towards
data science

Regression in the face of messy outliers? Try Huber regressor

Outliers in the data are ubiquitous, and they can mess up your regression problem. Try Hubber regressor to tackle this problem.

Tirthajyoti Sarkar 2 days ago · 5 min read★

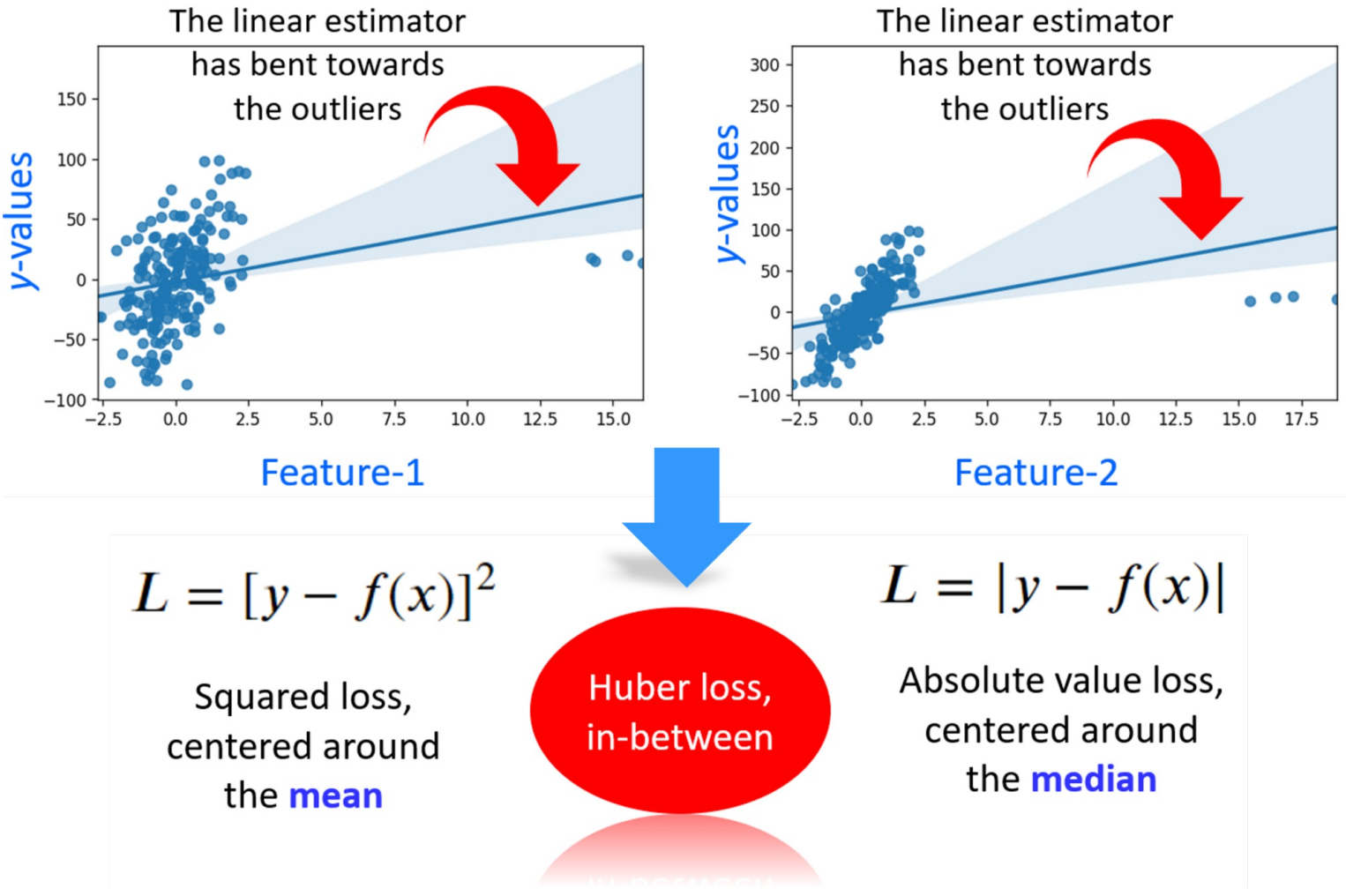


Image source: Created by the author

What's the problem?

Let's say you have a dataset with two features $X1$ and $X2$, on which you are performing linear regression. However, some noise/outliers got introduced in the dataset.

Say, **the *y*-value outliers are exceptionally low** as compared to what

they should be. How does that look like?

This is the data you got.

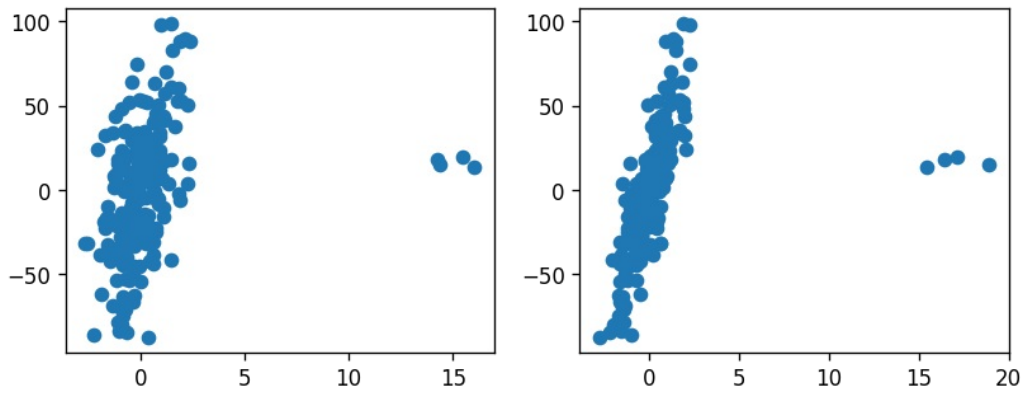


Image source: Created by the author

However, if you really think about the slope of the X-Y data, the expected y-values should have been much higher for those X-values. Something like the following,

Image source: Created by the author

These are obvious outliers and you can run a simple exploratory data analysis (EDA) to catch and discard them from the dataset before building the regression model.

But you cannot hope to catch all the outliers — ***at scale and in all dimensions***. Visualization of a dataset with 100 or 1000 dimensions (features) is challenging enough to manually examine the plots and discover outliers.

A regression algorithm that is *robust to outliers* sounds like a good bet

against those pesky bad data points. The Huber regressor is one such tool that we will discuss in this article.

Huber regression

In statistics, Huber loss is a particular loss function (first introduced in 1964 by **Peter Jost Huber**, a Swiss mathematician) that is used widely for robust regression problems — situations where outliers are present that can degrade the performance and accuracy of least-squared-loss error based regression.

Technical details

The loss is given by,

Image source: Wikipedia

We can see that the loss is the square of the usual residual ($y - f(x)$) only when the absolute value of the residual is smaller than a fixed parameter. The choice and tuning of this parameter are important to get a good estimator. Where the residual is greater than this parameter, the loss is a function of the absolute value of the residual and the Huber parameter.

Now, you may remember from elementary statistics that the **squared loss comes from the unbiased estimator around the mean whereas an absolute difference loss comes from an unbiased estimator around the median**. Median is much more robust to outliers than mean.

Huber loss is a balanced compromise between these two types. It is robust to the outliers but does not completely ignore them either. The tuning can be done with the free parameter, of course.

Image source: Created by the author

A Python demo

The demo notebook is here in my Github repo.

We created the synthetic data and added some noisy outlier with the following code,

```
import numpy as np
from sklearn.datasets import make_regression

rng = np.random.RandomState(0)
X, y, coef = make_regression(
    n_samples=200, n_features=2, noise=4.0, coef=True,
random_state=0)

# The first four data points are outlier
X[:4] = rng.uniform(10, 20, (4, 2))
y[:4] = rng.uniform(10, 20, 4)
```

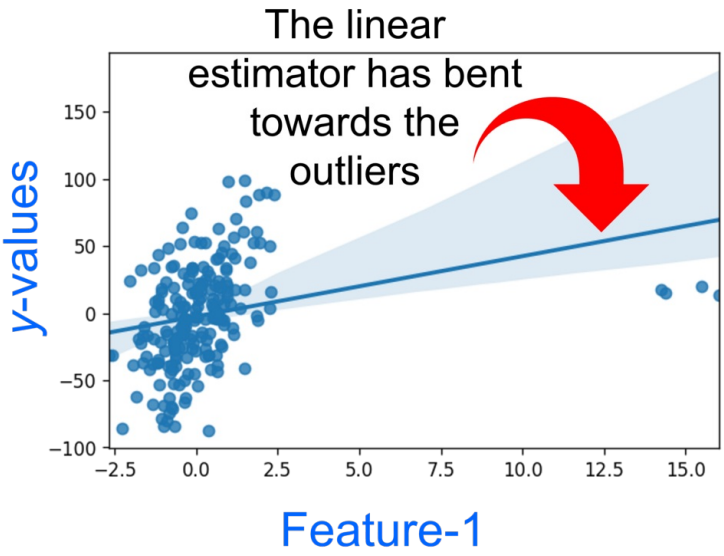
Now, all you have to do is to call Scikit-learn's built-in `HuberRegressor` estimator and fit the data. For comparison, we also have the standard `LinearRegression` method called in.

```
from sklearn.linear_model import HuberRegressor, LinearRegression

huber = HuberRegressor().fit(X, y)
linear = LinearRegression().fit(X, y)
```

Now, we know that the first 4 data points are outliers. So, if we try to predict the y-value with the first data point, we will get something like this,

```
linear.predict(X[:1,])

>> array([87.38004436])
```

As expected, the linear regression prediction is a low value for *y*. Why? Because the linear fit (based on least-squared loss) has bent towards the outliers due to their large leverage.



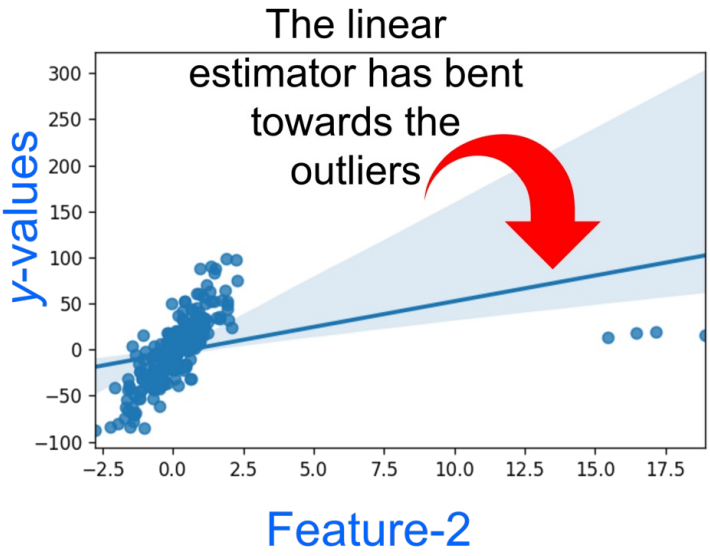


Image source: Created by the author

However, the Huber estimator predicts a more reasonable (high) value,

```
huber.predict(X[:1,])

>> array([806.72000092])
```

To demonstrate the robustness of the Huber estimator further, we can use the estimated coefficients and plot the best-fitted line,

```
huber_y1 = np.arange(-2.5,20,0.01)*huber.coef_[0] + \
            np.arange(-2.5,20,0.01)*huber.coef_[1] + \
            huber.intercept_

plt.figure(dpi=120)
plt.scatter(X[:,0],y)
plt.plot(np.arange(-2.5,20,0.01),
         huber_y1,
         color='red',linestyle='--')
plt.show()
```


Image source: Created by the author

Theil-Sen estimator

Although we are not discussing it in this article, readers are encouraged to check the Theil-Sen estimator, which is another robust linear regression technique and enjoys being highly insensitive to outliers.

As expected, Scikit-learn has a built-in method for this estimator too: **Scikit-learn Theil-Sen estimator**.

For multiple linear regression with a large number of features, this is a very efficient and fast regression algorithm among all the robust estimators.

Summary

In this brief article, we talked about the problem of linear regression estimators in the presence of outliers in the dataset. We demonstrated, with a simple example, that linear estimators based on traditional least-squared loss function, may predict completely wrong values because they are bent towards the outliers.

We discussed a couple of robust estimators and demonstrated the Huber regressor in detail. Non-parametric statistics use these robust regression techniques in many places, especially when the data is expected to be particularly noisy.

Data science students and professionals alike should also have a working knowledge of these robust regression methods for automating the modeling of large datasets in the presence of outliers.

. . .

*Loved the article? Become a **Medium member** to continue **learning without limits**. I'll receive a portion of your membership fee if you use the following link, **with no extra cost to you**.*

Data Science | Machine Learning | Python | Programming