, based on whether the day is a weekend (Saturday or Sunday) or a weekday, sets the working day indicator to 0 or 1 accordingly. If only one of the two fields is missing, the function infers the missing value from the available information, thus preserving the logical relationship between holidays and working days.

To address the missing values in the 'season' variable, a hierarchical rule-based imputation method was developed based on the corresponding date information 'dteday'. The imputation process begins by identifying entries in the dataset that share the same month and day with the observation containing the missing value. If such entries exist and contain non-null season labels, the most frequently occurring season among them is assigned.

In cases where no direct match is available for the same day, the method proceeds to consider all non-null season entries within the same month. By grouping these entries by day, the method computes the mode of the season values for each day and selects the season corresponding to the day closest to that of the missing entry, minimizing the absolute difference in day values.

If neither of the above approaches yields a result, the method defaults to a rule-based approximation of seasons based on astronomical definitions. Specifically, winter is defined as starting from December 21, spring from March 21, summer from June 21, and fall from September 21. For months without a seasonal boundary, general seasonal groupings are applied: January and February are considered winter, April and May as spring, July and August as summer, and October and November as fall.

This multi-step approach ensures that missing season values are imputed in a manner that reflects both temporal proximity and natural seasonal transitions, thereby enhancing the temporal consistency of the dataset.

The numerical weather features—'temp', 'atemp', 'windspeed', and 'hum'—are treated with particular care since it is supposed that these features will give significant information predicting the target value. Rather than using simple imputation methods like mean substitution, these variables are filled using predictions from a CatBoostRegressor model since they are continuous values. The model leverages other related features (such as month, season, year, holiday, weekday, and working day) to predict missing weather values. This method not only provides more accurate estimates but also preserves the underlying relationships in the data. Similarly, the categorical weather-related feature 'weathersit' is imputed using a CatBoostClassifier, ensuring that even the non-numeric, nominal aspects of weather are handled robustly.

Table 2: Top-5 Feature Correlation

Feature 1	**Feature 2**	**Corr.**
temp	atemp	0.99
season	mnth	0.83
weathersit	hum	0.56
season	atemp	0.35
season	temp	0.33

Table 3: VIF of Continuous Features

Feature	**VIF**
atemp	592.04
temp	535.71
hum	12.03
windspeed	5.10

Table 4: Correlation with Target (“cnt”)

Feature	**Corr.**
cnt	1.00
atemp	0.64
temp	0.63
yr	0.55
season	0.44
mnth	0.30
workingday	0.10
weekday	0.08
hum	-0.08
holiday	-0.15
windspeed	-0.23
weathersit	-0.27

Following an extensive exploratory data analysis (EDA), the dataset undergoes a structured feature selection process. Table 2 presents the correlation between input features, while Table 4 highlights their correlation with the target variable. Additionally, Table 3 illustrates the Variance Inflation Factor (VIF) values, which indicate potential multicollinearity issues. Based on these insights, the features 'dteday', 'atemp', 'weekday', and 'season' were identified as redundant or less informative and subsequently removed. This step effectively reduces noise and enhances the dataset's overall quality for modeling.

Subsequently, categorical but nominal features without inherent ordinal relationships—'mnth' and 'weathersit'—are transformed using one-hot encoding. This conversion ensures that the dataset is structured in a format suitable for machine learning models by representing categorical variables as binary indicators. The resulting preprocessed dataset, after excluding redundant features including the original 'instant' identifier, is now ready for further analysis and predictive modeling.

Through these comprehensive data preprocessing steps—ranging from date conversion and missing value imputation to feature selection and one-hot encoding—we ensure that the input data is both reliable and suitably formatted. This meticulous approach lays a solid foundation for the subsequent stages of model development and ultimately contributes to more accurate and robust bike sharing predictions.

5 Model Architecture and Techniques

The model employed for predicting bike sharing demand is a deep neural network designed with multiple fully connected layers to capture complex, non-linear patterns in the data. To ensure robustness and stability during training, the architecture integrates several modern techniques. The network begins with an input layer whose size is determined by the number of features present in the preprocessed dataset. This is followed by a sequence of hidden layers; the first layer consists of 256 neurons, after which batch normalization is applied to stabilize the activations, followed by the GELU activation function to introduce non-linearity. A dropout layer with a rate of 0.3 is then used to reduce the risk of overfitting by randomly deactivating a subset of neurons during each training iteration.

The deep learning model consists of:

- Input layer: Matching the number of features.
- Four hidden layers:
  - Layer 1: 256 neurons, Batch Normalization, GELU activation, Dropout (0.3)
  - Layer 2: 128 neurons, Batch Normalization, LeakyReLU activation, Dropout (0.3)
  - Layer 3: 64 neurons, Batch Normalization, LeakyReLU activation, Dropout (0.3)

  - Layer 4: 32 neurons, Batch Normalization, LeakyReLU activation, Dropout (0.0)

- Output layer: Single neuron with linear activation.

The architecture is repeated with diminishing neuron counts—128 neurons in the second layer, 64 in the third, and 32 in the fourth—each time applying batch normalization, LeakyReLU activation, and dropout with the same rate except for the Layer 4 where the dropout rate is set to 0. This progressive reduction in dimensionality helps in learning increasingly abstract representations while simultaneously reducing the number of parameters and preventing overfitting. The output layer is a single linear neuron designed to produce a continuous output, a bike counts. A notable feature of the model is its use of Xavier normal initialization for the weights in all linear layers. This initialization method helps maintain the variance of activations through the layers, promoting smoother gradients and facilitating faster convergence. Additionally, the inclusion of small constant biases further ensures that the initial network state is well-conditioned. Overall, the model's structure—with careful selection of activation functions, regularization techniques, and initialization—aims to maximize performance while mitigating common pitfalls such as overfitting and vanishing gradients.

6 Training

The training process follows two methodologies:

- Train-validation split (80-20) to monitor overfitting.
- Full dataset training for final submission.

Training parameters include:

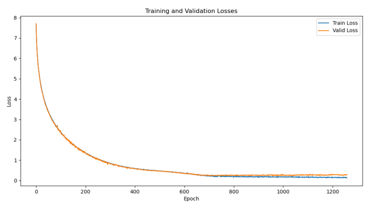
- Optimizer: AdamW (learning rate = $5e^{-3}$, weight decay = $1e^{-3}$)
- Learning Rate Scheduler: Cosine Annealing ($T_{max} = 500$, $\eta_{min} = 5e^{-4}$)
- Loss function: RMSLE
- Training epochs: 3500 with early stopping at 300 epochs of no improvement.

The training process was conducted using two distinct methodologies: one that employs a train-validation split to monitor and control overfitting, and another that utilizes the entire training dataset for final model training.

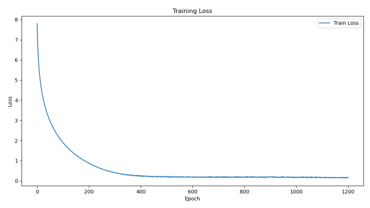
In the first approach, the dataset is split into training and validation sets, with 80% of the data used for training and 20% reserved for validation. The purpose of this division is to monitor the model's performance on unseen data, thereby detecting signs of overfitting early. The training loop for this approach uses an AdamW optimizer with a learning rate of 5e-3 and a weight decay of

1e-3, which is particularly effective in preventing overfitting by penalizing large weights. A Cosine Annealing Learning Rate Scheduler is applied with T_{max} set to 500 and η_{min} at 5e-4, which helps in gradually decreasing the learning rate as training progresses. The loss function chosen is the RMSLE (Root Mean Squared Logarithmic Error), which is well-suited for regression tasks where the target variable spans several orders of magnitude. The RMSLE (Root Mean Squared Logarithmic Error) is calculated as follows:

$$\text{RMSLE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\log(y_i+1)-\log(\hat{y}_i+1)\right)^2}$$



(a) With validation split



(b) Without validation split

Figure 4: RMSLE loss trends graphs

Figure 4a and Figure 4b show the training loss trends with and without a validation split, demonstrating similar loss reductions in both cases. In the first training strategy, the entire epoch is set to 3500. However, the process includes early stopping criteria; if the validation loss does not improve for 300 consecutive epochs, training is halted to prevent unnecessary computation and to mitigate overfitting. The model state corresponding to the best validation loss is saved. The best checkpoint training loss and validation loss was 0.19 and 0.22.

In contrast, the second training strategy foregoes a validation split entirely. In this "no-split" training configuration, all available training data is used to train the model, which is particularly advantageous for final submissions when the primary aim is to leverage every bit of data for learning. The configuration remains largely similar, with the same optimizer, learning rate, scheduler, and RMSLE loss function. However, in this case, the model is trained for a maximum of 1200 epochs without early stopping. In this strategy the final epoch's training loss was 0.16.

In the train-validation split approach, the loss graph in Figure 4a show that training and validation losses decreased nearly simultaneously, indicating that the model was learning effectively without signs of overfitting.

For the method without splitting, the epoch count was determined experimentally, and the model state at the final epoch was used for generating predictions; its loss trajectory is depicted in Figure 4b. Overall, these results confirm that both training configurations achieved robust and consistent performance. The public kaggle competition score is 0.36.

7 Conclusion

This project successfully developed a deep learning-based model for predicting bike-sharing demand, leveraging various meteorological and temporal features. Through extensive data analysis and preprocessing, essential features were identified and refined to optimize model performance. The model architecture incorporated advanced techniques such as batch normalization, dropout regularization, and Xavier initialization to enhance learning efficiency and mitigate overfitting. Training experiments demonstrated the effectiveness of both validation-based and full dataset training strategies, achieving a stable and reliable predictive performance. Future work may focus on enhancing the predictive power of the model by applying more advanced feature engineering techniques, integrating additional contextual information, and increasing the dataset size. By continuously refining these aspects, the model can further improve its accuracy and serve as a valuable tool for optimizing bike-sharing operations.