Stage 1 – Machine Learning Project

Metro Traffic Volume Forecasting

MAZEPA Noémie
MORLET Lorrain
MARCELINO Auriane
MARTIN Aymeric

ESILV DIA5

Contents

1 Business Scope

This dataset is designed to forecast public transport demand, considering factors like time of day, holidays, events, and weather, to help agencies adjust schedules and service levels in real-time.

The demand for public transport is influenced by multiple factors that vary in impact, creating a complex forecasting challenge. Real-time adjustments require managing both temporal patterns and external variables, such as weather, adding layers of operational and analytical complexity.

The core problem is to predict user demand accurately and in real-time, enabling transport agencies to optimize services, reduce overcrowding, and adapt schedules based on fluctuating demand patterns.

2 Problem Formalization and Methods

2.1 Algorithm Description

The algorithm starts with the EDA (Exploratory data analysis), we upload the CSV file and display some information about it. We printed the 10 first lines, the type of each column, some statistics about the dataset and the number of null values.

Preprocessing

Then it is the Pre-process, the goal is to clean and modify the dataset so that it can be used. We started changing the column `holiday` because it contained the name of the public holiday, so we just affected a 1 when it is public holiday and 0 when it's not. We categorize data in function of the season and the `partoftheday` (morning, afternoon...) and we class the `weatherdescription` for 10 to 1 depending on if the weather is good (10) or bad (1).

After that we made Visualization of the data with matplotlib in order to see the distribution over the day/week/year, and to see if the weather has an impact on the traffic. Finally, we plot the Correlation Matrix to view the correlation between columns.

Train-Test Splitting

After all these steps, we started the Train-test on different models. We start splitting the dataset into two parts: one to train the model and the rest to test it. Then we created a pipeline through which we will train our models, and we used different models to test which one would fit better.

Evaluation Metrics

To observe the model's performance, we will calculate the R^2 score, which indicates if the model has good predictions if it is close to 1 or bad predictions when the score is closer to 0. We will also evaluate the Mean Squared Error (MSE), which quantifies the average squared difference between the predicted and actual values, giving us insight into the model's prediction accuracy. Additionally, we will compare the models' performance on both the training and test datasets to assess for any signs of overfitting or underfitting, ensuring that our chosen model generalizes well to new, unseen data.

Visualizing underfitting and overfitting is crucial for understanding model performance. Underfitting occurs when a model is too simple to capture the data's patterns, leading to poor performance on both training and test data. However, overfitting happens when a model is too complex, performing well on training data but poorly on test data. By plotting learning curves and comparing training and validation errors, we can identify these issues and ensure better generalization and more reliable predictions.

We will also use cross-validation to evaluate our model. Cross-validation helps us better evaluate our model by splitting the data into multiple parts and training the model on different combinations of these parts. This way, every piece of data gets a chance to be in both the training and testing sets. By doing this, we get a more reliable measure of how well our model performs and can avoid overfitting.

2.2 Models Tested

Here are the models we will use to test which one fit better:

- **RandomForestRegressor:** This ensemble method combines multiple decision trees to minimize overfitting and handle complex data interactions, making it ideal for capturing non-linear relationships in traffic patterns.

- **GradientBoostingRegressor:** This boosting algorithm builds trees sequentially, with each one correcting the errors of the previous one. It effectively captures difficult patterns and is robust against outliers.

- **LinearRegression:** A simple but powerful model that assumes a linear relationship between the features and the target variable. It acts as a baseline and performs well when the relationships in the data are somewhat linear.

- **Ridge:** A regularized form of linear regression that adds a penalty to the size of the coefficients, helping to avoid overfitting. It's particularly useful when features are highly correlated.
- **KNeighborsRegressor:** This algorithm predicts traffic volume by averaging the values of the k-nearest neighbors. It's effective for capturing local patterns and can handle non-linear relationships.

We finish comparing the Mean Squared Error (MSE) and the R^2 score of each model.

3 Limitations

Linear Regression and Ridge Regression provided a simple, interpretable starting point. However, their poor performance ($R^2 = 0.388$) showed they were insufficient for capturing non-linear relationships in the data.

Gradient Boosting Regressor (GBR) and Random Forest Regressor (RFR) were highly effective, with R^2 scores of 0.970 and 0.968. These models excelled at handling feature interactions and non-linear patterns but required computational resources and careful hyperparameter tuning to avoid overfitting.

K Neighbors Regressor (KNN) performed moderately ($R^2 = 0.943$), relying on proximity-based predictions. However, it struggled with scaling and sensitivity to noisy or high-dimensional data.

4 Methodology

4.1 Data Description and Exploration

Missing Values

The missing values we had were in relation to vacations. For example, if the 14th of July 2019 was a holiday, only the statement for the 14th of July 2019 at 00:00 was noted as a holiday. We therefore had to modify some of the data so that all the other readings for the day would be considered holidays.

Also, we only had the date and the hour, so we had columns for the season and the moment of the day because the initial parameters were too precise.

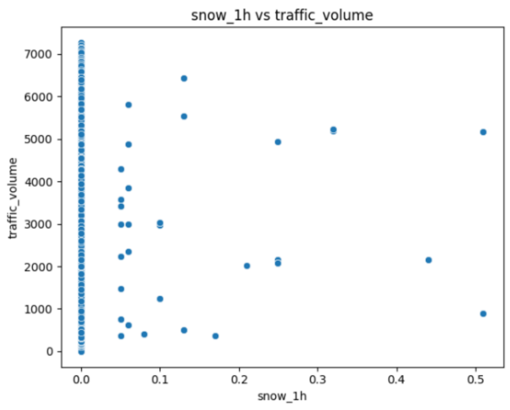


Figure 1: Snow1h vs TrafficVolume

To have a better estimation of the traffic, we would have others information like the air pollution, the remote working rate, if there was a nearby sporting or cultural event and if an accident occurred during the day.

Imbalanced Data

We can see that only fee days in the year are holiday (61 over more than 2,000 days). Non-holiday days are much more numerous than holiday days, we can therefore say that the holiday column has an imbalance.

Also, we see that it rarely snows (63 over more than 48,000 row) and rarely rains (3467 / 48,000). These two columns contain imbalanced data too. We can see on this graph that most surveys have a snowfall of 0.0:

The other columns contain data that are balanced.

Outliers

Our data set contained several outliers particularly in these four columns:

These graphics show the relationship between weather variables and traffic volume. For temperature, traffic is concentrated in a normal range (around -10 to 30°C), with outliers at extreme negative temperatures, which might be data errors: first around -270°C which is definitely not normal but also towards -20/-30°C which is very low for an American city. For rain, traffic is mostly unaffected by rainfall, but there's a significant outlier with extremely high rainfall (around 10,000 mm), likely an error as well. There are no trends for snow and cloud cover, though snow volume is typically minimal, around 0mm.

After removing all the outliers, we obtain the following graphs:

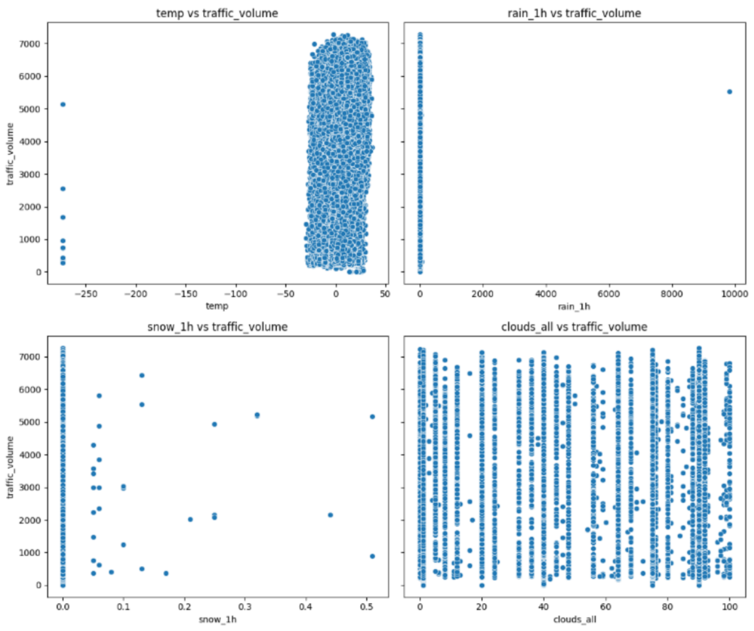


Figure 2: Correlation with TrafficVolume before removing the outliers

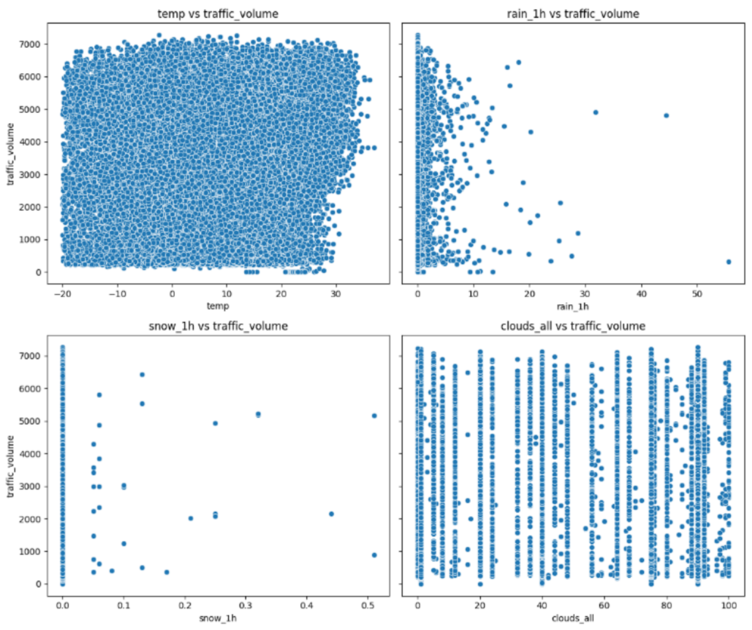
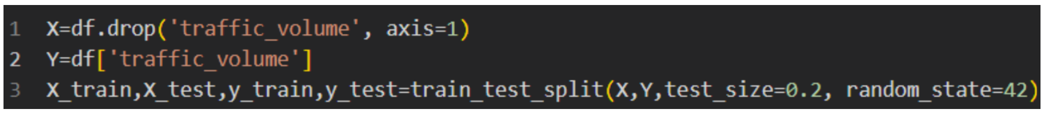


Figure 3: Correlation with TrafficVolume after removing the outliers

4.2 Data Splitting

Separating the dataset into X and y in an essential step because we want to train our model by clearly telling it the column it needs to predict, then test it on a sample for which it doesn't know the answers.

For that, we use, as usual, the `traintestsplit()` method: We decided



```
X=df.drop('traffic_volume', axis=1)
Y=df['traffic_volume']
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.2, random_state=42)
```

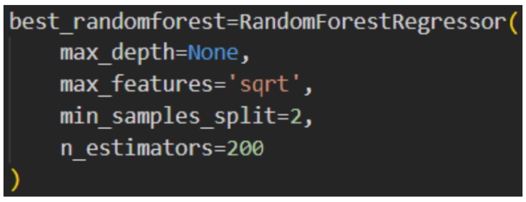
Figure 4: traintestsplit()

to keep only 20 percent of the dataset for the final test (`testsize`=0.2) in order to train our models with a big amount of data and keep some to check if it is well trained.

`randomstate`=42 ensures that the same rows are selected for the train and the test each time it is run.

4.3 Algorithm Implementation and Hyperparameters

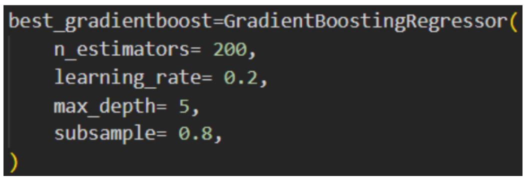
To find the bests hyperparameters, we didn't choose randomly, we used Grid-SearchCV() and we found this:



```
best_randomforest=RandomForestRegressor(
    max_depth=None,
    max_features='sqrt',
    min_samples_split=2,
    n_estimators=200
)
```

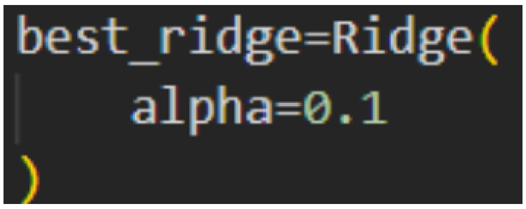
Figure 5: RandomForest

Linear Regression doesn't have any hyperparameters.



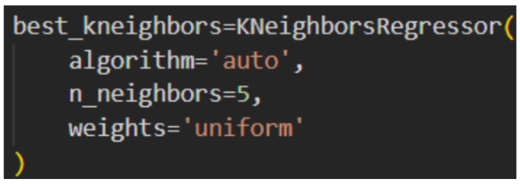
```
best_gradientboost=GradientBoostingRegressor(
    n_estimators= 200,
    learning_rate= 0.2,
    max_depth= 5,
    subsample= 0.8,
)
```

Figure 6: GradientBoosting Hyperparameters



```
best_ridge=Ridge(
    alpha=0.1
)
```

Figure 7: Ridge Hyperparameters



```
best_kneighbors=KNeighborsRegressor(
    algorithm='auto',
    n_neighbors=5,
    weights='uniform'
)
```

Figure 8: KNeighbors Hyperparameters

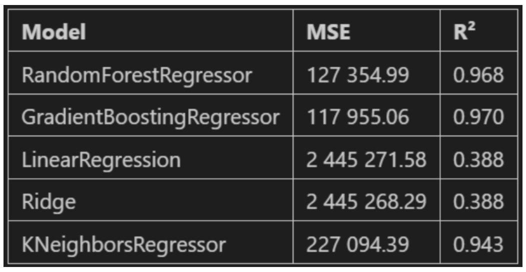
5 Results

5.1 Metrics

As metrics we used the Mean Squared Error (MSE) which measures the average squared difference between the predicted value and the real value, and the R^2 score which indicates the proportion of variance in the dependent variable explained by the model. Here are the values of the MSE and R^2 score corresponding to each model:

5.2 Overfitting

We didn't observe significant overfitting, as we can see in consistent performance across training and testing datasets. Regularization techniques and careful parameter tuning likely mitigated this risk.



Model	MSE	R^2
RandomForestRegressor	127 354.99	0.968
GradientBoostingRegressor	117 955.06	0.970
LinearRegression	2 445 271.58	0.388
Ridge	2 445 268.29	0.388
KNeighborsRegressor	227 094.39	0.943

Figure 9: Comparison MSE and R2 score

5.3 Evaluation

We evaluated the models using learning curve and holdout test sets and we compared the MSE and R^2 with graphs.

6 Discussion and Conclusion

These results suggest that ensemble methods like GradientBoostingRegressor and RandomForestRegressor are more effective for this particular dataset compared to k-nearest neighbors and linear models which failed due to the inherent complexity of the dataset, which includes non-linear and interaction effects. The GradientBoostingRegressor stands out as the top performer, making it the preferred choice for this regression task.