Huber Regression: A Smarter Approach to Handle Outliers in Machine Learning

AlgoStalk Follow 8 min read · Nov 6, 2024

If you prefer visual understanding, check out my 8 mins video link on the **Huber Regressor!**

Introduction 🌟

In traditional linear regression, extreme data points — known as outliers — can heavily skew predictions. But what if we could create a model that's less sensitive to these outliers without entirely ignoring them? Enter **Huber Regression**: a hybrid solution that adapts to outliers while preserving the accuracy of regular data points. This article will guide you through how Huber Regression works, the math behind it, and how to implement it with visualisations.

1. What's the Problem with Outliers in Linear Regression? 🤔

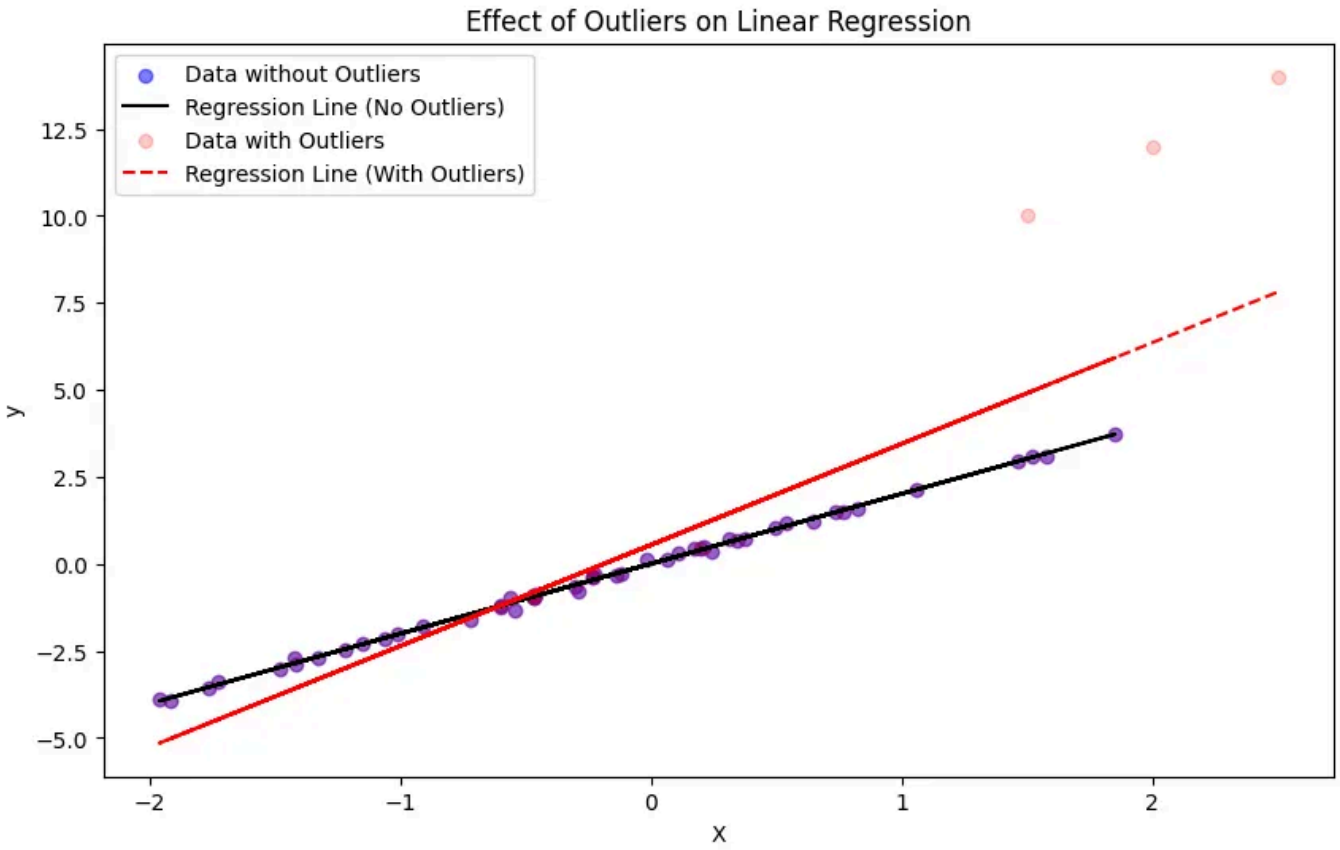
Outliers are data points that differ significantly from the majority of a dataset. While sometimes informative, outliers can disrupt the prediction quality in standard regression models.

Example Visualization: Outliers and Linear Regression

Here's a visual comparison between a simple linear regression model and a dataset with outliers. Let's use Python to show how these outliers can affect the regression line.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generate synthetic data
np.random.seed(42)
X = np.random.normal(0, 1, 50).reshape(-1, 1)
y = 2 * X.flatten() + np.random.normal(0, 0.1, 50)

# Add outliers
X_outliers = np.append(X, [[1.5], [2], [2.5]])
y_outliers = np.append(y, [10, 12, 14])

# Plot Linear Regression with and without outliers
model = LinearRegression()
plt.figure(figsize=(10, 6))

# Without outliers
model.fit(X, y)
plt.scatter(X, y, color="blue", label="Data without Outliers")
plt.plot(X, model.predict(X), color="black", label="Regression Line (No Outliers

# With outliers
model.fit(X_outliers.reshape(-1, 1), y_outliers)
plt.scatter(X_outliers, y_outliers, color="red", label="Data with Outliers")
plt.plot(X_outliers, model.predict(X_outliers.reshape(-1, 1)), color="red", line

plt.legend()
plt.xlabel("X")
```

```
plt.ylabel("y")
plt.title("Effect of Outliers on Linear Regression")
plt.show()
```



2. Introducing Huber Loss Function 📉

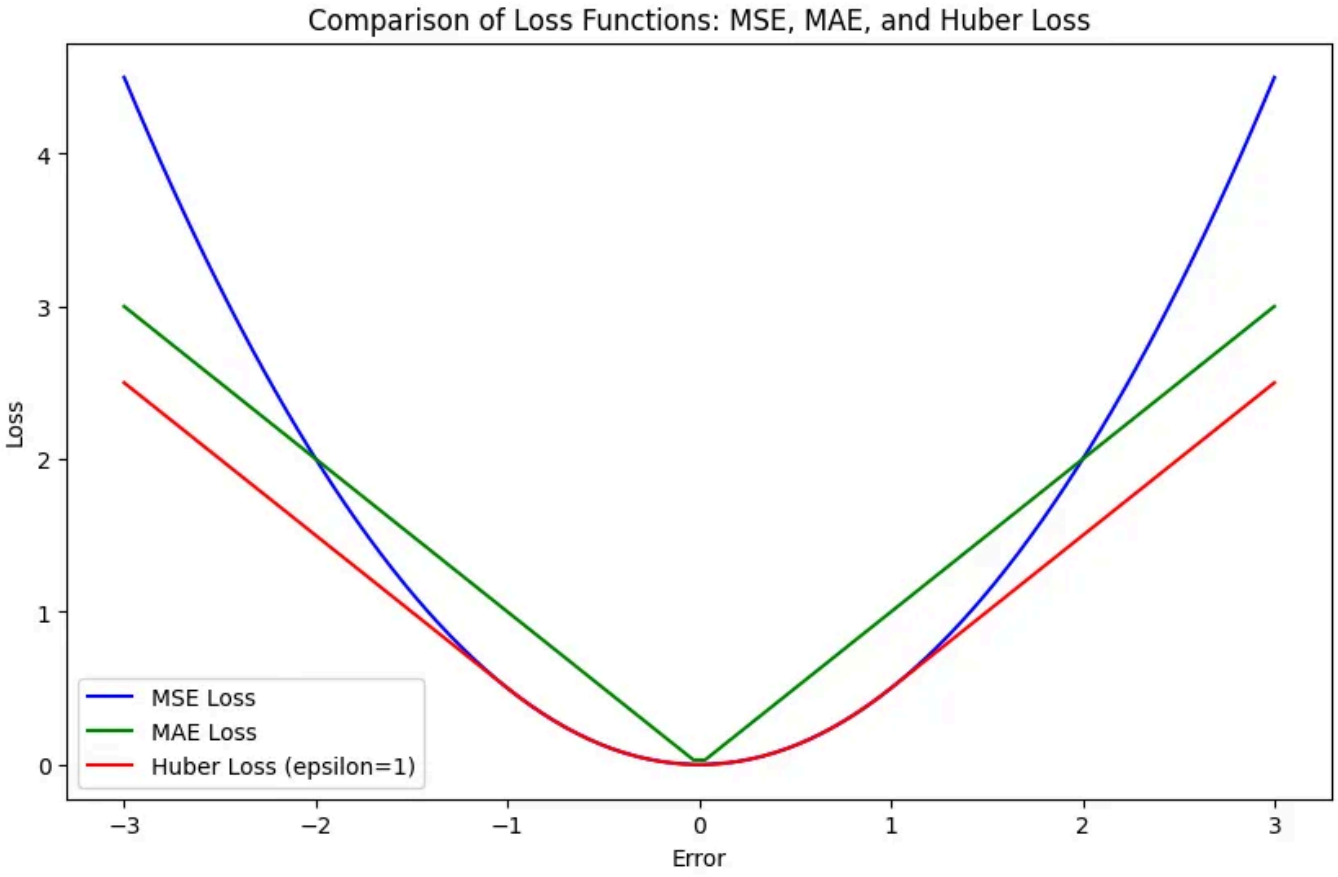
Huber Regression modifies the linear regression loss function to reduce the impact of outliers. In its core, **Huber Loss** combines the strengths of Mean Squared Error (MSE) and Mean Absolute Error (MAE), switching from MSE to MAE when errors exceed a specific threshold, known as epsilon (ϵ).

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y - f(x)| \leq \delta \\ \delta \cdot (|y - f(x)| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

Visualising Huber Loss vs. MSE and MAE

We can compare the loss functions to see how Huber Loss adjusts error handling based on ϵ.

```
def huber_loss(y_true, y_pred, epsilon=1):
    error = y_true - y_pred
    loss = np.where(np.abs(error) <= epsilon, 0.5 * error ** 2, epsilon * (np.ab
    return loss

# Plotting Huber Loss vs. MSE and MAE
epsilon = 1
errors = np.linspace(-3, 3, 100)
huber_losses = [huber_loss(e, 0, epsilon) for e in errors]
mse_losses = 0.5 * errors ** 2
mae_losses = np.abs(errors)

plt.figure(figsize=(10, 6))
plt.plot(errors, mse_losses, label="MSE Loss", color="blue")
plt.plot(errors, mae_losses, label="MAE Loss", color="green")
plt.plot(errors, huber_losses, label=f"Huber Loss (epsilon={epsilon})", color="r
plt.xlabel("Error")
plt.ylabel("Loss")
plt.legend()
plt.title("Comparison of Loss Functions: MSE, MAE, and Huber Loss")
plt.show()
```



3. How Huber Regression Works Step-by-Step 🔍

Huber Regression applies the Huber loss function iteratively, using several parameters to fine-tune the model:

- **Epsilon (ϵ)**: the threshold where Huber Loss switches from quadratic to linear.
- **Alpha**: acts as a regularisation term, minimising the impact of large coefficients and helping with model generalisation.
- **Learning Rate**: controls the step size during coefficient updates, preventing the model from overshooting.
- **Median Absolute Deviation (MAD)**: estimates the scale of the data, particularly useful when there are outliers.

Example: Step-by-Step Visualisation of the Huber Loss and Model Update Process

To understand **Huber Regression**, we first need to discuss two key concepts: **Median Absolute Deviation (MAD)** and the **scaling of residuals.** Huber Regression combines the benefits of **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**, allowing for robust regression by minimising the influence of outliers. Let's break down the process step by step using the provided code.

Step 1: Initialize Parameters

In Huber Regression, we start by defining parameters that will guide the regression process:

```
epsilon = 1.35  # Threshold for Huber loss
alpha = 0.0001  # Regularization strength
max_iter = 100  # Maximum iterations
tol = 1e-5  # Tolerance for convergence
n_samples, n_features = X_train_with_intercept.shape

# Initialize coefficients (including intercept)
coef = np.zeros(n_features)
```

- **`epsilon`** : This value sets the threshold for determining which residuals are treated as outliers.
- **`alpha`** : This controls the amount of regularization applied to the regression model.
- **`max_iter`** and **`tol`** : These determine how many times the algorithm will run and when to stop if changes are minimal.

Step 2: Generate Synthetic Data

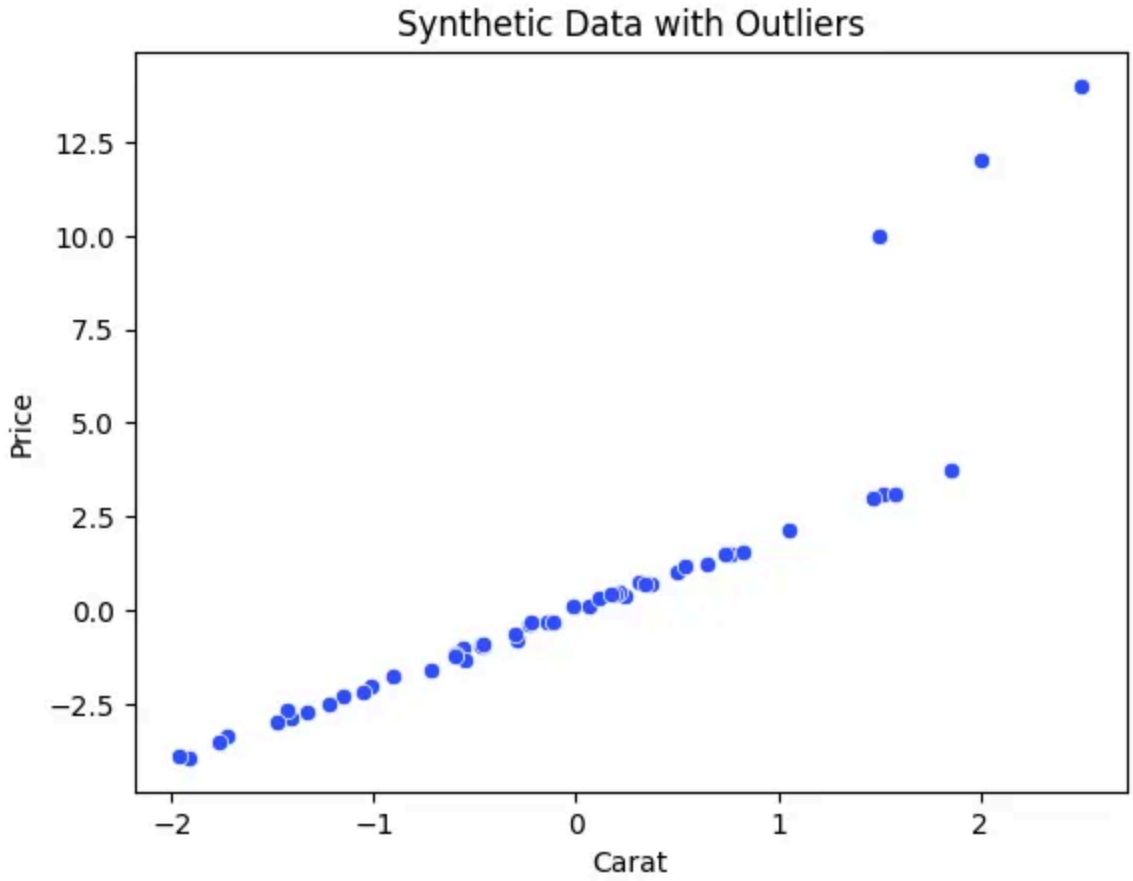
We'll generate a dataset with a linear relationship, introduce some outliers, and prepare it for the Huber Regression.

```
# Step 1: Generate synthetic data
np.random.seed(42)  # For reproducibility
X = np.random.normal(0, 1, 50).reshape(-1, 1)  # 50 samples, 1 feature
y = 2 * X.flatten() + np.random.normal(0, 0.1, 50)  # Linear relation with noise

# Add outliers
X_outliers = np.append(X, [[1.5], [2], [2.5]])
y_outliers = np.append(y, [10, 12, 14])

# Combine the data into a DataFrame for easier plotting
df_sample_train_10 = pd.DataFrame({'carat': X_outliers.flatten(), 'price': y_out

# Prepare the data for Huber Regression
X_train_with_intercept = np.hstack((np.ones((X_outliers.shape[0], 1)), X_outlier
y_train = y_outliers
```

Step 3: Visualising Data

We visualise our training data to see how `carat` relates to `price`.

```
sns.scatterplot(data=df_sample_train_10, x='carat', y='price', color='#304FFE')
```

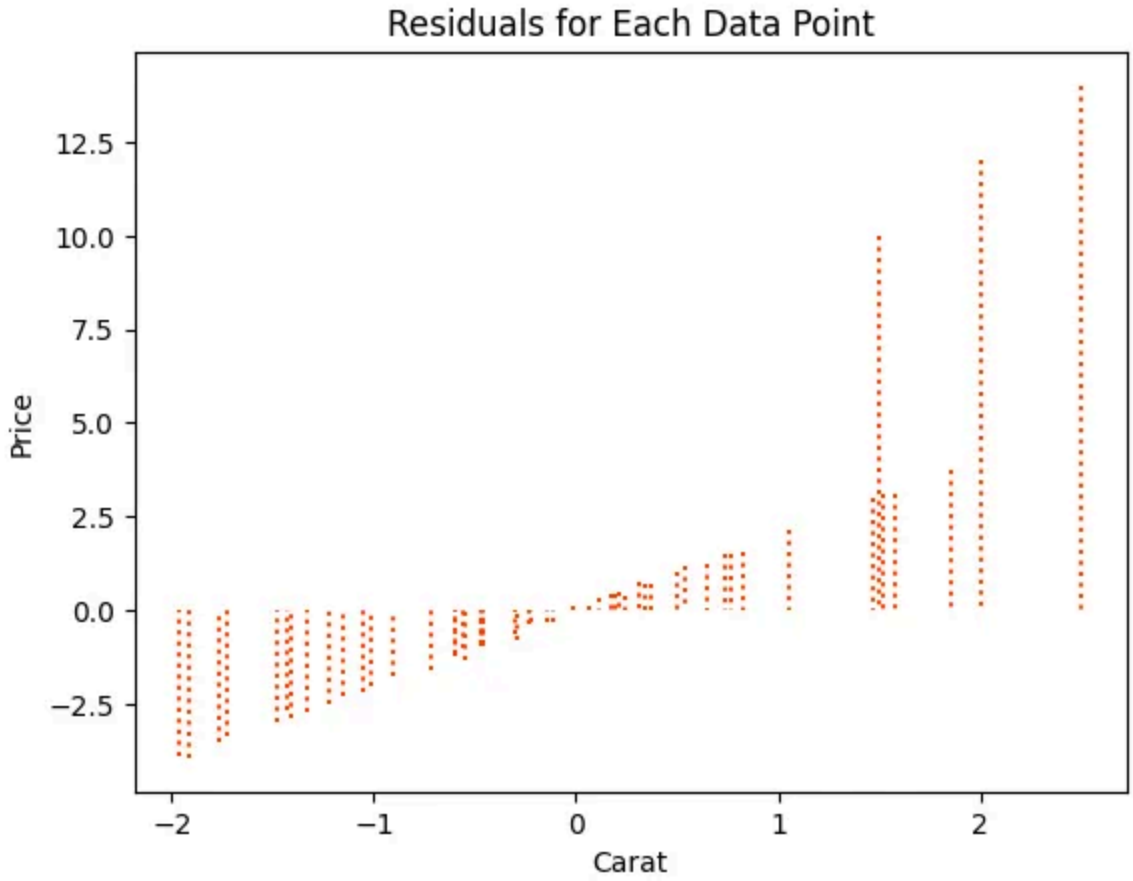


Step 4: Compute Predictions and Residuals

Next, we compute the predictions using the current coefficients and calculate the residuals:

```
predictions = X_train_with_intercept @ coef  # Predictions with intercept
residuals = y_train - predictions
```

Since the initialised coefficients were 0*x + 0, all predictions will be 0 for the first iteration and hence all residuals will equal their respective y values.

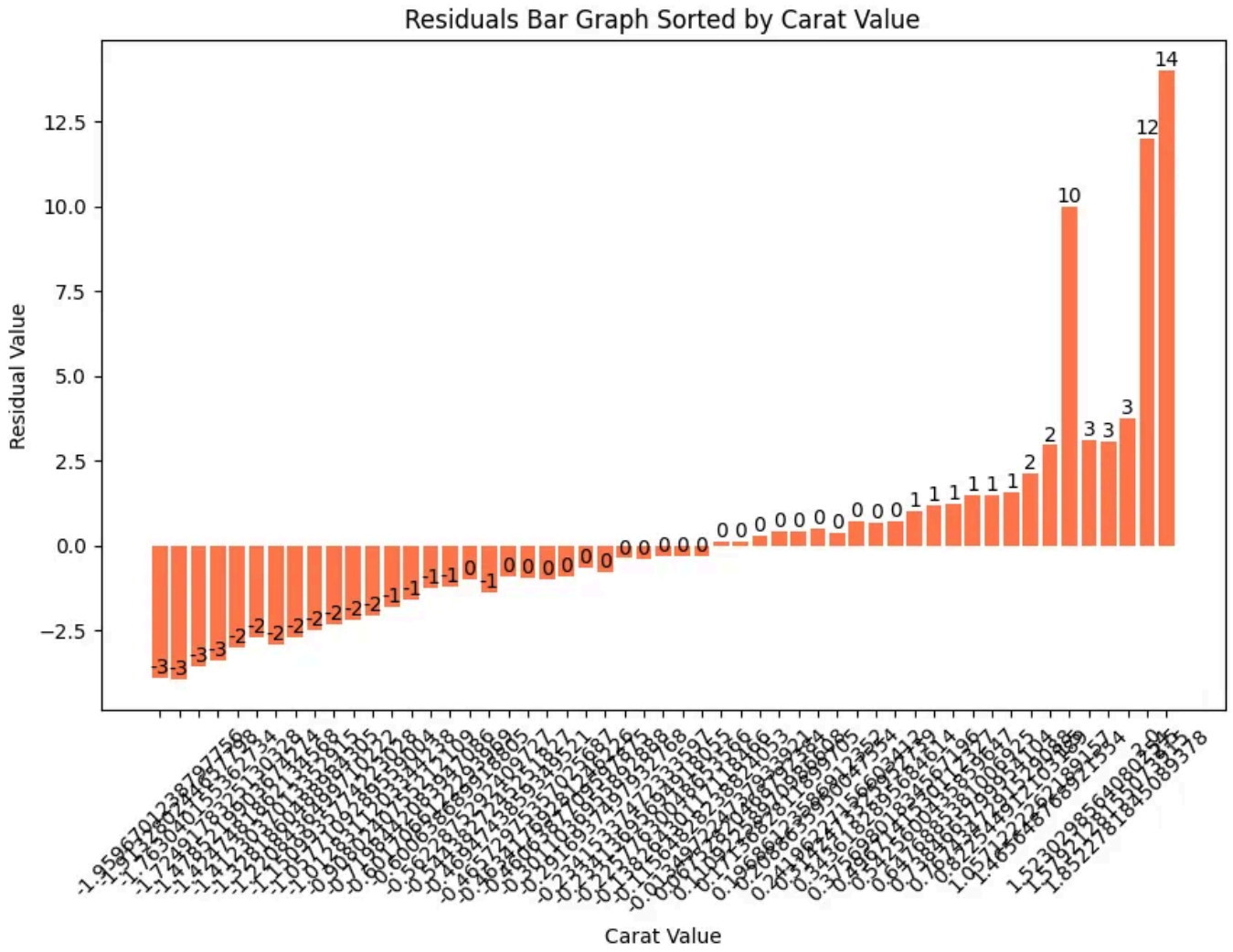


Step 5: Visualise Residuals

To understand the distribution of residuals, we can plot them against the `carat` values:

```
df_sample_train_10['residuals'] = residuals
df = pd.DataFrame(df_sample_train_10)

# Sort the DataFrame by carat value
df_sorted = df.sort_values(by='carat')

# Create a bar graph of the residuals
plt.figure(figsize=(10, 6))
bars = plt.bar(df_sorted['carat'].astype(str), df_sorted['residuals'], color='#F

# Annotate the bars with their values
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, f'{int(yval)}', ha='center',
```

```
plt.xlabel('Carat Value')
plt.ylabel('Residual Value')
plt.title('Residuals Bar Graph Sorted by Carat Value')
plt.xticks(rotation=45)
plt.show()
```



Step 6: Calculate Median Absolute Deviation (MAD)

Now, we compute the MAD, which helps us assess the spread of residuals.

Median Absolute Deviation (MAD): This method provides a robust measure of variability in the data. The MAD is defined as the median of the absolute

deviations from the median of the dataset. This approach is particularly useful for robust statistics because it is less sensitive to outliers than standard deviation.

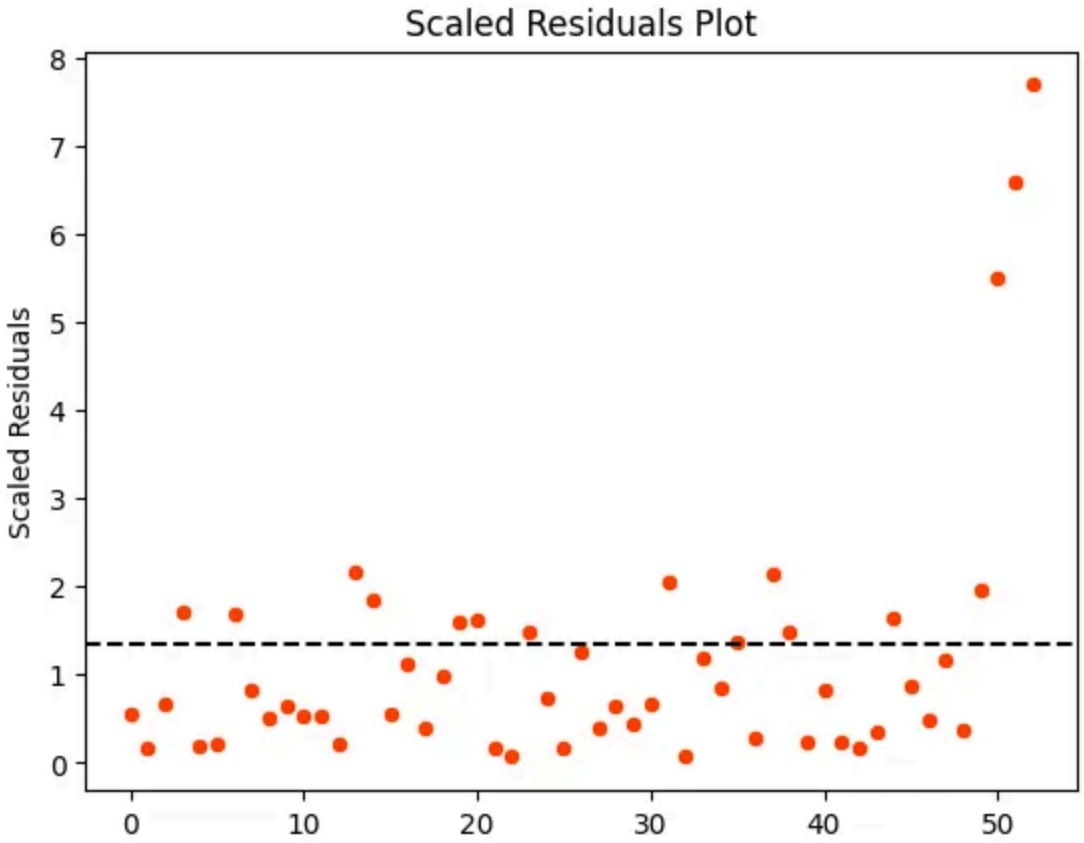
```
MAD = np.median(np.abs(residuals))
```

Median Absolute Deviation (MAD): 1.227684876170789

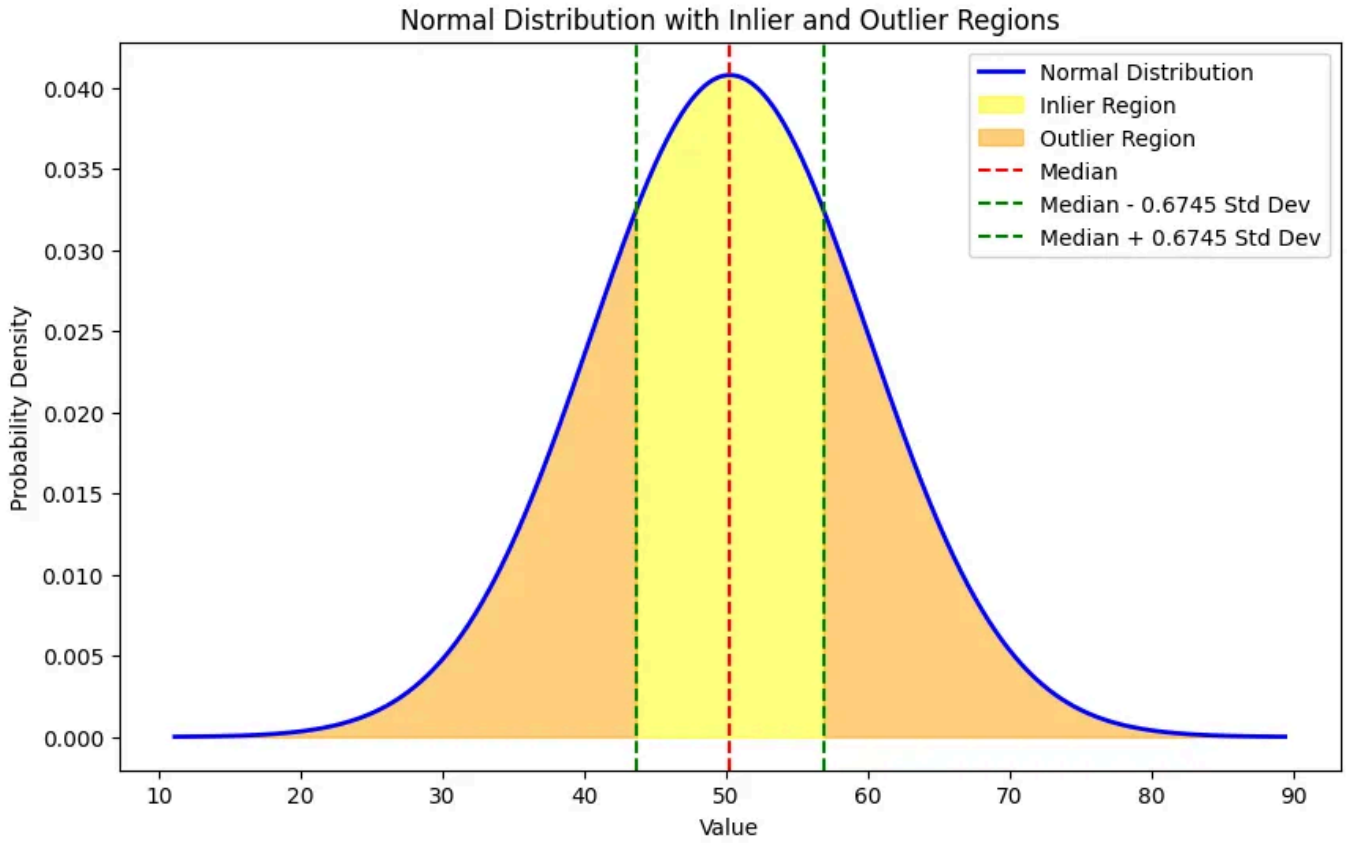
Step 7: Scale Residuals

Scaling the residuals helps to make them more uniform:

```
scale = MAD / 0.6745  # Scale
scaled_residuals = residuals / scale
```



The division by 0.6745 is a normalisation factor that adjusts the MAD to be consistent with the standard deviation for normally distributed data. Specifically, it relates the MAD to the standard deviation of a normal distribution, where approximately 50% of the data lies within 0.6745 standard deviations from the median.



Step 8: Compute Weights for Huber Loss

Next, we compute weights based on the scaled residuals:

```
weights = np.where(np.abs(scaled_residuals) <= epsilon, 1, epsilon / np.abs(scal
```

Weights: Residuals less than `epsilon` get a weight of 1 (treated normally), while larger residuals are down-weighted, reducing their impact on the regression.

Step 9: Weighted Least Squares Update

Now we apply the weights to the features for the regression update:

```
X_weighted = X_train_with_intercept * weights.reshape(-1, 1)  # Weighting the fe
```

This step modifies the feature set to reflect the weights derived from the residuals.

Step 10: Regularisation

We also add a regularisation term to avoid overfitting:

```
regularization_matrix = alpha * np.eye(n_features)  # Regularization matrix
regularization_matrix[0, 0] = 0  # Exclude intercept from regularization

# Coefficient update with regularization
coef_update = np.linalg.pinv(X_weighted.T @ X_weighted + regularization_matrix)
```

Step 11: Check for Convergence

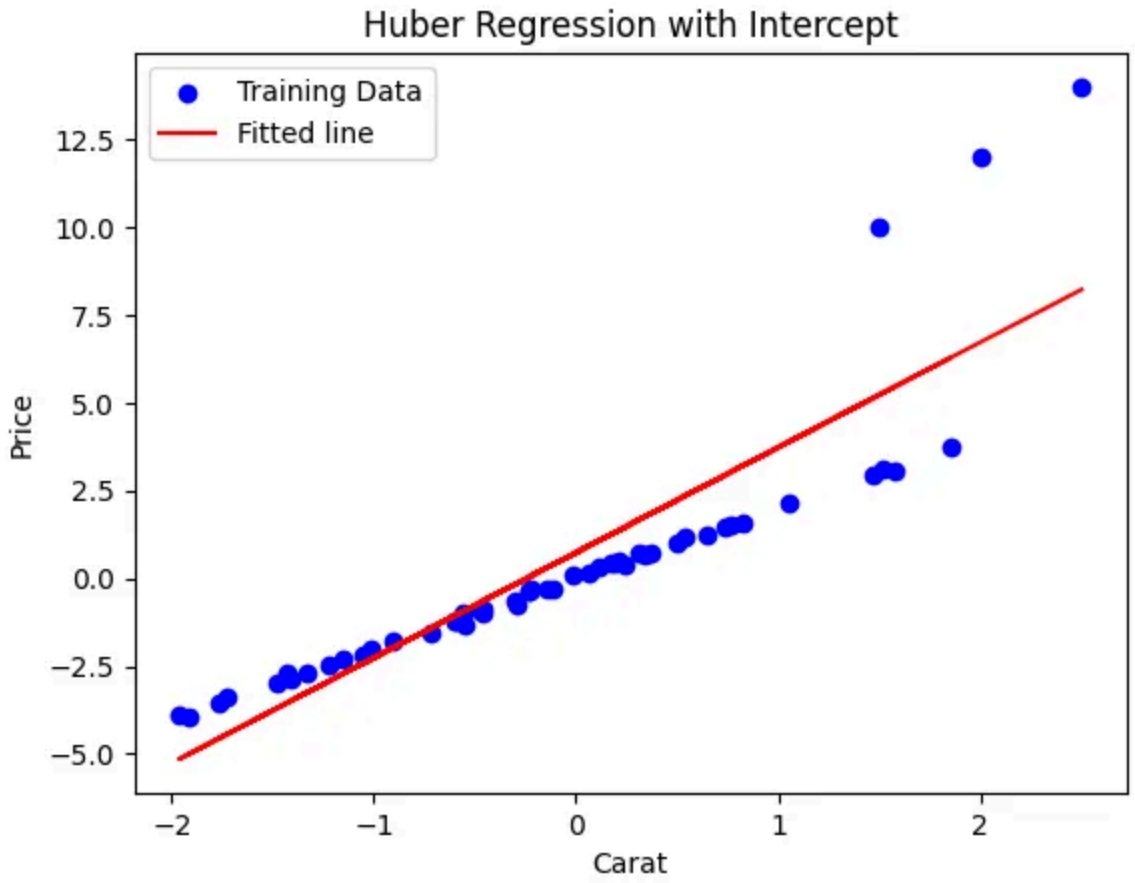
We check if the coefficients have converged:

```
if np.linalg.norm(coef - coef_update) < tol:
    print("break")

coef = coef_update
```

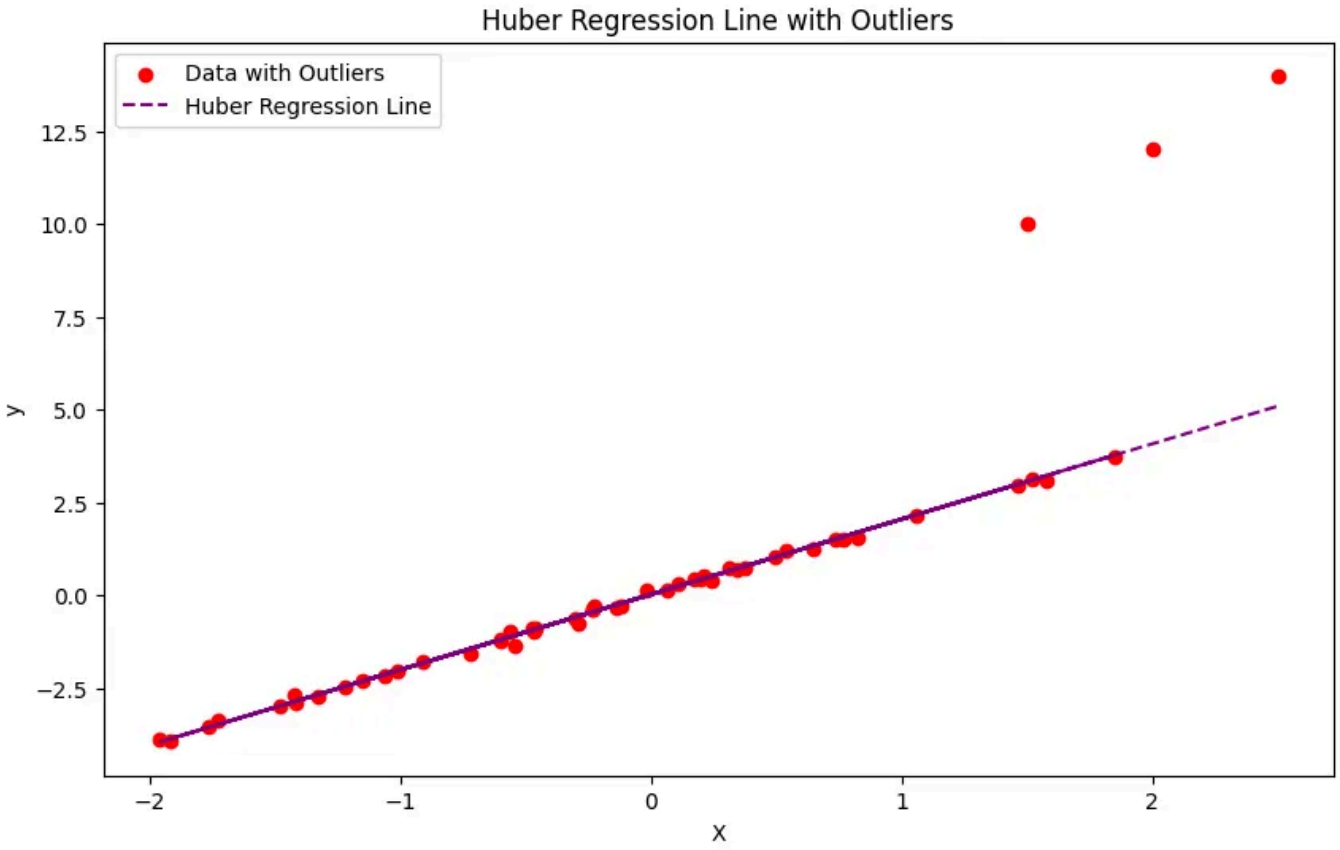
Step 12: Visualising Results

Finally, we visualise the results of the Huber regression:

```
plt.scatter(X_train, y_train, color='blue', label='Training Data')  # Training d
plt.plot(X_train, X_train @ coef[1:] + coef[0], color='red', linestyle='-')  # F
plt.title('Huber Regression with Intercept')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.show()
```



These results are only after 1 iteration. Huber implements multiple iterations on the dataset to reach to a convergence point thereby reducing the impact of outliers.



4. Where Huber Regression Shines: Real-World Applications 🌍

Huber Regression is invaluable in finance, environmental sciences, and any field where data can have unexpected spikes. It's especially useful in cases where we don't want to ignore outliers entirely but instead adjust our model's sensitivity to them.

Practical Example: Stock Prices

Imagine we're analyzing stock prices. Large fluctuations often occur due to market events, but we don't want these events to dominate our predictions. Huber Regression allows us to incorporate these "outlier" points while still capturing the overall trend.

5. Pitfalls and Misconceptions of Huber Regression 🚧

While Huber Regression is effective, it's essential to choose epsilon carefully. An epsilon that's too large will treat all points as inliers, resembling MSE, while too small an epsilon will make it similar to MAE. Additionally, Huber Regression might not perform well when outliers make up a large portion of the dataset, as the algorithm might lose the overall trend.

6. Conclusion 🌟

Huber Regression is a flexible and powerful tool, designed to be resilient against outliers while staying accurate for most data points. By understanding the math behind it and leveraging it properly, we can develop models that are robust, balanced, and better suited for complex datasets with both regular patterns and extreme values.

Give Huber Regression a try on your next dataset with outliers — let your model be sensitive to the overall trend, but don't let outliers dominate the story!

If you prefer visual understanding, check out my 8 mins video link on the **Huber Regressor!**

Outliers | Machine Learning | Linear Regression | Data Science

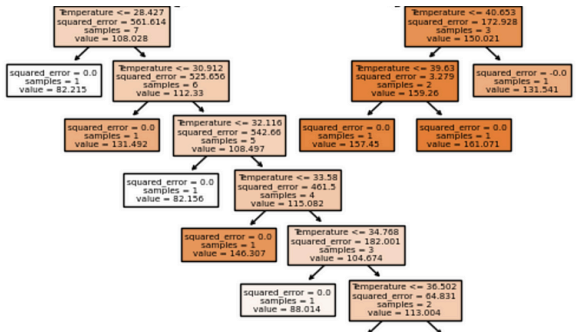
Written by AlgoStalk
2 followers · 0 following
Follow

No responses yet

Write a response

What are your thoughts?

More from AlgoStalk



AlgoStalk

Use SGD Regressor When Data is Constantly Updating: A...

If you prefer visual understanding, check out my video link on the SGD Regressor!

Oct 30, 2024

AlgoStalk

Decision Tree Regressor: The Unsung Foundation for Mastering...

If you prefer visual understanding, check out this our link on the Decision Tree Regressor!

Dec 1, 2024

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

ix of input features (size $n \times p$).

r of target values (size $n \times 1$).

or of estimated coefficients.

AlgoStalk

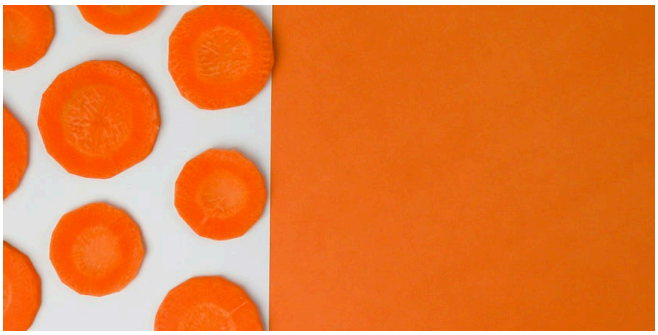
Mastering Linear Regression: How to Tackle Overfitting with Ridge,...

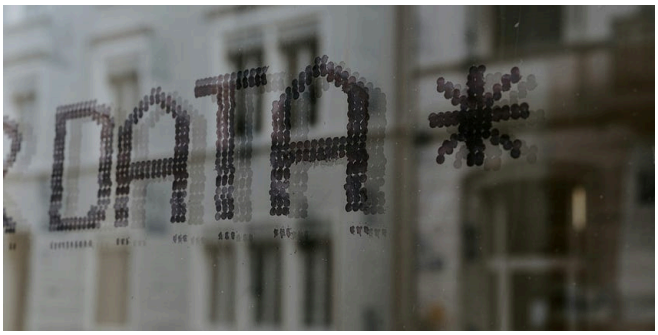
Mastering Linear Regression, Overfitting, and Regularisation: A Deep Dive

Oct 16, 2024

See all from AlgoStalk

Recommended from Medium





In Data Science Collective by Kuriko Iwai

The EM Algorithm and Gaussian Mixture Models for Advanced Dat...

A deep dive into the core concepts of unsupervised clustering with practical...

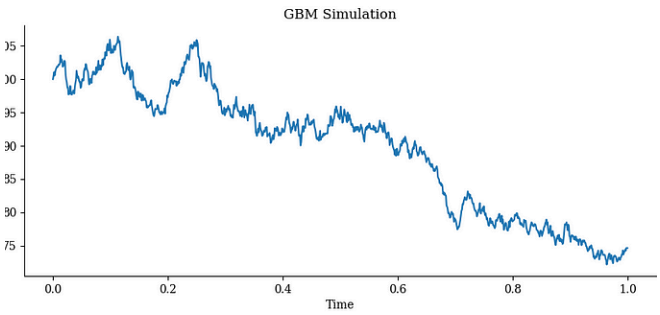
5d ago 140 2

Software Chasers

Understanding Normalization: Why It Matters for Regression and...

Normalization is typically performed for both regression and classification tasks when...

Dec 28, 2024 543 2



In Python in Plain English by Kyle Jones

Getting Started with Itô's Lemma for Stochastic Finance

From geometric Brownian motion to Ornstein-Uhlenbeck, the math behind financial...

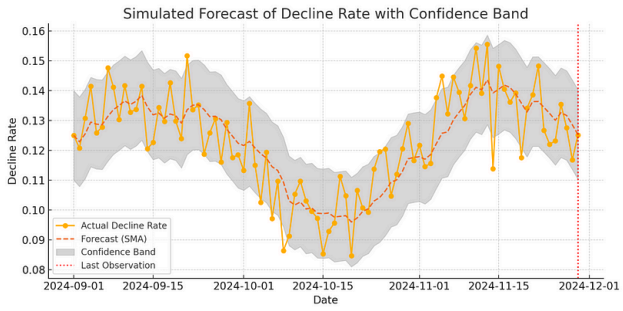
Jun 17 12 1

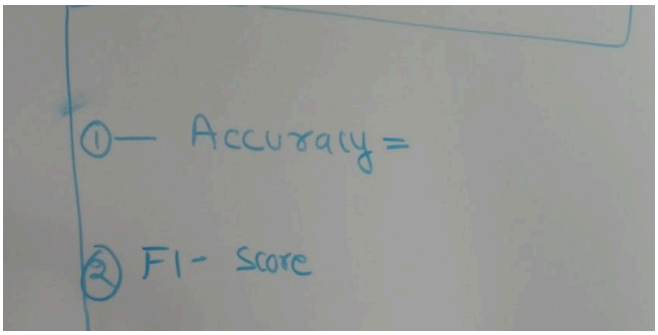
Sarmita Majumdar

XGBoost—Master Guide For Time Series In Python

A contemporary method for structured, multivariate time series forecasting.

Apr 10 44





Ruth Yang

Data Clarity #10: Smarter Alerts, Less Noise—Building a Batch...

How we used modular forecasting with Prophet to surface real issues behind fraud,...

Prem Vishnoi(cloudvala)

Machine Learning Evaluation Metrics: Accuracy, F1, ROC-AUC...

We are going talk few basic for ML KPI eg Accuracy , F1 Score , ROC-AUC

6d ago

May 27 109

See more recommendations