Transformations of Variables in Regression Models

Why Linear Relationships Are Sometimes Not Enough

Any model will make specific assumptions about the relationship between predictors and the target. A linear regression model makes a pretty giant assumption, that the relationship is linear! Real-world data often violate these assumptions. We will tackle this issue in a few different ways. The first is by using transformations.

- Transforming the Target: To make the data more “normal” or to decrease the effect of long tails on model fit.
- Transforming the Predictors: To capture non-linear relationships.
- Both can make our models more accurate and interpretable.

Regression with a Log Transformation

Standard Linear Regression:

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon$$

With Log Transformation on Target Variable:

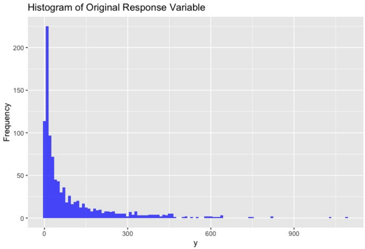
$$\log(y) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon$$

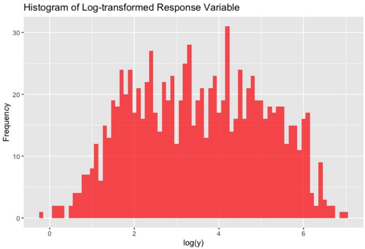
```python
import numpy as np
from sklearn.linear_model import LinearRegression

# Log transformation of target variable
y_log = np.log(y)

# Fit the model
model = LinearRegression().fit(X, y_log)
```

Original vs. Log-Transformed Target Variable





- Left: Original scale shows right-skewed distribution.
- Right: Log transformation achieves a more symmetric and often more "normal" distribution, facilitating linear regression modeling.

Log-Transformed Target Variable

Even for non-linear models like regression trees, taking the log of the target variable can be helpful. Remember regression trees minimizes the sum of squared errors in a bin. A long right tail would essentially force a lot of model effort into making those sum of squared errors small.

Introduction to Residuals and Fitted Values

We can also detect a need for transformations using residual diagnostics.

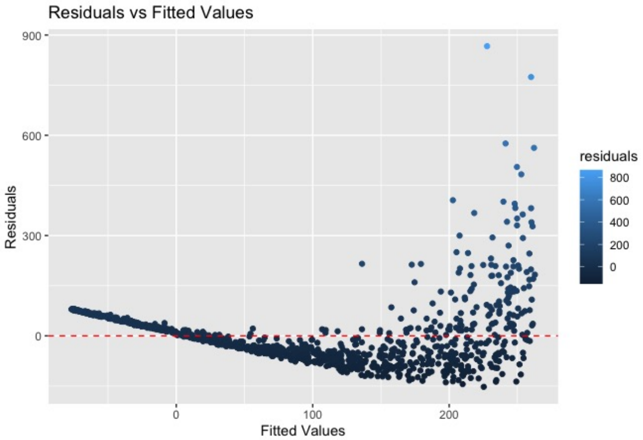
- **Residual**: The difference between the observed value of the dependent variable (y) and the predicted value ($\hat{y}$).

$$\text{Residual} = y - \hat{y}$$

- **Fitted Value**: The predicted value of the response variable (y) for a particular value of the predictor variable (x).

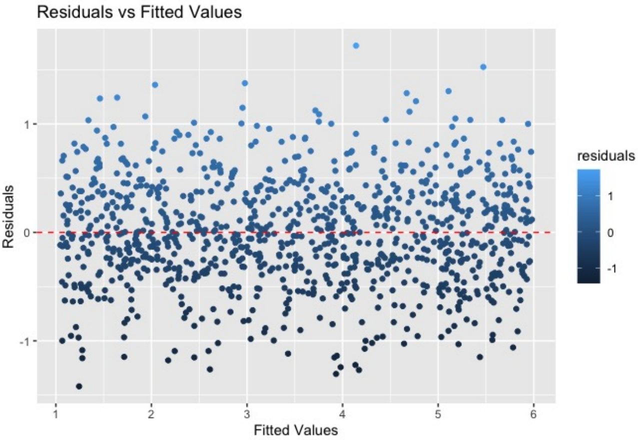
$$\text{Fitted Value} = \hat{y}$$

Residuals vs Fitted Values



- Patterns in this plot can suggest issues with linearity.
- If you see this thin ono the left and spread out towards the right, you should consider a log transformation

Residuals vs Fitted Values



- A blob is a good thing

Predictions and Residuals vs. Fitted Values

```
# Predictions
y_pred = model.predict(X)

# Residuals
residuals = y - y_pred

# Plotting
import matplotlib.pyplot as plt
plt.scatter(y_pred, residuals)
plt.axhline(y=0, color=‘r’,linestyle=‘--’)
plt.xlabel(‘Fitted values’)
plt.ylabel(‘Residuals’)
plt.title(‘Residuals vs. Fitted Values’)
plt.show()
```

Diagnosing the Need for Log Transformation

One last way to tell if you need a log transformation to try with and without and check model performance.

1. Split the data into train and test group
2. Fit model 1 without the log transformation on the training set. Get the predictions on the test set.
3. Build model 2 using $y' = log(y)$ as the target variable on the training set. Get predictions on the test set.
4. For model 2 predictions, transform back to the original scale, $\hat{y} = e^{(\hat{y'})}$
5. Compare predictions using something like MSE or R^2.

It is very important that you back transform predictions, otherwise the two models will not be comparable.

Implementation of Log Transformation Diagnostic

```python
# 1. Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
# 2. Fit model without log transformation
model1 = LinearRegression().fit(X_train, y_train)
y_pred1 = model1.predict(X_test)
# 3. Fit model with log transformation
y_train_log = np.log(y_train)
model2 = LinearRegression().fit(X_train, y_train_log)
y_pred_log = model2.predict(X_test)
# 4. Back transform predictions
y_pred2 = np.exp(y_pred_log)
# 5. Compare predictions using MSE
mse1 = mean_squared_error(y_test, y_pred1)
mse2 = mean_squared_error(y_test, y_pred2)
```

Transformations of Predictors

While transforming target variables is common, predictor variables can also benefit from transformations, especially when a predictor's relationship with the outcome is non-linear.

- **Log Transformation of a Predictor**: Can linearize relationships or reduce the effect of extreme values.
- While taking the log of a target is something to consider for all models, taking the log of predictors is not as useful for regression trees. In fact most transformations on predictors are ways to make linear regression a little less linear.

Implementing Log Transformation of a Predictor

```
# Log transformation of a predictor variable
X_log = np.log(X[’predictor_column’])

# Replace original predictor with the transformed one
X[’predictor_column’] = X_log
```

Transformations of Predictors

How can you tell if you need to do a log transformation of a predictor?

- Plot the histogram of the variable. A long right tail indicates you may want too use a log
- Check the scatterplot of the predictor vs the target. If you see most of the data smashed to one side instead of towards the middle, you may want to log transform one or both of the variables
- Check p-values of the variable with and without the log transformation. If the original scale variable is not significant but the log transform is, you want the log transform.
- Check out of sample performance in either case

Example Scenario: Housing Prices

Consider a dataset with the following attributes:

- Target variable: House Price (in thousands of dollars)
- Predictor variable: Square Footage of the house

We aim to investigate if a log-transformation of the Square Footage predictor improves the model.

Square Footage	House Price
1000	200
1200	220
1400	250
1600	275

Table: Sample data points

Methodology

To evaluate the need for a log transformation, we will:

- Fit two models: one with original Square Footage and another with log-transformed Square Footage.
- Assess their performance based on p-values and out-of-sample metrics like R^2 and MSE.

Model	p-value	R^2
Original	0.08	0.75
Log-Transformed	0.01	0.80

Table: Model performance metrics

Discussion and Conclusion

Based on the results:

- The *p*-value for the original model is 0.08 (> 0.05), suggesting it's not very significant.
- The *p*-value for the log-transformed model is 0.01 (< 0.05), indicating significance.
- The R^2 value also improved from 0.75 to 0.80.

Conclusion: A log-transformation of the Square Footage predictor is beneficial.

Introduction: Why Standardize and Scale?

Standardizing and scaling variables is a preprocessing step that's crucial for many machine learning algorithms. This is important because:

- Different feature scales can slow down the learning process.
- Coefficients of features can be misinterpreted.

The Importance of Feature Scaling

Problems due to different scales:

- Slower convergence of optimization algorithms.
- Misleading interpretation of feature importance.

Example: Suppose you're predicting house prices using features like square footage (range: 500-5000) and number of bedrooms (range: 1-5). The scales are vastly different, and this affects the model adversely.

Mathematical Background

Formula for Standardization:

$$z = \frac{x - \mu}{\sigma}$$

Where μ is the mean and σ is the standard deviation of the feature. This formula transforms your feature to have zero mean and unit variance, making it easier for algorithms to interpret them equally.

Implementing Standardization in Python

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import
    train_test_split

# Split data into training and test set
X_train, X_test, y_train, y_test =
    train_test_split(X, y)

# Create and fit scaler object on training set
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)

# Transform the test set
X_test_scaled = scaler.transform(X_test)
```

The scaler object is fit only on the training set to prevent data leakage from the test set.

Standardizing Target Variable

```python
# Create and fit a new scaler object for the target variable
scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1, 1))

# Transform the test target variable
y_test_scaled = scaler_y.transform(y_test.reshape(-1, 1))
```

It's not always necessary to standardize the target variable, but it can help in regression problems for the same reasons as feature scaling.

Original Regression Formula

The original regression model is:

$$y = \beta_0 + \beta_1 \times X_1 + \beta_2 \times X_2$$

where y is the dependent variable, and X_1, X_2 are predictor variables.

Transformation Process

To standardize, each variable X is transformed as:

$$Z_X = \frac{X - \mu_X}{\sigma_X}$$

And the dependent variable y as:

$$Z_y = \frac{y - \mu_y}{\sigma_y}$$

Transformed Regression Formula

The transformed regression model becomes:

$$Z_y = \beta_0' + \beta_1' \times Z_{X_1} + \beta_2' \times Z_{X_2}$$

These coefficients will be different than the coefficients if you did not back standardize.

Impact on Coefficients

Original:

$$y = 100 + 0.1 \times \text{Square Footage} + 20 \times \text{House Age}$$

The coefficients β_1, β_2 indicate the expected change in y per one unit change in X_1, X_2.
Transformed:

$$Z_y = 0.1 + 0.3 \times Z_{\text{Square Footage}} + 1.2 \times Z_{\text{House Age}}$$

The coefficients β_1', β_2' now indicate the change in Z_y per one unit change in Z_{X_1}, Z_{X_2}.

Model Inference

The R^2 and p-values remain unchanged after standardization. This ensures that model inference remains consistent before and after transformation.

Back Transformation

To revert the standardized prediction to the original scale:

$$y = \sigma_y \times Z_y + \mu_y$$

Fitting Regression Model on Scaled Data

```python
from sklearn.linear_model import LinearRegression

# Initialize and fit the model
model = LinearRegression()
model.fit(X_train_scaled, y_train_scaled)

# Make predictions on scaled test data
y_test_scaled_pred = model.predict(X_test_scaled)
```

We fit the model on the scaled training data and make predictions on the scaled test data.

Back-Transforming Predictions

```python
# Back-transform the scaled predictions to original scale
y_test_pred = scaler_y.inverse_transform(y_test_scaled_pred)

# Now y_test_pred can be compared with y_test
```

The back-transformed predictions are in the original scale and can be compared to the actual test values.

Detailing the Prediction Process

- Fit the regression model on scaled training data.
- Make predictions on scaled test data.
- Back-transform these predictions to the original scale.
- Compare the back-transformed predictions with the actual test values.

This method ensures that the predictive power of the model is properly evaluated without data leakage from the test set.

Why Polynomial Regression?

Linear regression assumes a linear relationship between predictors and the target variable. However, real-world data often violate this assumption. Polynomial regression can help capture these non-linear patterns.

- Standard Linear Regression: $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p + \epsilon$
- Polynomial Regression: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$

Polynomial Regression Formula

Mathematical Formula:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \ldots + \beta_n x^n + \epsilon$$

- β_0 is the intercept.
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients.
- $x, x^2, \ldots, x^n$ are the predictors.

Implementing Polynomial Regression

```python
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Create polynomial features
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X)

# Fit the model
model = LinearRegression().fit(X_poly, y)
```

Real-World Example: Sales Data

Consider a dataset that shows the relationship between advertising expenditure (in thousands) and sales revenue (in millions):

Advertising Expenditure	Sales Revenue
10	20
20	25
30	40
40	37
50	44

Interpreting Coefficients

Let's say the polynomial regression model for the sales data results in the equation:

$$\text{Sales} = 10 + 2.5 \times \text{Ad Spend} - 0.1 \times \text{Ad Spend}^2$$

- The coefficient of Ad Spend is 2.5, indicating for each \$1000 increase in ad spend, sales increase by \$2.5 million, initially.
- The coefficient of Ad Spend^2 is -0.1, which suggests the rate of increase in sales slows down as ad expenditure increases.

Strategy for Using Polynomial Variables

1. Start with a linear model.
2. Examine residuals and plots to determine if a polynomial model is appropriate.
3. Choose the degree of the polynomial carefully. Higher degrees can lead to overfitting.
4. Compare models using out-of-sample metrics like R^2 and MSE.