Crash Course on Regression Analysis

Concept 1: Linear Regression Basics

Linear regression is a statistical algorithm we use to model the relationship between a dependent variable *y* and one or more independent variables *x*

Singular Linear Regression

For a single independent variable, the model is:

$$y = \beta_0 + \beta_1 x_1 + \epsilon$$

Where:

- β_0 = the intercept of the regression line.
- β_1 = the slope of the regression line.
- ϵ = the error term capturing deviations from the model.

Multiple Linear Regression

For multiple independent variables, the model generalizes to:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

Here, $x_1, x_2, \ldots, x_n$ are the predictors, and $\beta_1, \beta_2, \ldots, \beta_n$ are their corresponding coefficients.

Concept 2: Estimating Coefficients in Linear Regression

Slope and Intercept

The Linear Regression Algorithm estimates coefficients using the **least squares method**, minimizing the sum of squared errors (ϵ^2 + ϵ^2 + ϵ^2) between real & predicted values. Since we only have a

data set & not a formula, we cannot get the real value of the slope β_1 - but we can obtain the estimated value $\hat{\beta}_1$ (pronounced beta hat 1)

Key Calculations

1. **Sum of Deviations (S_x):**

$$S_x = \sum(x_i - \bar{x})$$

   - $\bar{x}$ being the mean of x.
   - & x_i being the value of the i'th index of x (remember x is a set of the x values of our dataset & not a continuous domain)
2. **Sum of Deviations Squared (S_{xx}):**

$$S_{xx} = \sum(x_i - \bar{x})^2$$

3. **Estimated Slope ($\hat{\beta}_1$):**

$$\hat{\beta}_1 = \frac{S_{xy}}{S_{xx}}$$

   (where $S_{xy} = \sum(x_i - \bar{x})(y_i - \bar{y})$)
4. **Estimated Intercept ($\hat{\beta}_0$):**

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1\bar{x}$$

5. **Overall ($\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 * x_1$):**

$$\hat{y}_i = b + mx_i = \hat{\beta}_0 + \hat{\beta}_1 * x_i = \bar{y} - \hat{\beta}_1\bar{x} + \hat{\beta}_1 * x_i$$

$$= \bar{y} - \frac{S_{xy}}{S_{xx}}\bar{x} + \frac{S_{xy}}{S_{xx}} * x_i$$

$$= \bar{y} - \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}\bar{x} + \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2} * x_i$$

$$= \hat{y}_i = \bar{y} + \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}(x_i - \bar{x})$$

   & this should be all you need to implement your very own linear regression

also keep in mind: this also works for multiple linear regression - if the above evaluates to $y = m_1x_1 + b$, then the following can be used for $y = m_1x_1 + m_2x_2 + b$ (consider a table of [y, x1, x2] as oppose to [y, x])

- **Multiple Regression Formula:**

$$\hat{y}_i = \bar{y} + \frac{\sum(x_{1i} - \bar{x}_1)(y_i - \bar{y})}{\sum(x_{1i} - \bar{x}_1)^2}(x_{1i} - \bar{x}_1) + \frac{\sum(x_{2i} - \bar{x}_2)(y_i - \bar{y})}{\sum(x_{2i} - \bar{x}_2)^2}(x_{2i} - \bar{x}_2)$$

Concept 3: Non Linear Functions

Linear regression works for linear relationships, but what if we find a non-linear pattern along our adventures? For these cases, we use polynomial regression - a generalized form of linear regression. This involves fitting a curve, such as a quadratic (degree 2) or cubic (degree 3) equation, to the data.

Quadratic Regression

Quadratic regression models data as:

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

The process involves adding a new feature x^2 (the square of the independent variable) and performing linear regression on the extended set of features.

Code Walkthrough for Quadratic Regression:

```
def quadratic_regression(x, y):
        # add constant (intercept) and quadratic term (x^2) to the feature set
    X = sm.add_constant(np.column_stack((x, np.power(x, 2))))

        # fit an ordinary least squares (OLS) (linear) model
    model = sm.OLS(y, X).fit()

        # extract coefficients for intercept, linear, and quadratic terms
    intercept, linear, quadratic = model.params

        # calculate mean absolute error between actual and predicted values
    error = np.mean(np.abs(y - model.predict(X)))

        # return the error n formula
    return error, intercept + linear * age + quadratic * age**2
```

Process:

1. **X parameter engineering**: We extend x to include both the original values and their squares (x^2).
2. **Model Fitting**: The `sm.OLS` function performs Ordinary Least Squares regression using x and x^2 as features.
3. **Prediction**: Predictions are made using the equation:

$$\hat{y} = \beta_0 + \beta_1 x + \beta_2 x^2$$

In other words: Map $[y, x]$ onto $[y, x_1, x_2]$ where $x_1 = x$ & $x_2 = x_1^2$

Cubic Regression

Cubic regression extends this approach to include a third cubic term (x^3):

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$$

Code Walkthrough for Cubic Regression:

```
def cubic_regression(x, y):
        # add quadratic term (x^2), and cubic term (x^3) to the feature set
    X = sm.add_constant(np.column_stack((x, np.power(x, 2), np.power(x, 3))))

        # fit lin model
    model = sm.OLS(y, X).fit()

        # extract coefficients for intercept, linear, quadratic, **and cubic** terms
    intercept, linear, quadratic, cubic = model.params

        # calculate mean absolute error between actual and predicted values
    error = np.mean(np.abs(y - model.predict(X)))

        # return error and the cubic prediction formula
    return error, intercept + linear * age + quadratic * age**2 + cubic * age**3
```

Explanation: x is extended to include x^2 and x^3 as features -- then we regress on all the parameters

& More regression:

Just a lightning round over what Plot_Finder.py is doing:

```
def poly_regression(x, y, degree):
    poly_terms = [np.power(x, i) for i in range(1, degree + 1)]
    X = sm.add_constant(np.column_stack(poly_terms))
    model = sm.OLS(y, X).fit()
        ...

def exp_regression(x, y):
    log_y = np.log(y)
    X = sm.add_constant(x)
    model = sm.OLS(log_y, X).fit()
        ...

def logarithmic_regression(x, y):
    X = sm.add_constant(np.log(x))
    model = sm.OLS(y, X).fit()
        ...

def sin_regression(x, y):
    sin_x = np.sin(x)
    X = sm.add_constant(sin_x)
    model = sm.OLS(y, X).fit()
        ...
```

Key Takeaways:

- Polynomial regression uses higher-degree terms ($x^2, x^3, \ldots$) to capture non-linear relationships.
- Quadratic regression adds one new feature (x^2), while cubic regression adds two (x^2 and x^3).
- Both methods rely on the same principles as linear regression, with the added complexity of managing more predictors.

This framework is flexible and can be extended to higher degrees, though increasing the polynomial degree too much risks over-fitting.

Concept 4: The Error Term (ϵ)

Error Expectations on Linear functions

In a typical linear regression on a linear function, the error term ϵ is assumed to satisfy 2 conditions:

1. **Expected Value**: The expected value of ϵ is zero ($E[\epsilon] = 0$).
   -- This is because $E[\epsilon] = mean(\epsilon) = 0$
2. **Normality**: The error term is normally distributied ($\epsilon \sim N(0, \sigma^2)$).

Error from Regression on Different Functions

For $y = x$:

- The error ϵ has $E[\epsilon] = 0$ since the model perfectly fits $y = x$ (thus the mean evaluates to 0)

However for $y = x + \sin(x)$:

- The expected value of the error term ϵ is $E[\epsilon] = \sin(x)$, reflecting the sinusoidal component not captured by the linear model.
- How do we reduce the error further? *By applying a regression on the error term*

This is the idea for Plot_Finder.py

- Step 1: Apply a regression
  - This should return $\hat{y}_1 = \hat{\beta}_0 + \hat{\beta}_1 * x_i$ (assuming linear gives the least error)
- Step 2: calculate the error term
  - $\epsilon = y_{real} - \hat{y}_{(estimated)}$
- Step 3: Regress the error term & add that to our function
  - This should return $\hat{y}_2 = \hat{\beta}_0 + \hat{\beta}_1 * x_i + \hat{\beta}_2 * sin(x_i)$ (assuming sinosudial error)
- Finally: Repeat steps 2-3 until a near 0 error or enough terms are calculated