Bike Sharing Demand

Phudish Prateepamornkul
Department of Statistics
University of California, Berkeley
Berkeley, CA 94704
phudish_p@berkeley.edu

Abstract

Since the rise of the demand of the bike-sharing scheme especially in Seoul. Where it was reported that in 2021, the daily rental number amounted to 87.8 thousand, up from about 65 thousand in the previous year. Additionally, since the start of the service in 2015, the rate has increase by more than twice every year [1]**?**. Therefore, it is essential to make the rental bike available and accessible to the public. The purpose of this paper is therefore to estimate the demand of the bike rental hourly in Seoul. The research involved training six unique statistical regression models including 1.Linear Regression 2. Support Vector Machine 3. Decision Tree 4. Gradient Boosting Machine 5. eXtreme Gradient Boosting and 6. Neural Network on a training dataset. Additionally, the study utilized the grid search technique for identifying optimal hyper parameters through cross-validation. These optimal hyper parameters were then applied to assess the models' performance on a separate test dataset. The most successful outcome was observed with the Gradient Boosting Machine, which has a root mean square training error of 37.38 and an R^2 value of 0.99 in the train data set, a testing error of 157.23, and an R^2 value of 0.94 in the test data set.

1 Background

Bike sharing has become increasingly popular in urban cities as a convenient, affordable and environmentally friendly mode of transportation. It operates as a shared service accessible to individuals by allowing user to borrow a bike from one station and return it to another station.

Currently, Seoul has introduced Ttareungi which is the public bicycle sharing system that can be used by anywhere,anyone and any time. This application has been one of the most popular mode for transportation especially during COVID where people are trying to avoid the crowded of people in public transport such as train and buses. It has been reported that the number of Ttareungi users increased by 24% year on year to 2.78 million in 2020 [2]. Additionally, the bicycle sharing program in Seoul has produces more than 1 millions rides a month with more than 1500 bicycle stations and 20000 bicycles operating in the city [2]. Hence, it is very crucial and important for us to be able to predict the rented bike count accurately to meet the people demand.

2 Data Preparation

2.1 Information of the data

The data set use is publicly available at UC Irvine Machine Learning Repository website `http://archive.ics.uci.edu/dataset/560/seoul+bike+sharing+demand`. The dataset consists of hourly rented bike count one year from 2017 December to 2018 November from Seoul which contains 8760 rows and 14 columns including Date, Rented Bike Count, Hour,

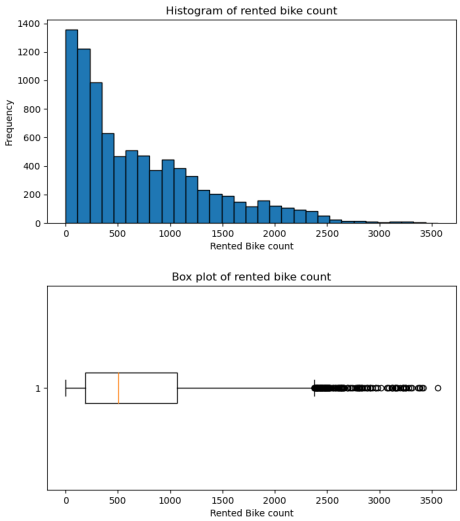

Figure 1: (a) Histogram of Rented Bike count (b) Box plot of Rented Bike count

Temperature(°C), Humidity(%), Wind speed(m/s),Visibility(10m), Dew point temperature(°C), Solar Radiation(MJ/m2), Rainfall(mm), Snowfall(cm), Seasons, Holiday, Functioning Day. Where Seasons(Winter,Spring,Summer,Autumn), Holiday(Yes,No), Functioning Day(Yes,No) are categorical data while the rest are continuous data.

2.2 Exploratory Data Analysis

We want to explore our data first before we do any modelling. Figure 1 shows the Histogram and Box plot of the rented bike count. Clearly, we see that the data is right skewed and there are a few outlier show in the box plot. Therefore, before we put our data into the model we should do some transformation to our target variable first. Figure 2 shows the Histogram of different numerical value in our data. This time we see that most of the features are skew. So we should apply standardization to our features.

2.3 Data preparation and engineering

emphesis on the smaller value more. We started by transforming our response variable first since it is right skew we will apply square root to each of the response so that the distribution look more like normal distribution as shown in Figure 3. Additionally, we should try to utilize most of our features first. We created two more features which are 1.Month (look at the Month of the corresponding Date) 2. Week Type (which is categorical type to look at whether at a given day is it a weekend or weekday). Once we did that then we start to drop our target variable in this case Rented Bike Count column and work with the rest of the features. When dealing with categorical features such as Seasons,Holiday,Functioning Day and Week Type, we use one-hot encoding. Which creates new columns for each category and fill a value with 1 if the record belongs to that category and 0 otherwise.After handling the categorical data we focused on our numerical features. We applied standardization to our numerical features by subtract the mean of the features from each value and divided by the standard deviation. Note that we did not apply standardization to the month. We then split our dataset into training and testing where 25% of the data is use for testing. We have 6570 rows for training data and 2190 rows for testing data.

3 Models

Since we are tackling a regression problem, our chosen metrics for evaluating training and testing errors include 1. Root Mean Square Error (RMSE), 2. R^2 score, and 3. Mean Absolute Error (MAE). To optimize each model, we implemented grid search to determine the best possible input parameters.

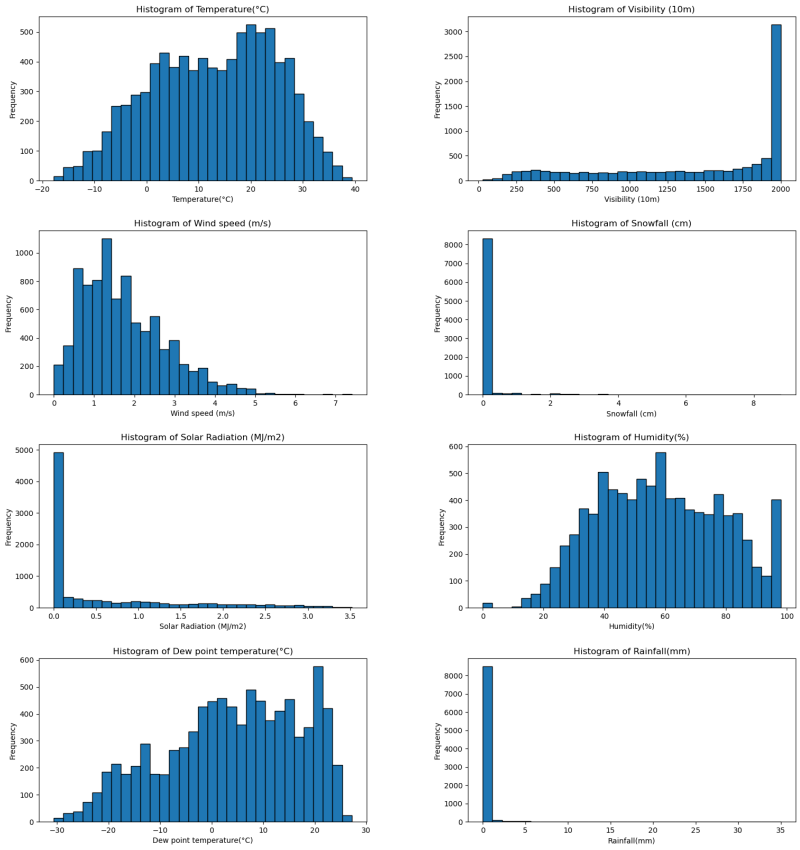

Figure 2: Histogram of numerical features

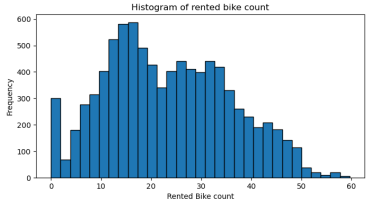

Figure 3: Histogram of square root of response variable

This process involved using 5-fold cross-validation and negative mean squared error as the scoring metric. After identifying the optimal parameters, we trained the models on the training set. The way in which we find the grid search is first we tried a few values where each of them is pretty far from each other first. Then after we find the best one in that set of values then we try again using the best one from the previous time but now having each of them to be closer to each other.

3.1 Linear Regression

Linear Regression started by assuming a linear relationship between our target(Rented Bike Count) which we will call it as y and our feature which we will call it as X. We then want to find a vector of β that minimize $(y - X\beta)^2$. Since there is no parameter we have to input we simply fit our data and report the result we got.

3.2 Support Vector Machine (SVM)

SVM is use for regression by transforming data into a higher dimension feature space so that the we can apply linear regression in this new feature space. We have that there are many kernel we can use to map our data into a higher dimension including linear,polynomial,sigmoid and radial basis function(RBF). In our case we will fix our SVM kernel to be RBF. Additionally, we are going to tune two parameters which are 1.The regularization parameter which tells us how the model would handles errors between predicted values and the actual values in training and 2. Gamma (γ) which is the coefficient in the RBF kernel. After, we did grid search we found that the best parameters are 3200 and 0.5 respectively.

3.3 Decision Tree

Decision Tree regression is a flowchart like tree structure that work by making predictions based on how the previous set of questions were answered. Typically it started by having a root node which branches into possible outcomes. Each of those outcomes also lead to additional node which also branch off into other possibilities. We have that the internal node is the node that symbolizes a choice of our input feature and a leaf node is a node without any child nodes that indicates a numerical value. We are going to tune three parameters which are 1. The maximum depth of the tree 2. The minimum number of samples require to split an internal node and 3. The minimum number of samples required to be at a leaf node. We did grid search and found that the best parameters are 16, 7 and 5 respectively.

3.4 Gradient Boosting Machine (GBM)

Gradient Boosting Machine work by combing a weak learners into a single a stronger learner usually the weak learners are the models that make very few assumptions about the data for example decision trees. This is done by fitting new tree to a residual errors made by existing trees. We then go to the next step by fitting the new tree and then our model will become the first and second tree combine together. We then grow a new tree to fit the residual errors from the combined trees. We are going to tune 5 parameters including 1.number of boosting stages we want to perform 2. the maximum depth of individual regression model 3. the minimum number of samples required to split an internal node 4. the minimum number of samples required at a leaf node and 5. the fraction of samples to be use for fitting the individual base learners. After, grid search we found out that the best parameters are 480, 19, 200, 10 and 0.9 respectively.

3.5 eXtreme Gradient Boosting (XGBoost)

XGBoost applied Gradient Boosting technique like before but the different is that it can use L_1 and L_2 regularization which can help the model to prevent over fitting. We are going to tune 5 parameters including 1. learning rate,2. maximum tree depth for base learners,3. L_1 regularization term on weights,4. L_2 regularization term on weights and 5.number of gradient boosted trees. We found that after grid search the best parameters are 0.1,45,15,27 and 240 respectively.

3.6 Neural Network

We created a simple feed forward Neural Network using ReLU as an activation function. Additionally, we have 3 hidden layers for our model. After grid search we found that the number of node in first,second and third layers are 90,50 and 40 respectively.

4 Result

We compare 6 models by reporting their corresponding performance using RMSE during cross-validation, and RMSE, R^2, and MAE on both the training and testing datasets. Note that the RMSE GridSearch(GS) is not the same as the RMSE Training because the RMSE GS is on the cross-validation instead of the whole dataset like the Training error. Additionally, since we transform our response variable by taking the square root which mean that in our prediction for both training and testing datasets we must square our outcome and the response variable first before we can put it in the metric that use to measure the error. However, note that in the RMSE GS we still report the value of error before squaring the prediction and the actual value.

Model	RMSE (GS)	R² (Train)	MAE (Train)	RMSE (Train)	R² (Test)	MAE (Test)	RMSE (Test)
Linear Regression	-	0.58	280.94	419.74	0.60	271.92	399.62
SVM	4.49	0.99	4.63	5.28	0.88	127.84	214.85
Decision Tree	4.08	0.96	77.93	132.72	0.87	128.59	225.78
GBM	2.78	0.99	23.84	37.38	0.94	85.44	157.23
XGBoost	2.74	0.99	30.86	51.77	0.94	86.19	157.96
Neural Network	4.68	0.70	219.01	355.99	0.69	218.03	352.79

Table 1: Model Evaluation Metrics (GS: GridSearch, Train: Training, Test: Testing)

5 Discussion

From the table we can see that the best model we achieved is the GBM. This is something that is not surprisingly since GBM has been one of the best machine learning model when it come to tabular data. However, one thing that is surprising is the bad performance of the neural network. I think there are probably two reasons for this one being that we are using tabular data which make neural network face many challenges including mixed feature types of our data and a lack of knowledge of the data structure unlike image and text. Secondly,the size of our dataset is pretty small as it is only 8760 rows only. We then compare our result to this paper [3] where they were able to achieved R^2 of 0.96 , RMSE 117.81 and MAE 79.77 for training data and R^2 of 0.92 , RMSE 172.73 and MAE 109.78 for testing data using GBM. While their XGBoost did second best which achieved R^2 of 0.96 , RMSE 127.63 and MAE 85.21 for training data and R^2 of 0.91 , RMSE 183.80 and MAE 119.59 for testing data. This is very similar result to what we got that GBM and XGBoost have been the best models to achieved the best result. I think that there are a few things that we did differently, one being that we have one more extra feature which is the month we get from the date. Additionally, we also perform square root transformation for our response variable and standardization to most of our numerical columns except for the month which we do not think it is appropriate to do so. When we perform transformation to our response we emphasis our loss on smaller value more than the large value because larger value are reduced more significantly than smaller value so during the loss the smaller value will be emphasis more. This is particular useful since most of our values are pretty small. Lastly, we also have that we did more grid search parameters than what they did in their paper. For example, the paper did grid search for boosting iterations and max tree depth for GBM and XGBoost. However, for our case we did more grid search parameters than what they did which make our result better. To improve our result and model I think that we could try to tune more parameters of the input or we can perform more feature selection by removing certain features and test whether the result improve or not.

References

[1] Korea Herald. (January 19, 2022). Average daily number of Seoul Bike (public bike rental system) rentals in South Korea from 2015 to 2021 (in 1,000s) [Graph]. In Statista. Retrieved November 27, 2023, from https://www.statista.com/statistics/997380/south-korea-seoul-bike-daily-rental-number/

[2] Stellar Market Research. (n.d.). South Korea Bike Sharing Market: Industry Analysis and Forecast (2023-2029) by Component, Level of Automation, Fuel Type, and Region. Retrieved November 28, 2023, from https://www.stellarmr.com/report/South-Korea-Bike-Sharing-Market/252.

[3] V E, S., Park, J., Cho, Y. (2020). Using data mining techniques for bike sharing demand prediction in metropolitan city. Computer Communications, 153, 353-366. https://doi.org/10.1016/j.comcom.2020.02.007

Appendix and Code

All of the dataset and code are available at `https://github.com/PhudishTam/STAT254_HW_6.git`code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv("SeoulBikeData.csv",encoding='unicode_escape')
```

```
plt.figure(figsize=(8,4))
plt.hist(df["Rented Bike Count"],bins=31,edgecolor="black")
plt.xlabel("Rented Bike count")
plt.ylabel("Frequency")
plt.title("Histogram of rented bike count")
plt.savefig("histogram")

plt.figure(figsize=(8,4))
plt.boxplot(df["Rented Bike Count"],vert=False)
plt.xlabel("Rented Bike count")
plt.title("Box plot of rented bike count")
plt.savefig('boxplot')
plt.show()

plt.figure(figsize=(8,4))
plt.hist(df["Temperature( C )"],bins=31,edgecolor="black")
plt.xlabel("Temperature( C )")
plt.ylabel("Frequency")
plt.title("Histogram of Temperature( C )")
plt.savefig('histogram_temperature')

plt.figure(figsize=(8,4))
plt.hist(df["Wind speed (m/s)"],bins=31,edgecolor="black")
plt.xlabel("Wind speed (m/s)")
plt.ylabel("Frequency")
plt.title("Histogram of Wind speed (m/s)")
plt.savefig('histogram_wind_speed')

plt.figure(figsize=(8,4))
plt.hist(df["Visibility (10m)"],bins=31,edgecolor="black")
plt.xlabel("Visibility (10m)")
plt.ylabel("Frequency")
plt.title("Histogram of Visibility (10m)")
plt.savefig('histogram_visibility')

plt.figure(figsize=(8,4))
plt.hist(df["Dew point temperature( C )"],bins=31,edgecolor="black")
plt.xlabel("Dew point temperature( C )")
plt.ylabel("Frequency")
plt.title("Histogram of Dew point temperature( C )")
plt.savefig('histogram_dew_point_temperature')

plt.figure(figsize=(8,4))
plt.hist(df["Solar Radiation (MJ/m2)"],bins=31,edgecolor="black")
plt.xlabel("Solar Radiation (MJ/m2)")
plt.ylabel("Frequency")
plt.title("Histogram of Solar Radiation (MJ/m2)")
plt.savefig('histogram_solar_radiation')

plt.figure(figsize=(8,4))
plt.hist(df["Rainfall(mm)"],bins=31,edgecolor="black")
plt.xlabel("Rainfall(mm)")
plt.ylabel("Frequency")
plt.title("Histogram of Rainfall(mm)")
plt.savefig('histogram_rainfall')

plt.figure(figsize=(8,4))
plt.hist(df["Snowfall (cm)"],bins=31,edgecolor="black")
plt.xlabel("Snowfall (cm)")
plt.ylabel("Frequency")
plt.title("Histogram of Snowfall (cm)")
plt.savefig('histogram_snowfall')

plt.figure(figsize=(8,4))
plt.hist(df["Humidity(%)"],bins=31,edgecolor="black")
```

```
plt.xlabel("Humidity(%)")
plt.ylabel("Frequency")
plt.title("Histogram of Humidity(%)")
plt.savefig('histogram_humidity')

df['square'] = np.sqrt(df['Rented Bike Count'].values)
plt.figure(figsize=(8,4))
plt.hist(df["square"],bins=31,edgecolor="black")
plt.xlabel("Rented Bike count")
plt.ylabel("Frequency")
plt.title("Histogram of rented bike count")
plt.savefig("histogram_root")
```

Listing 1: Download all of our plot (image.py)

```
import numpy as np
import pandas as pd
from datetime import datetime
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
from dask_ml.model_selection import GridSearchCV

def preprocess_data(df):
    df['Month'] = pd.to_datetime(df['Date'], format='%d/%m/%Y').dt.
    month
    df['Week_Type'] = df['Date'].apply(lambda x: "Weekend" if datetime
    .strptime(x, '%d/%m/%Y').weekday() > 4 else "Weekday")

    X = df.drop(['Date', 'Rented Bike Count'], axis=1)
    y = df['Rented Bike Count']
    y = np.sqrt(y)

    cat_columns = ['Seasons', 'Holiday', 'Functioning Day', 'Week_Type
    ']
    for col in cat_columns:
        X[col] = X[col].astype('category')

    X = pd.get_dummies(X, columns=cat_columns, dtype=int)
    return X, y

def scale_features(X_train, X_test):
    numeric_columns = ['Temperature( C )', 'Humidity(%)', 'Dew point
    temperature( C )',
                       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', '
    Visibility (10m)',
                       'Snowfall (cm)', 'Wind speed (m/s)']
    scaler = StandardScaler()
    X_train[numeric_columns] = scaler.fit_transform(X_train[
    numeric_columns])
    X_test[numeric_columns] = scaler.transform(X_test[numeric_columns
    ])
    return X_train, X_test

def tune_hyperparameters(estimator, param_grid, X_train, y_train):
    grid_search = GridSearchCV(estimator=estimator, param_grid=
    param_grid, scoring='neg_mean_squared_error', cv=5)
    grid_search.fit(X_train, y_train)
    root_mean_square_error_train = -grid_search.best_score_
    return grid_search.best_params_, root_mean_square_error_train

def evaluate_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    y_pred_train_root = model.predict(X_train)
    y_pred_train = (y_pred_train_root)**2
```

```
    r2_score_train = r2_score(y_train**2, y_pred_train)
    mean_absolute_error_score_train = mean_absolute_error(y_train**2,
    y_pred_train)
    root_mean_square_error_train = np.sqrt(mean_squared_error(y_train
    **2, y_pred_train))
    y_pred_test_root = model.predict(X_test)
    y_pred_test = (y_pred_test_root)**2
    r2_score_test = r2_score(y_test**2, y_pred_test)
    mean_absolute_error_score_test = mean_absolute_error(y_test**2,
    y_pred_test)
    mean_squared_error_test = mean_squared_error(y_test**2,
    y_pred_test)
    root_mean_square_error_test = np.sqrt(mean_squared_error_test)
    return (r2_score_train, mean_absolute_error_score_train,
    root_mean_square_error_train,
            r2_score_test, mean_absolute_error_score_test,
    root_mean_square_error_test)
```

Listing 2: Preprocess the data (preprocess.py)

```
import numpy as np
import pandas as pd
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
from sklearn.linear_model import LinearRegression
from preprocess import preprocess_data,scale_features,
    tune_hyperparameters,evaluate_model

df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
X, y = preprocess_data(df)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.25, random_state=60)
X_train, X_test = scale_features(X_train, X_test)
r2_score_train,mean_absolute_error_score_train,
    root_mean_square_error_train,r2_score_test,
    mean_absolute_error_score_test,root_mean_square_error_test =
    evaluate_model(LinearRegression(),X_train,y_train,X_test,y_test)
print(f"The r2 score on training is : {r2_score_train}")
print(f"The mean absolute error training is : {
    mean_absolute_error_score_train}")
print(f"The root mean square error training is : {
    root_mean_square_error_train}")
print(f"The r2 score on testing is : {r2_score_test}")
print(f"The mean absolute error testing is : {
    mean_absolute_error_score_test}")
print(f"The root mean square error testing is : {
    root_mean_square_error_test}")
```

Listing 3: Linear Model (lm.py)

```
import numpy as np
import pandas as pd
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
from sklearn.svm import SVR
from dask.distributed import Client, LocalCluster
from dask_ml.model_selection import GridSearchCV
```

```
from preprocess import preprocess_data,scale_features,
    tune_hyperparameters,evaluate_model

if __name__ == "__main__":
    df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
    X, y = preprocess_data(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.25, random_state=60)
    X_train, X_test = scale_features(X_train, X_test)
    cluster = LocalCluster(n_workers=32, threads_per_worker=3)
    client = Client(cluster)
    t0 = time.time()
    estimator = SVR(kernel = 'rbf')
    param_test_1 = {'C': [3200,3500,3800],'gamma': [1.5,1,0.5]}
    best_params_svm, best_score_svm = tune_hyperparameters(estimator,
    param_test_1, X_train, y_train)
    print(f"Best paramaeters: {best_params_svm}")
    print(f"The root mean square error training is: {np.sqrt(
    best_score_svm)}")
    best_model = SVR(C= best_params_svm['C'],gamma = best_params_svm["
    gamma"])
    r2_score_train,mean_absolute_error_score_train,
    root_mean_square_error_train,r2_score_test,
    mean_absolute_error_score_test,root_mean_square_error_test =
    evaluate_model(best_model,X_train,y_train,X_test,y_test)
    print(f"Time it took: {time.time() - t0}")
    print(f"The r2 score on training is : {r2_score_train}")
    print(f"The mean absolute error training is : {
    mean_absolute_error_score_train}")
    print(f"The root mean square error training is : {
    root_mean_square_error_train}")
    print(f"The r2 score on testing is : {r2_score_test}")
    print(f"The mean absolute error testing is : {
    mean_absolute_error_score_test}")
    print(f"The root mean square error testing is : {
    root_mean_square_error_test}")
    cluster.close()
```

Listing 4: SVM Model (svm.py)

```
import numpy as np
import pandas as pd
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
from sklearn.tree import DecisionTreeRegressor
from dask.distributed import Client, LocalCluster
from dask_ml.model_selection import GridSearchCV
from preprocess import preprocess_data,scale_features,
    tune_hyperparameters,evaluate_model

if __name__ == "__main__":
    df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
    X, y = preprocess_data(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.25, random_state=60)
    X_train, X_test = scale_features(X_train, X_test)
    cluster = LocalCluster(n_workers=32, threads_per_worker=3)
    client = Client(cluster)
    t0 = time.time()
    estimator = DecisionTreeRegressor()
```

```
    param_test_1 = {'max_depth': range(1, 100),'min_samples_split':
    range(2,10),'min_samples_leaf':range(1,9)}
    best_params, best_score = tune_hyperparameters(estimator,
    param_test_1, X_train, y_train)
    print(f"Best paramaeters: {best_params}")
    print(f"The The root mean sqaure error training is: {np.sqrt(
    best_score)}")
    best_model = DecisionTreeRegressor(max_depth = best_params['
    max_depth'],min_samples_split = best_params['min_samples_split'],
    min_samples_leaf = best_params['min_samples_leaf'])
    r2_score_train,mean_absolute_error_score_train,
    root_mean_square_error_train,r2_score_test,
    mean_absolute_error_score_test,root_mean_square_error_test =
    evaluate_model(best_model,X_train,y_train,X_test,y_test)
    print(f"Time it took: {time.time() - t0}")
    print(f"The r2 score on training is : {r2_score_train}")
    print(f"The mean absolute error training is : {
    mean_absolute_error_score_train}")
    print(f"The root mean square error training is : {
    root_mean_square_error_train}")
    print(f"The r2 score on testing is : {r2_score_test}")
    print(f"The mean absolute error testing is : {
    mean_absolute_error_score_test}")
    print(f"The root mean square error testing is : {
    root_mean_square_error_test}")
    cluster.close()
```

Listing 5: Decision Tree Model (decision_tree.py)

```
import numpy as np
import pandas as pd
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
from sklearn.ensemble import GradientBoostingRegressor
from dask.distributed import Client, LocalCluster
from dask_ml.model_selection import GridSearchCV
from preprocess import preprocess_data,scale_features,
    tune_hyperparameters,evaluate_model

if __name__ == "__main__":
    df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
    X, y = preprocess_data(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.25, random_state=60)
    X_train, X_test = scale_features(X_train, X_test)
    cluster = LocalCluster(n_workers=32, threads_per_worker=3)
    client = Client(cluster)
    t0 = time.time()
    estimator = GradientBoostingRegressor(learning_rate=0.1,
    min_samples_split=500,min_samples_leaf=50,max_depth=8,subsample
    =0.8,random_state=10)
    param_test1 = {'n_estimators': range(20,500, 20)}
    best_params_n_estimators, best_score_n_estimators =
    tune_hyperparameters(estimator, param_test1, X_train, y_train)
    print(f"Best paramaeters: {best_params_n_estimators}")
    print(f"The root mean sqaure error training is: {np.sqrt(
    best_score_n_estimators)}")
    estimator = GradientBoostingRegressor(learning_rate=0.1,
    n_estimators=best_params_n_estimators['n_estimators'],
    min_samples_leaf = 50, subsample=0.8,random_state=10)
```

```
    param_test2 = {'max_depth': range(5, 20, 2), 'min_samples_split': range(200, 1500, 200)}
    best_params_max_depth_min_samples_split, best_score_max_depth_min_samples_split = tune_hyperparameters(estimator, param_test2, X_train, y_train)
    print(f"Best parameters: {best_params_max_depth_min_samples_split}")
    print(f"The root mean sqaure error training is: {np.sqrt(best_score_max_depth_min_samples_split)}")
    estimator = GradientBoostingRegressor(n_estimators=best_params_n_estimators['n_estimators'],max_depth = best_params_max_depth_min_samples_split["max_depth"], min_samples_split = best_params_max_depth_min_samples_split["min_samples_split"],random_state=10)
    param_test3 = {'min_samples_leaf': range(10, 70, 5),'subsample': [0.6,0.7,0.75,0.8,0.85,0.9]}
    best_params_min_samples_leaf_subsample, best_score_min_samples_leaf_subsample = tune_hyperparameters(estimator, param_test3, X_train, y_train)
    # feature_importances = best_model.feature_importances_
    # features_and_importances = zip(X_train.columns, feature_importances)
    # sorted_features_and_importances = sorted(features_and_importances, key=lambda x: x[1], reverse=True)
    # print("Feature Importances:")
    # for feature, importance in sorted_features_and_importances:
    #     print(f"{feature}: {importance}")
    print(f"Best parameters: {best_params_min_samples_leaf_subsample}")
    print(f"The The root mean sqaure error training is: {np.sqrt(best_score_min_samples_leaf_subsample)}")
    best_model = GradientBoostingRegressor(n_estimators=best_params_n_estimators['n_estimators'],max_depth = best_params_max_depth_min_samples_split["max_depth"], min_samples_split = best_params_max_depth_min_samples_split["min_samples_split"],min_samples_leaf=best_params_min_samples_leaf_subsample["min_samples_leaf"], subsample = best_params_min_samples_leaf_subsample['subsample'], random_state=10)
    r2_score_train,mean_absolute_error_score_train, root_mean_square_error_train,r2_score_test, mean_absolute_error_score_test,root_mean_square_error_test = evaluate_model(best_model,X_train,y_train,X_test,y_test)
    print(f"Time it took: {time.time() - t0}")
    print(f"The r2 score on training is : {r2_score_train}")
    print(f"The mean absolute error training is : {mean_absolute_error_score_train}")
    print(f"The root mean square error training is : {root_mean_square_error_train}")
    print(f"The r2 score on testing is : {r2_score_test}")
    print(f"The mean absolute error testing is : {mean_absolute_error_score_test}")
    print(f"The root mean square error testing is : {root_mean_square_error_test}")
    cluster.close()

    cluster.close()
```

Listing 6: GBM Model (gbm.py)

```
import numpy as np
import pandas as pd
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
import xgboost as xgb
from dask.distributed import Client, LocalCluster
from dask_ml.model_selection import GridSearchCV
from preprocess import preprocess_data,scale_features,
    tune_hyperparameters,evaluate_model

if __name__ == "__main__":
    df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
    X, y = preprocess_data(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.25, random_state=60)
    X_train, X_test = scale_features(X_train, X_test)
    cluster = LocalCluster(n_workers=32, threads_per_worker=3)
    client = Client(cluster)
    t0 = time.time()
    estimator = xgb.XGBRegressor()
    param_test_1 = {'learning_rate': [1,0.1],'max_depth'
    :[45,50,55,60],'n_estimators': [200,220,240],'reg_alpha':
    [15,16,17,18],'reg_lambda': [25,27,29,31]}
    best_params, best_score = tune_hyperparameters(estimator,
    param_test_1, X_train, y_train)
    print(f"Best paramaeters: {best_params}")
    print(f"The root mean square error training is: {np.sqrt(
    best_score)}")
    best_model = xgb.XGBRegressor(learning_rate = best_params['
    learning_rate'],max_depth = best_params['max_depth'],n_estimators
    = best_params['n_estimators'],reg_alpha = best_params['reg_alpha']
     , reg_lambda = best_params['reg_lambda'] )
    r2_score_train,mean_absolute_error_score_train,
    root_mean_square_error_train,r2_score_test,
    mean_absolute_error_score_test,root_mean_square_error_test =
    evaluate_model(best_model,X_train,y_train,X_test,y_test)
    print(f"Time it took: {time.time() - t0}")
    print(f"The r2 score on training is : {r2_score_train}")
    print(f"The mean absolute error training is : {
    mean_absolute_error_score_train}")
    print(f"The root mean square error training is : {
    root_mean_square_error_train}")
    print(f"The r2 score on testing is : {r2_score_test}")
    print(f"The mean absolute error testing is : {
    mean_absolute_error_score_test}")
    print(f"The root mean square error testing is : {
    root_mean_square_error_test}")
    cluster.close()
```

Listing 7: XGB Model (xgb.py)

```
import numpy as np
import pandas as pd
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error,
    mean_absolute_error
import torch
from dask_ml.model_selection import GridSearchCV
from preprocess import preprocess_data,scale_features,evaluate_model
from torch import nn
from skorch import NeuralNetRegressor

df = pd.read_csv("SeoulBikeData.csv", encoding='unicode_escape')
X, y = preprocess_data(df)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=60)
X_train, X_test = scale_features(X_train, X_test)
device = "cuda" if torch.cuda.is_available() else "cpu"
X_train = torch.tensor(X_train.values, dtype=torch.float32)
X_test = torch.tensor(X_test.values, dtype= torch.float32)
y_train = torch.tensor(y_train.values,dtype= torch.float32).unsqueeze(dim = 1)
y_test = torch.tensor(y_test.values,dtype= torch.float32).unsqueeze(dim = 1)
X_train = X_train.to(device)
X_test = X_test.to(device)
y_train = y_train.to(device)
y_test = y_test.to(device)

class NeuralNetwork(nn.Module):
    def __init__(self, input_features = 20, out_features = 1, hidden_units_1 = 10, hidden_units_2 = 10, hidden_units_3 = 10):
        super().__init__()
        self.linear_layer_stack = nn.Sequential(
            nn.Linear(in_features=input_features, out_features=hidden_units_1),
            nn.ReLU(),
            nn.Linear(in_features=hidden_units_1, out_features=hidden_units_2),
            nn.ReLU(),
            nn.Linear(in_features=hidden_units_2, out_features=hidden_units_3),
            nn.ReLU(),
            nn.Linear(in_features=hidden_units_3, out_features=out_features)
        )
    def forward(self, x):
        return self.linear_layer_stack(x)


t0 = time.time()
net = NeuralNetRegressor(
    NeuralNetwork,
    criterion=torch.nn.MSELoss,
    optimizer=torch.optim.Adam,
    max_epochs=10,
    device=device,
    verbose=0
)
params = {
    'module__hidden_units_1': [50,60,70,90],
    'module__hidden_units_2': [30,40,50,70],
    'module__hidden_units_3': [10,20,30,40],
}
best_model = GridSearchCV(net, params, cv=5, scoring='neg_mean_squared_error',n_jobs = -1)
X_train = X_train.cpu()
y_train = y_train.cpu()
X_test = X_test.cpu()
y_test = y_test.cpu()
r2_score_train,mean_absolute_error_score_train,root_mean_square_error_train,r2_score_test,mean_absolute_error_score_test,root_mean_square_error_test = evaluate_model(best_model,X_train,y_train,X_test,y_test)
print(f"Time it took: {time.time() - t0}")
print(f"The best parameters: {best_model.best_params_}")
print(f"The root mean sqaure error training is : {np.sqrt(-best_model.best_score_)}")
print(f"The r2 score on training is : {r2_score_train}")
```

```
66 print(f"The mean absolute error training is : {
       mean_absolute_error_score_train}")
67 print(f"The root mean square error training is : {
       root_mean_square_error_train}")
68 print(f"The r2 score on testing is : {r2_score_test}")
69 print(f"The mean absolute error testing is : {
       mean_absolute_error_score_test}")
70 print(f"The root mean square error testing is : {
       root_mean_square_error_test}")
```

Listing 8: Neural Network Model (neural_network.py)