Final Project Report

December 6, 2019

MCSC 6020G
Fall 2019
Derick Smith

1 Regression Assumption Testing

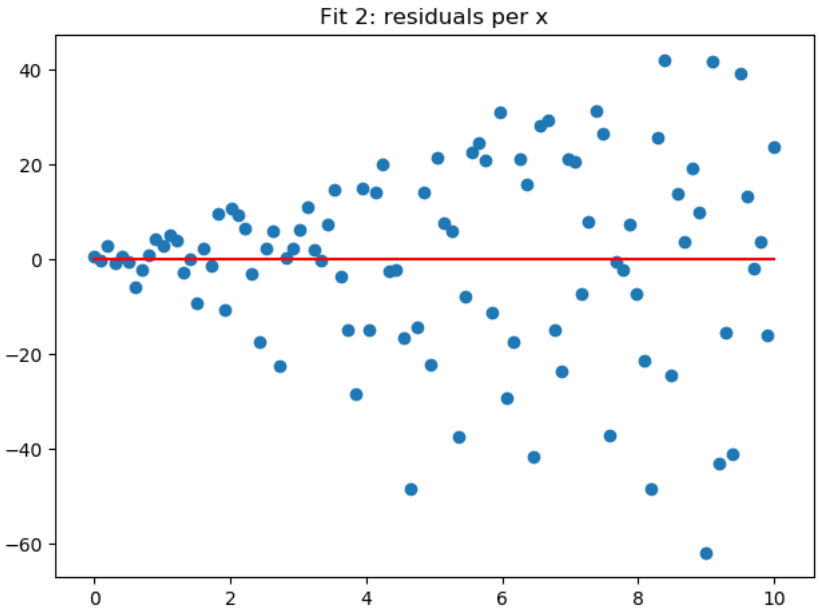
1.1 Project Explanation

The purpose of this project was to investigate the means in which we test the assumptions we make when performing linear regression. The focus of the investigation was on the assumptions made about residual errors in the construction of polynomial fits as it tends to be the most interpretable and easiest to explain. At an undergraduate level, testing assumptions is typically an introduction to the topic. Throughout the course of research, the methods of testing can be less than rigorous in some cases.

For instance, in the testing of the independence of residual error in a fit based on a set of data with noise, or irreducible error, is subjective at its core. The definition of structurelessness of a predictor to residual plot cannot be to stringent or else regression in real world application would not be feasible. If there were a fool-proof way to detect it with a usable degree of certainty, it would be taught and an industry standard. Machine learning might be a way to accomplish this. Humans have incredible vision systems but the ability to eye-ball what is structureless and what is not has its limitations. One day, one might consider there is structure and another day, might not. A machine could observe billions of regression examples and their effectiveness relative to the structure of residuals, perhaps even using mean cluster size with deviation. If there are any important applied mathematical fields using subjective testing, it would be drastically improved by the use of a machine. This is the case for testing the independence of residual error and also for the other assumptions.

Testing for constant variance falls into this same category subjectivity. The test can even use the same predictor to residual plot, as mentioned above. The key difference being, if there is constant variance, the plot points will be evenly distributed throught the plot space. Again, this is not a watered down explanation. Of course, there is a little more to it but the mathematical algorithms are not being skipped over because they don't exist by industry standards.

Figure 1: From file <Non-normalResiduals.py>. Demonstrating non-constant variance and non-normality of residuals.



Thankfully, testing for normality is a slightly better story. Yes, there are tests that will provide a rankable measure that can be compared to other predictor to residual plots from any regression fit from any set of data. Example being the Shapiro-Wilks test. It is efficient at confirming that a distribution is normal but it cannot tell if a distribution is not normal. A subtle difference. Although the true irreducible error may be normal, as its distribution begins to deviate from normal - due to noise, skew, small sample size, or some hidden variables - it becomes indistiguishable from other types of distributions such as uniform or Poisson. This is due to the random nature of error. Although Shapiro-Wilks is used (as it should be), the Quantile-Quantile (Q-Q) plot is heavily relied on to determine normality as it demonstrates a direct comparison to what a normal distribution of errors would appear to be and how much the residuals have deviated.

Figure 2: From file <Non-normalResiduals.py>. Demonstrating the difficulty of determining normality from a frequency plot of residuals.

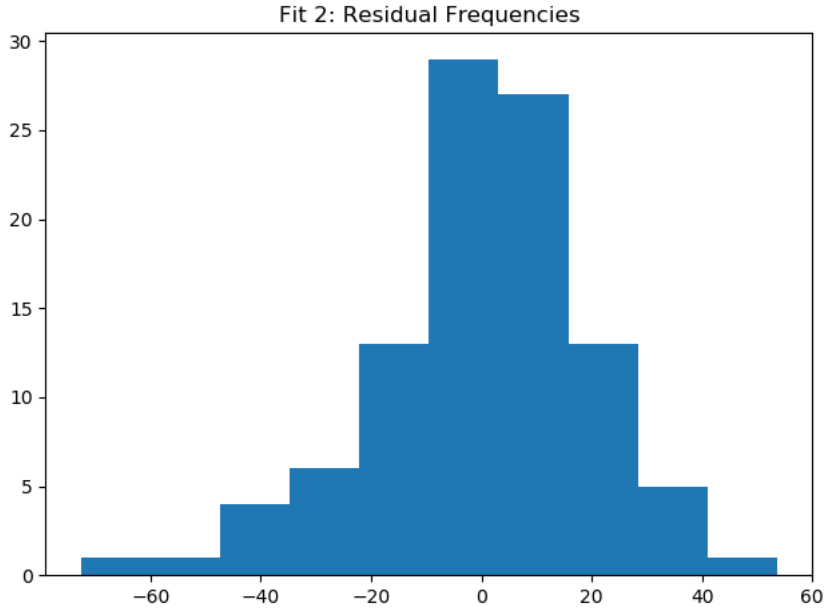
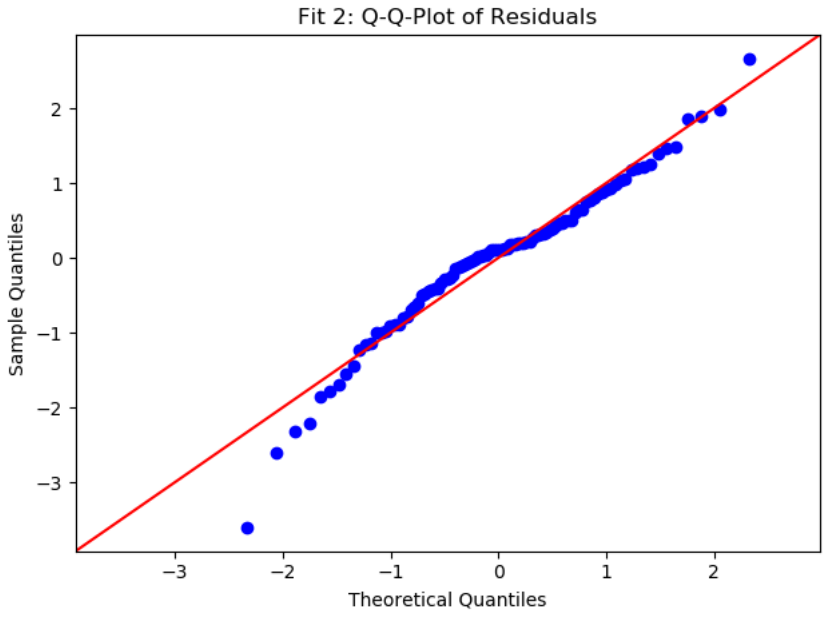


Figure 3: From file <Non-normalResiduals.py>. Demonstrating the power of the Quantile-Quantile plot over the frequency of residuals plot.



There are more methods of testing assumptions but these are fairly important and commonly used. All of which would benefit from machine learning

to handle the subjective analysis as it can build upon a larger foundation of examples. Larger meaning it's not even close. A machine could theoretically learn from every single regression analysis that has ever been completed and make decisions based on that meta-anaylsis. This is not to make it sound the current state of statistics gloomy. Regression and regression testing (t-tests, z-test, F-tests, and so on) are robust. This means that even when assumptions are slightly violated, they retain inferential and predictive power but does not mean we should not optimize the ways of confirming regression assumptions.

The following is a list of references that were used:

- An Introduction to Mathematical Statistics and Its Applications, Fifth Edition, R. Larsen and M. Marx
  - Chapters three to six covers the premise of statistics and regression analysis.
- A Second Course in Statistics: Regression Analysis, Seventh Edition, W. Mendenhall and T. Sincich
  - Personal favourite. The notation is beautiful.
  - Chapter eight covers the testing of error assumptions.
- Design and Analysis of Experiments, Ninth Edition, D. Montgomery
  - Section 3.4 covers the testing of error assumptions.

1.2 Code Documentation

1. <NormalResiduals.py>
   (a) yHatFunc(x, betas)
      i. This function calculates the estimated y values (aka $\hat{y}$) based on the set of predictors values, x, using their corresponding coefficients, β, from the regression model.
      ii. Returns the array $\hat{y}$.
   (b) lineFunc(x, betas)
      i. This function creates the set of data needed to construct a line representing the regression model fit to be super impose onto the data.
      ii. Returns (x, y) tuple array of the regression fit.
   (c) # Import data #
      i. Self-explanatory. Imports csv file and to create a panda dataframe.
   (d) # Var Selection #
      i. This section simply simulates a watered down way of selecting predictors from a set of data.

ii. It creates scatter plots the response (dependent) variable, y, with each predictor (independent) variable, x.

(e) # Select x7 #

i. This simulates that through some method of variable selection that x_7 best describes the behaviour of y.

ii. Any other variable can be selected to see how it impacts the testing that follows.

(f) # Degree zero polyfit #

i. Uses Numpy's function called polyfit() to find the least squared fit at degree zero. This is unnecessary at degree zero since the fit is just the mean of the response.

ii. # Polyfit

A. Plots the data with a super-imposed line of the fit.

iii. # Plot residuals

A. Plots the residual values on the vertical axis and their corresponding predictor values on the horizontal axis to observe if the residuals are structureless and have constant variance.

iv. # Histogram Residuals

A. This is to observe the frequency distibution of residuals and also demonstrate the difficulty of assessing whether the residuals are normal from this plot.

v. # QQ Plot Residuals

A. This is to contrast the histogram with powerfulness of the Q-Q plot in determining how normal the residuals appear to be.

B. The residuals are standardized before calling the statsmodels.api package function qqplot() since it is much easier to interpret the plot with both vertical axis (experimental quantiles) and horizontal axis (theoretical quantiles) sharing the same scale.

(g) # Degree one polyfit #

i. This section is identical to # Degree zero polyfit #, except the degree of fit used in polyfit() have been incremented by one along with the python code variable names.

(h) # Degree two polyfit #

i. This section is identical to # Degree one polyfit #, except the degree of fit used in polyfit() have been incremented by one along with the python code variable names.

2. <Non-normalResiduals.py>

(a) The purpose of this file is to simulate a case when assumptions have been violated and how it impacts the testing of assumptions.

(b) This file is identical to <NormalResiduals.py> except for # Create Data # has replaced # Import Data # and a new function called yFunc().

(c) yFunc()

i. The purpose of this function is to simulate data that has an unknown interaction term that causes the response variance to increase as the predictor value increases.

ii. Every simulated value of the response has noise added. The noise is normally distributed with a mean of zero and deviation of one.

iii. This random normal distribution of noise may cause some response values to be negative. To ensure non-negative response values in the data, the Numpy function, abs(), computes the absolute value of each response array element.

iv. Returns simulated response array.

(d) # Create Data #

i. This purpose of this section is to simulate data with one response variable, one known predictor, and one hidden predictor. It is constructed in a way to violate the normality and constant variance assumption.

ii. Two predictor variable arrays are created, x_0 and x_1 with the same domain. Their domain space is split evenly into intervals. The array x_1 is shuffled to ensure a level of independence from x_0.

iii. yFunc(x_0, x_1) is called to populate the array y, the simulated response variable.

3. <Non-NormalResiduals_Transformation.py>

(a) The purpose of this file is to simulate the way in which a transformation might be used on the response variable data in order to reel in a non-constant variance or non-normal residual distribution.

(b) This file is identical to <Non-NormalResiduals.py> except it includes a new section # Transform y #.

(c) # Transformation y #

i. The purpose of this section is to show how the scatter plot of different transformed response variables and the predictor changes.

ii. In this simulation, the square root of the response is chosen to perform the proceeding model building and assumption testing, however, other transformations can be selected to see how it changes the results.