Linear Regression

- Linear Regression
- Theory of linear Regression
- Simple Implementation with python
- Scikit-learn overview
- Linear Regression with Scikit-learn
- polynomial Regression
- Regularization
- overview of project datasets.

History and Motivation behind Linear Regression.

- Brief History, linear Relation ship
- Ordinary least squares, cost function
- Gradient decent - Vectorization

The history invention
↓
Navigation methods (1700s)
Age of exploration-

- 1722 — Roger Cotes - combining different observation yields better estimates of true value-

- 1750 - Tobias Mayer explore averaging different result under similar condition In studying librations of Moon

1757 – Roger Joseph [illegible]

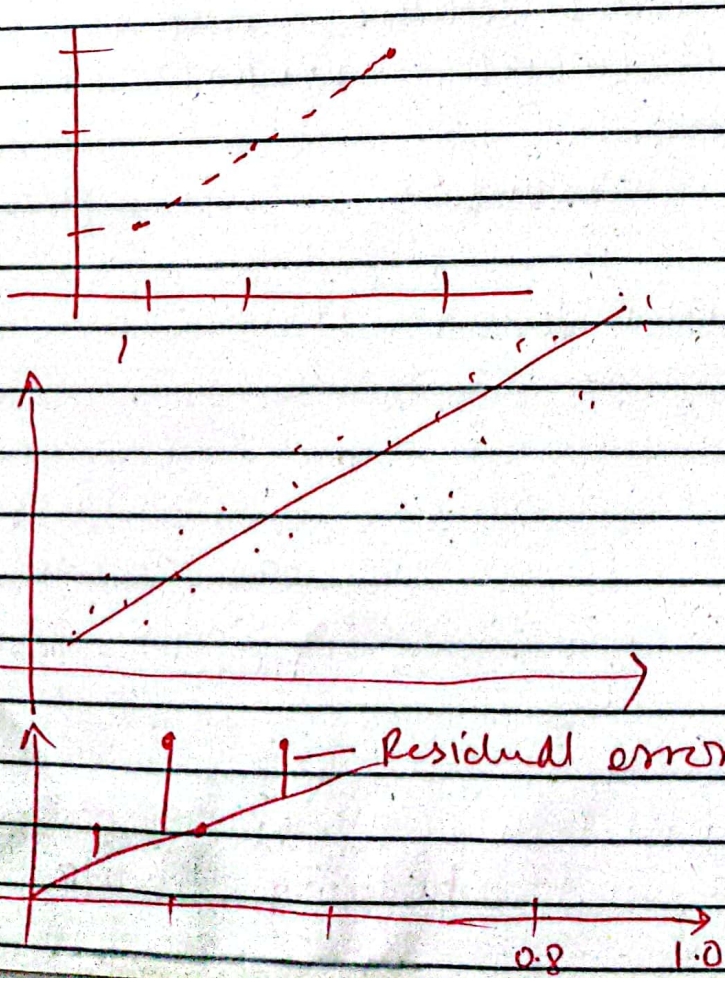
1788 – Pierre-Simon Laplace Laplace

→ 1805 – First public exposition on Linear Regression with least square method – published by Andrien-Marie Legendre –

• 1809 – Carl Friedrich Gauss publish methods of calculating orbits of celestial bodies

– claimed to have invented least-square back in 1795! –

born in 1977



- Ordinary least squares works by minimizing the sum of the squares of the difference between the observed dependent (variable (value of the observed) in the given dataset and those predicted by the linear function.
- Minimize the squared error

Equation :-

$y = mx + b \rightarrow$ incerpt

↓

slope

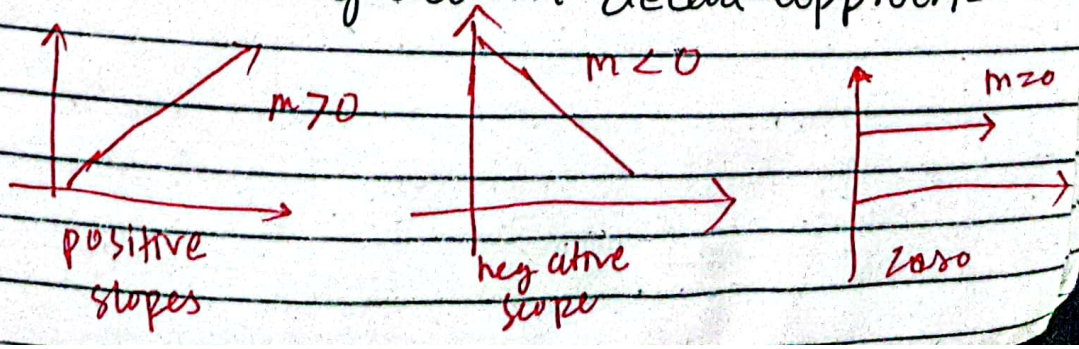
X_1	X_2	X_3	y
X_1^1	X_1^2		
X_1^2	X_1^3		
X_1^3	X_1^4		
X_1^4	X		

$\hat{y} = \beta_0 x_0 + \cdots + \beta_n x_n$

$$\hat{y} = \sum_{i=0}^{n} \beta_i x_i$$

$\hat{y}$ is estimation (cant fit)

- For single feature we can use analytical approch and for multiple features we can use gradient decent approch.



$y = b_1 x$

How to derive b_1 and b_0:

$b_1 = \rho_{x,y} \frac{\sigma_y}{\sigma_x}$

$\rho_{x,y}$ = pearson correlation coefficient

σ_x, σ_y = standard deviation

$$= \frac{\Sigma(x-\bar{x})(y-\bar{y})}{\sqrt{\Sigma(x-\bar{x})^2}\sqrt{\Sigma(y-\bar{y})^2}} \cdot \frac{\sqrt{\frac{\Sigma(y-\bar{y})^2}{n}}}{\sqrt{\frac{\Sigma(x-\bar{x})^2}{n}}}$$

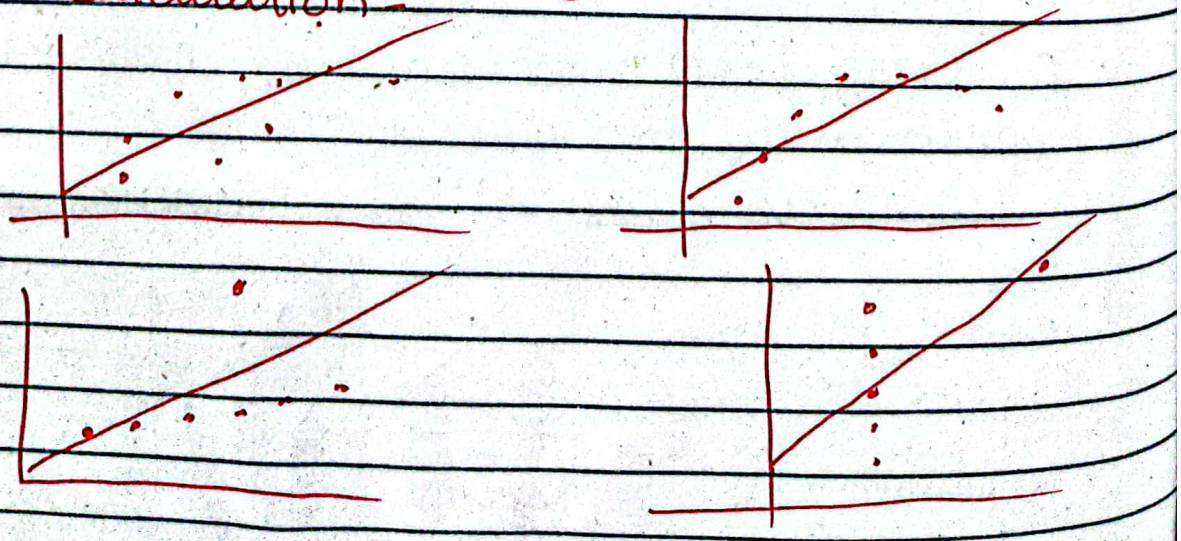
$$= \frac{\Sigma(x_i-\bar{x})(y_i-\bar{y})}{\Sigma(x_i-\bar{x})^2}$$

$$\boxed{b_1 = \frac{\Sigma(x_i-\bar{x})(y_i-\bar{y})}{\Sigma(x_i-\bar{x})^2}}$$

$$\boxed{b_0 = \bar{y} - b_1\bar{x}}$$

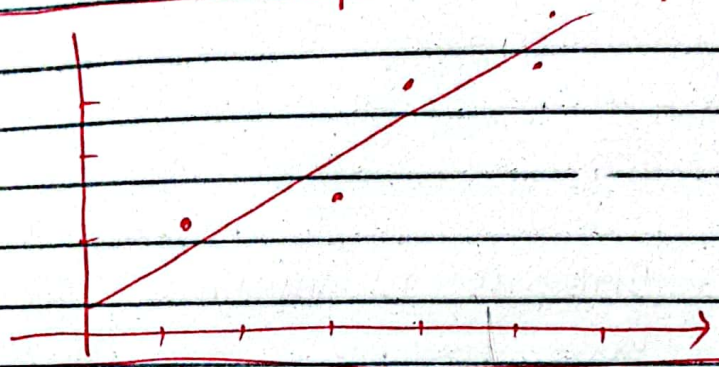
Limitation of linear Regression

Anscombes Quartet illustrate the pitfalls of relying on pure calculation.



Find the plot data

production Hours	production volume
34	102
35	109
39	137
42	148



$(x-\bar{x})$	$(y-\bar{y})$	$(x-\bar{x})(y-\bar{y})$	$(x-\bar{x})^2$
7	24	168	49

$$b_1 = \frac{\sum(x_i-\bar{x})(y-\bar{y})}{\sum(x_i-\bar{x})^2} = \frac{558}{128} = 4.5$$

$$y = -46 + 4.5x$$

For multiple features of independent variable it will be hard to use Ordinary least square methods.

- We will use gradient descent approch to minimize the ~~errors~~ cost function..

$$\hat{y} = \sum_{i=0}^{n} \beta_i x_i$$

Best fit line is to minimize the square error.

Residual Error

$$= \frac{1}{m} \sum_{j=1}^{m} (y^j - \hat{y}^j)^2$$ Average squared error.

A cost function is defined by some measure of error.

$$J(\beta) = \frac{1}{(2)m} \sum_{j=1}^{m} (y^j - \hat{y}^j)^2$$

↓ just for convient 2 is gonna cancel 2 - when taking derivate

$$J(\beta) = \frac{1}{2m} \sum_{j=1}^{m} \left(y^j - \sum_{i=0}^{n} \beta_i x_i^j \right)^2$$

From calculus to minimize a function you take derivate and set it equal to zero.

$$\frac{\partial J}{\partial \beta_k}(\beta) = \frac{\partial}{\partial \beta_k}\left(\frac{1}{2m}\sum_{j=1}^{m}\left(y^j - \sum_{i=0}^{n}\beta_i x_i^j\right)^2\right)$$

$$\frac{\partial J(\beta)}{\partial \beta_k} = \frac{1}{m}\sum_{j=1}^{m}\left(y^j - \sum_{i=0}^{n}\beta_i x_i^j\right)(-x_k^j)$$

We will describe this cost function as vectorize Matrix Notation and use gradient descent to have a computer figure out the set of Beta coefficient values that minimize the cost/Loss function.

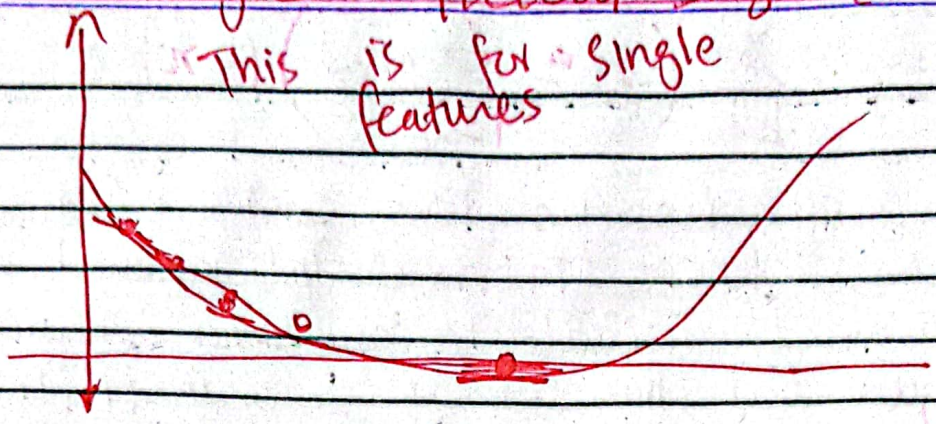
$$\nabla_\beta J = \begin{bmatrix} \frac{\partial J}{\partial \beta_0} \\ \vdots \\ \frac{\partial J}{\partial \beta_n} \end{bmatrix}$$

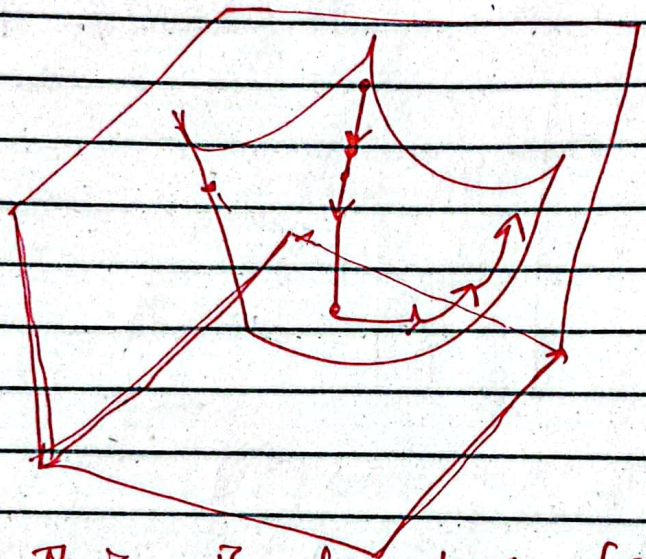
$$X = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \cdots & x_n^1 \\ 1 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_1^m & x_2^m & \cdots & x_n^m \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \beta_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

$$\nabla_\beta J = -\frac{1}{m}\begin{bmatrix} \sum_{j=1}^{m} y^j x_0^j \\ \vdots \\ \sum_{j=1}^{m} y^j x_n^j \end{bmatrix} + \frac{1}{M}\begin{bmatrix} \sum_{j=1}^{m}\sum_{i=0}^{n}\beta_i x_i^j x_0^j \\ \vdots \\ \sum_{j=1}^{m}\sum_{i=0}^{n}\beta_i x_i^j x_n^j \end{bmatrix}$$

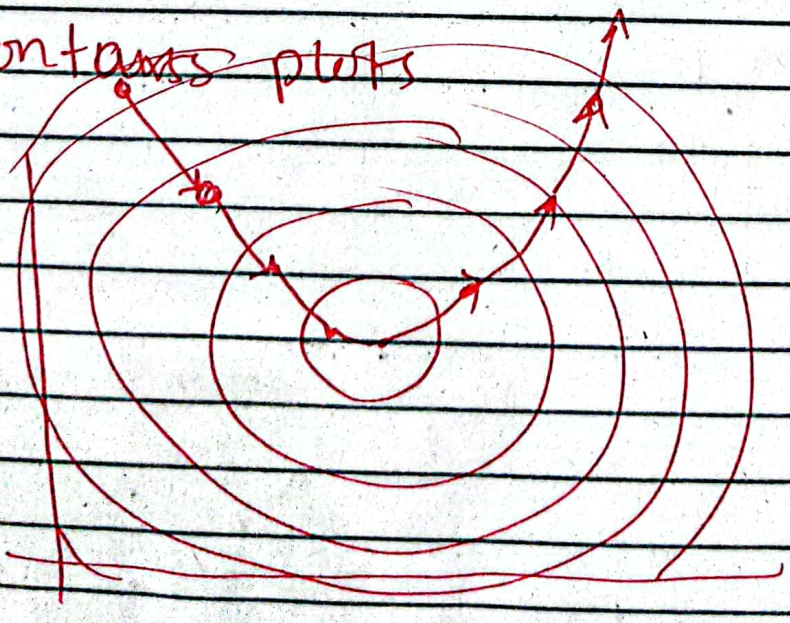
Single Coefficient Single (Beta)





This is for two features and visualizing in Movie then two features is not possible.

Contours plots



Python Coding Simple linear Regression :-

Chapter # 3 ISLR

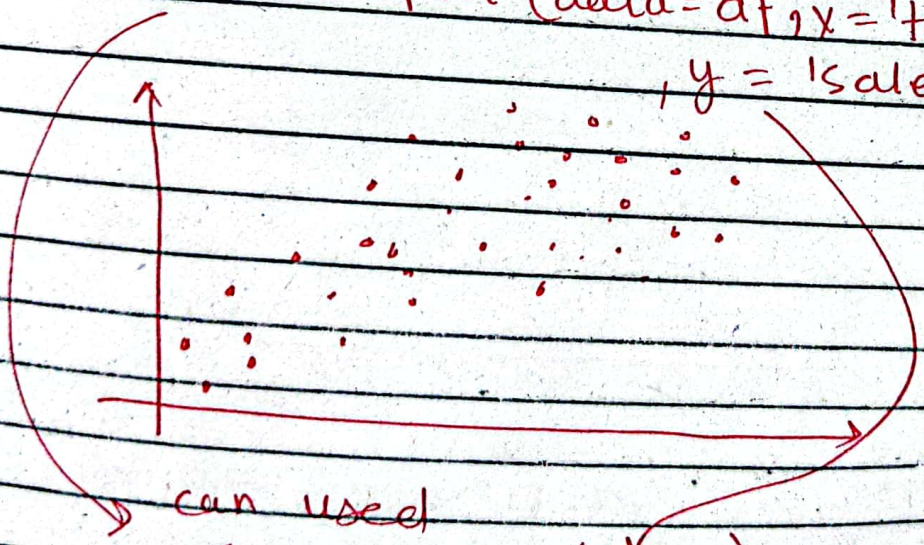
$y = mx + b$

```
import numpy as np
import matplotlib.pyplot as pd
import pandas as pd
import seaborn as sns

df.head(); 5 rows view

df['total spend'] = df['TV']+df['radio']
                    + df['sales']
df.head();

sns.scatterplot(data=df, x='total-spend'
                , y='sales')
```



can used
sns.regplot(same)

```
X = df['total_spend']
y = df['sales']
```

help(np.polyfit) — help to read documentation

y = mX + b

y = B1 X + B0

np.polyfit(X, y, degree = 1)

array([0.0486878 , 4.24302833])

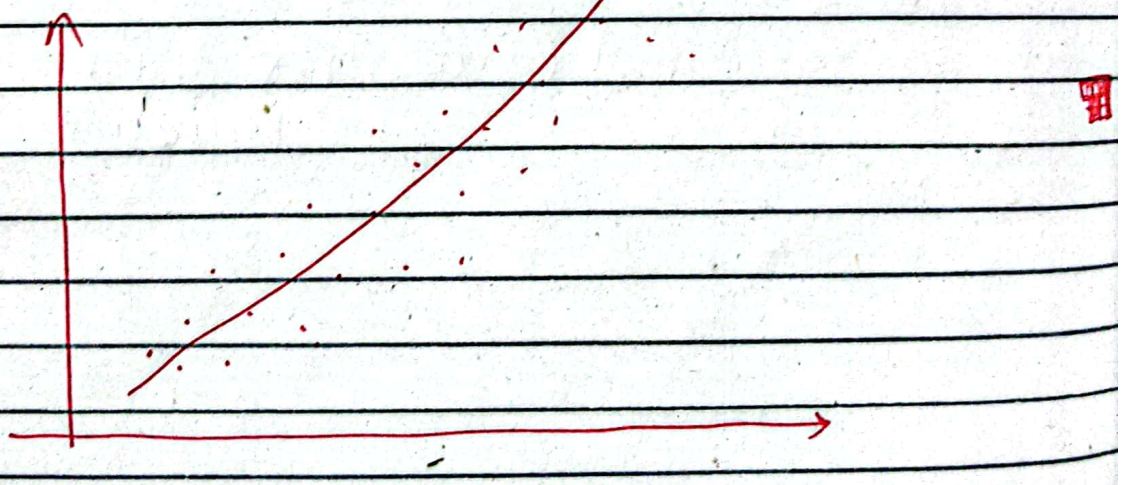
Between 0 to 500 points give me 100 points

potential_spent = np.linspace(0, 500, 100)

predicted_sales = 0.0486878 * potential spent + 4.2430283

plt.plot(potential_spend, predicted_sales)

sns.scatterplot(X = to same as above)



Spend = 200

predicted_sales = 0.0486878 * spend + 4.418

Best.

np.polyfit(x, y, 3)

array([3.1×10^-, -1.8m, -1.83×10^-, 2.2m

y = B1x + B0

y = B3x**3 + B2*x**2 + B1x + B0

pot-spend = np.linspace(0, 500, 100)

Scikit-learn

- Scikit-learning is a library containing many machine learning algorithms.
- It utilizes generalized estimator framework to calling the models.
- This mean the way algorithm are imported, fitted and used in a uniform across all algorithms.
- This allows user to easily swap algorithms in and out and test various approches.
- This uniform framwork also mean user can easily apply almost any algorithm effectively without truely understanding what the algorithm is doing!
- Many convience tools
  - train-test-split
  - Metrics
  - Cross-validation

Philosophy of Scikit-learn

- statsmodels – library

Code #1

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y)
```

```
from sklearn.model_family import ModelAlgo
```

Example not true

Instansce of a model and can change parameters

```
mymodel1 = ModelAlgo(param1, params2)
mymodel.fit(X_train, y_train)
predictions = mymodel.predict(X_test)
```

Linear Regression with Scikit-learn

part-one

Data-setup and Model Training

```
imports numpy as np
import pandas as pd
import matplotlib as pyplot
import seaborn as sns

df = pd.read_csv("file")
```

Visualization.

```
x = df.drop('sales', axis=1)
y = df['sales']

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
len(df)

from sklearn.linear_model import LinearRegression
#help(LinearRegression)

model = LinearRegression();
model.fit(x_train, y_train)
model.predict(x_test)
```

Evaluation Metrics

1) Mean Absolute Error
2) Mean Squared Error
3) Root Mean Square Error

Only Apply to Regression task

Mean Absolute Error = $\frac{1}{n} \times \sum |y_i - \hat{y}_i|$

↓ problem

(will not punish when values are far off)

Mean Squared Error =

$$\frac{1}{n} \sum_{1}^{n} (y_i - \hat{y}_i)^2$$

Larger error are "punished" more then with MAE making MSE more popular.

- Mean Squared Error (MSE):
  - Issue with MSE:
    - Different units than y
    - ~~It reports Units of y squared!~~
      - hard ↓ dollar

Root Mean Squared error = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

• most popular
also • also punish larger values
  - Also will be in same meters.

#? What is good "RMSE"
  - Contex is everything!

A RMSE of $10 is fantastic for predicting the price of a house, but horible for predicting the price of a candy bar.

Code #

```
model.fit(X_train, y_train)
test_prediction
from sklearn.metrics import mean_absolute_error,
                            mean_squared_error
df["sales"].mean()

sns.histplot(data=df, x="sales")
                    bin=20

mean_absolute_error(y_test, test_prediction)
```

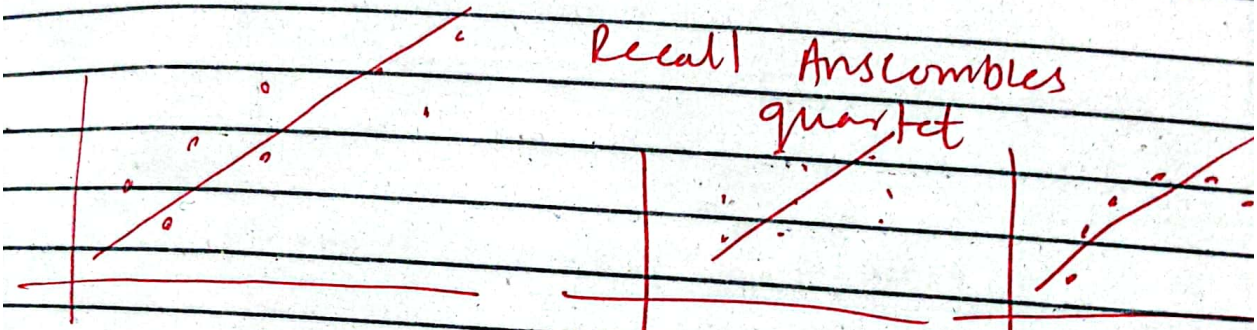
mean_squared_error(y_test, test_prediction))

# RMSE
np.sqrt(mean_squared_error(y, ŷ))

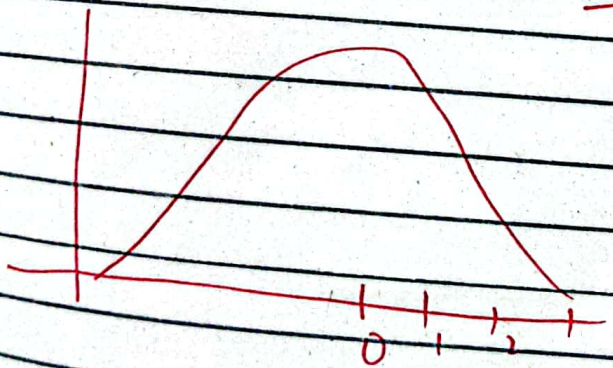
RMSE is standard deviation
MSE variance of the prediction

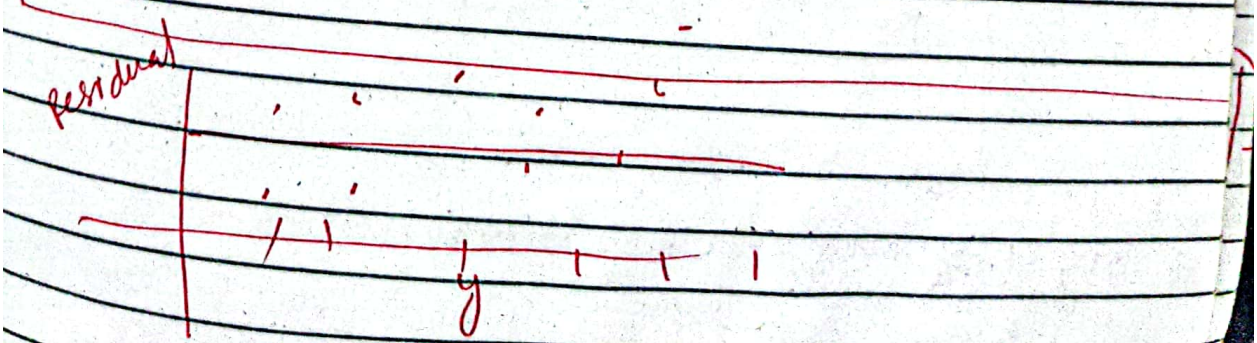
Evaluating Residuals

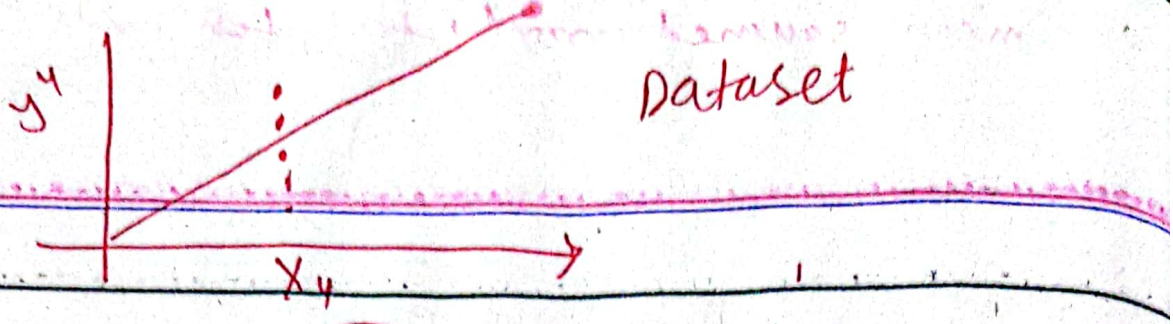
Residual = $(y - \hat{y})$

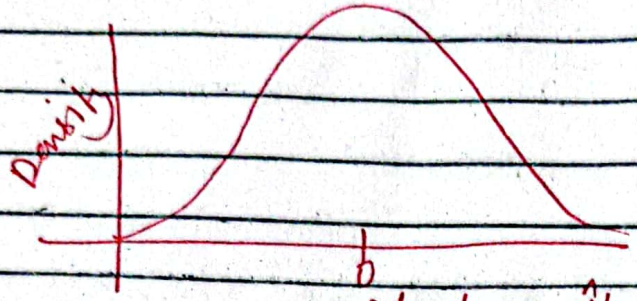


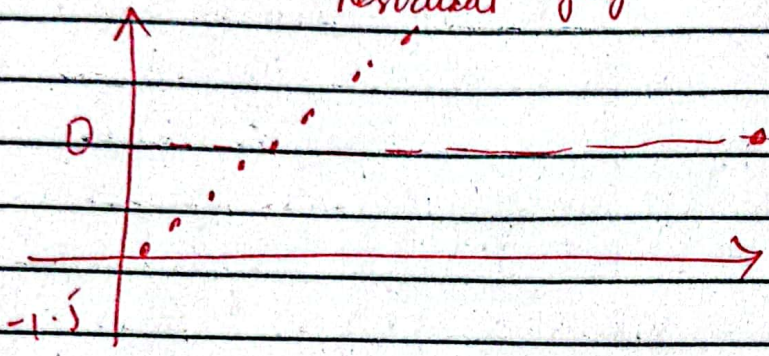
residual tries to normal distribution







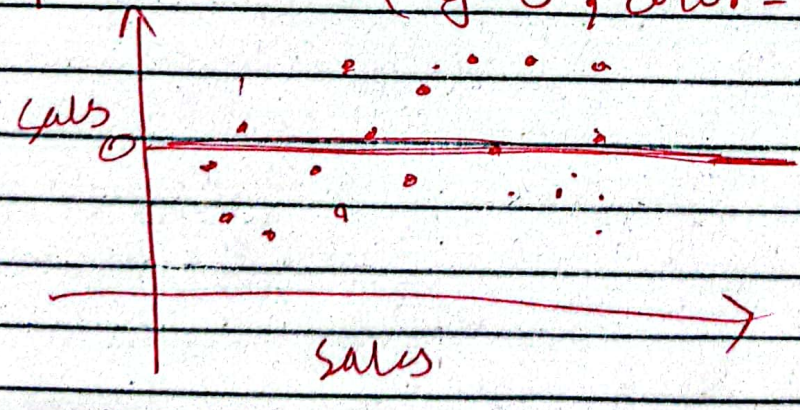




Code # Residuals

```
test_residuals = y_test - test_pred
# test resi
```

```
sns.scatterplot(x = y_test, y = residuals)
plt.axhline( y=0 , color = 'r', ls='---')
```




```
sns.displot ( test_residuals, bins = 25 ,
                                kde = True)
```

import scipy as sp

create a figure and axis to plot on

fig, ax = plt.subplot(figsize=(6,8), dpi=100)

probplot returns the raw value if needed.

we just want to see the plot, so we assign these values to _

_ = sp.stats.probplot(test_residual, plot=ax)

Model Deployement

Code #

final_model = LinearRegression()

final_model.fit(X, y)

final_model.coef_

array [0.0457645 , 0.18853002 , -0.00108749])

↓

Very less value mean there is no real relation ship between sales with Newspaper.

y_hat = final_model.predict(X)

from joblib import dump, load

dump(final, 'final_sales_model.joblib')

load_model = load('final_sales_model.joblib').

Campaign = [[ ]]

149 TV , 22 Radio, 12 Newspaper

Campaign = [[149, 22, 12]]

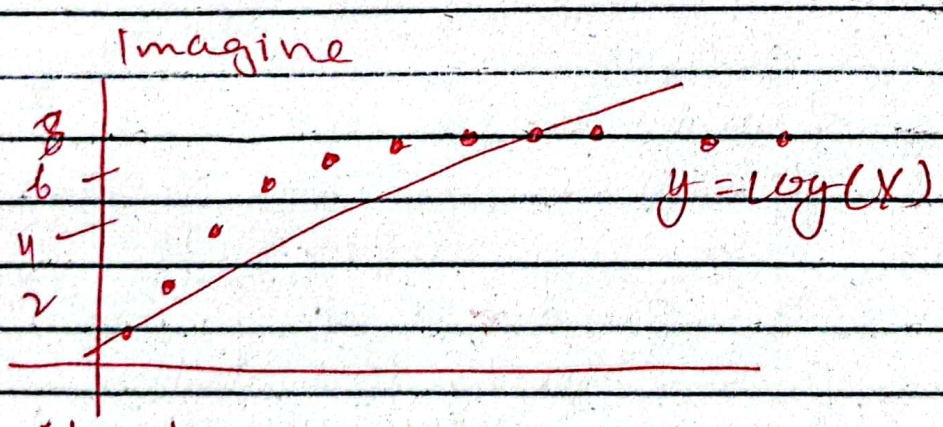
loaded_model = Model.predict(campaign)

~~predict~~ print(loaded_model)

Polynomial Regression:-

Two issues polynomial Regression will address for us -

- Non linear feature relationship to Labels -
- Interaction terms between features.



iteration terms :-

Syngergy between original multiple features

- polynomialfeatures

The features created include:

- The bias (the value of 1.0)
- Value raised to the power for each degree (e.g. X^1, X^2, X^3 ....)
- Interaction between all pairs of features- $X_1 * X_2$, $X_1 * X_3$

A, B

1, A, B, A^2, B^2, AB

1, X_1, X_2, X_1^2, X_2^2, $X_1 X_2$, X_2^2

$X_1 = 2$ and $X_2 = 3$

1, 2, 3, 4, 6, 9

Imports

Files Read

```
X = df.drop("Sales", axis=1)
y = df['Sales']
```

```
from sklearn.preprocessing import PolynomialFeatures
```

```
polyconverter = PolynomialFeatures(degree=2, bias= False)
```

include.

```
polyconverter.fit(X)
polyconverter.transform(X)
```

$[2.301 \times 10^{2} \quad 3.78 \times 10^{2} \quad \ldots\ldots\ldots]$

fit - first and

Train-Test-split

polynomial converter also need saved

```
from sklearn.model_selection import
                    Train_test split

X_train, X_test, y_train, y_test = train_test_split
                    (poly_features, y, test_split

model.fit(X_train, y_train)

test_predictn = model.predict(X_test)
# model_coefficient
mean_absolute_Error(y_test, y_models)
mean_squared_Error = (y_test, y_models)

model.coefficient
```

Bias-Variance Trade-off

also known as

Overfitting vs Underfitting

Amodel

In general,

- Increase model complexity in search of better performance leads to a Bias-Variance tradeoff.
- We want to have a model that can generalize well on unseen data - but also

account for variance and patterns in the know data.

- Extreme bias or variance can lead to poor model prediction.

high bias (underfit)
high variance (overfit)

Overfitting :-

- When model fits much to the noise from the data.
- This often result in low error on training set but high error on test/validation sets

to much on noise or variance

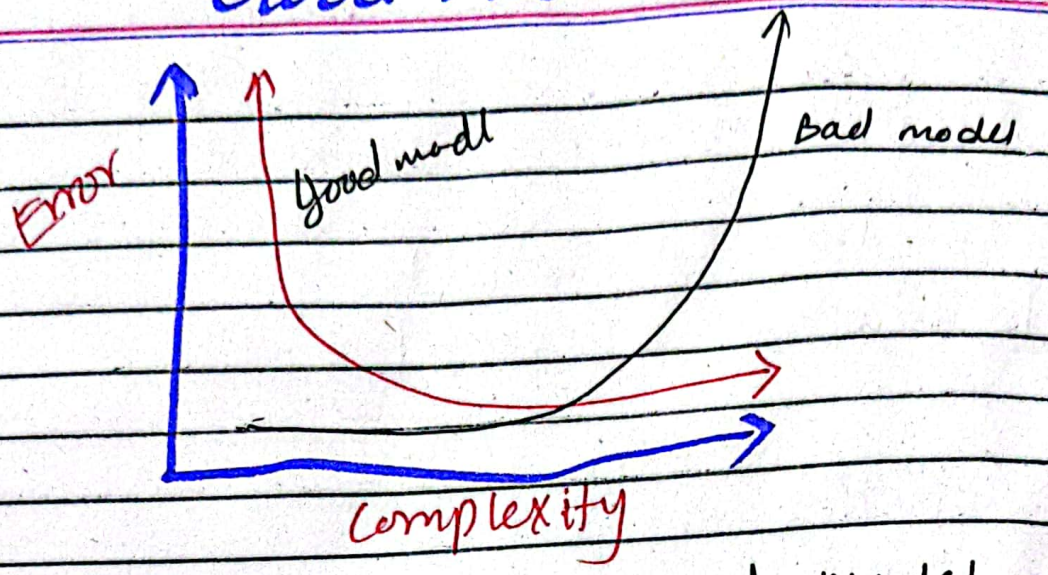
Underfitting :-

Too much bias

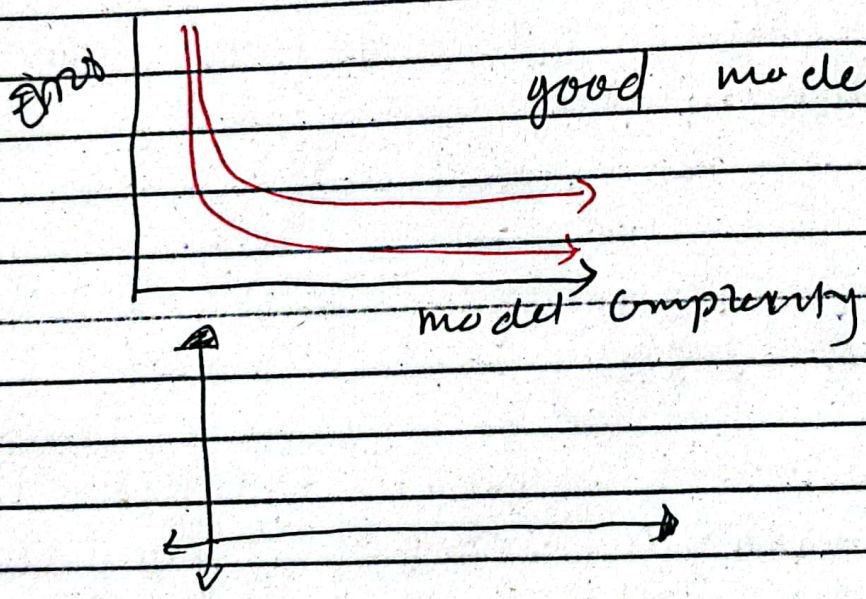
- ~~the ends~~ model does not capture the underlaying trends of the data and does not fit the data well enough.
- Low variance but bias
- Underfitting is often in result of an excessively simple model.

- Model has high bias and is generalizing too weak.

Good Model



plot the error and model complexity



Polymical Regression

Error Againts degree of complexity

```
# create the different polynomial
# split poly features, train_test split
# stores/save the rmse for both
  train and test
# plot the Result(error vs poly ord

Train - rmse_error = [ ]
test-rmse - errors = [0]

for d in range(1,10):
    poly_converter = polynomialfeatures
                     (degree=d,)
                     include_
                     bias=false)
    poly_features = polyconverter.
                    fit_Transform)
    X_train, Xtest, ytrain, ytest =
                    train test split(
    model = linearRegression()
    model.fit(X_train, y_train)
    train_model_predict(Xtrain)
    test preds = models.preds(X,test)

    train_rmse_errors.append( )
    test_rmse_errors.append( )
```

Deploying polynomial model

```
final_poly_converter = PolynomialFeatures(
    degree=3, include_bias=False)
final_model = LinearRegression()
full_converted_X = final_poly_converter
    .fit_transform(X)
final_model.fit(full_converted_X, y)
from joblib import dump, load
dump(final_poly_converter, 'final_
    converter.joblib')
loaded_model = load('final_
```

Regularization :-

- Regularization seeks to solve a few common model issue by:
  - Minimizing model complexity
  - penalizing the loss function
  - Reduce model overfitting (add more bias to reduce variance)

In general, we can think of regularization as a way to reduce model overfitting and variance-

- Requires some additional bias
- Require search for optimal penalty hyperparameter

Three Main Regularization:-
- L1 Regularization
  - Lasso Regression
- L2 Regularization
  - Ridge Regression
- Combining L1 and L2
  - ElasticNet

L1 regularization:-

Add a penality equal to the absolute value of the magnitude of coeficients.

- Limit the size of coeficients
- can yield sparse model where some weffiCient can became zero.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = RSS + \lambda\sum_{j=1}^{p}|\beta_j|$$

L2 regularization:-

L2 regularization adds a penality equal to the square of the magnitude of coefficient.

- All coeficient are shrunk by the same fector.
- Does not eliminate coefficient.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = RSS + \lambda\sum_{j=1}^{p}\beta_j^2$$

Elastic Net :

ElasticNet combines L1 and L2 with the addition of an alpha parameters deciding the Ratio between them.

$$\sum_{i=1}^{n} \frac{(y_i - x_i^T \hat{\beta})^2}{2n} + \lambda \left(\frac{1-\alpha}{2} \sum_{j=1}^{m} \hat{\beta}_j^2 + \alpha \sum_{j=1}^{m} (|\hat{\beta}_j|) \right)$$

These Regularization methods do have a cost:

- Introduce an additional hyperparameter thats need to be tuned.
- A multiplier to the penalty to decide the "strength" of the penality

Later on, we will actually cover L2 regularization

Feature Scaling

- many benefits in machine learning -
  - some machine learning models that rely on distance metrics (e.g KNN) requires scaling to perform well
  - Improves the convergences of steepest decent algorithms, which do not posses the property of scale invarience -
  - For decent algorithms helps in converge at same time -

Those algos where
- Scaling won't have any effect
  Decesion Tree, Regression Tree, Random Forest..
- Important in comparing measurent that have different units -
- Allow us to directly compare model coeficients to each other -
- Features scaling caveats:
  - Must alwcdy scale new unseen data before feeding it to model -
  - Effects direct interpretability of features coefficients -

- Easier to compare coefficients to one another, harder to relate back to original unseen features-

- Feature scaling benefits:-
  - Can lead to great increase in performances
  - Absolutly necessary for some models-
  - Virtually no real downside to scaling features-

Two ways to scale features:-

- Standarization:- Z-score Normal- Rescale data to mean of (μ) zero and Standard deviation of 1 -

standard normal distribution

$$X_{changed} = \frac{X - \mu}{\sigma}$$

- Normalization:- Rescale all data value to be between 0 - 1 -

$$X_{changed} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Many more methods standarization

Standardization:-

A.fit() in standardization will calculate a mean, and standard deviation of the dataset (statistical terms)

Normalization:-

Will calculate X_{min}, X_{max} of the data a.fit() will calculate

- A.transform() call actually scale the data and return the new scaled version of data.

#

Very important consideration for and transform.

- We only fit to training data.
- Calculating statistical information should only come from training data.
- Don't want to assume prior knowledge of the test set.

Data leakage

- Using the full dataset would cause data leakage.
  - calculating statistics from full data leads to some information of the test set leaking into training process upon transform conversion.

Feature Scaling process:-

- perform train test split
- fit to training features data
- ~~Transform test features data~~
-

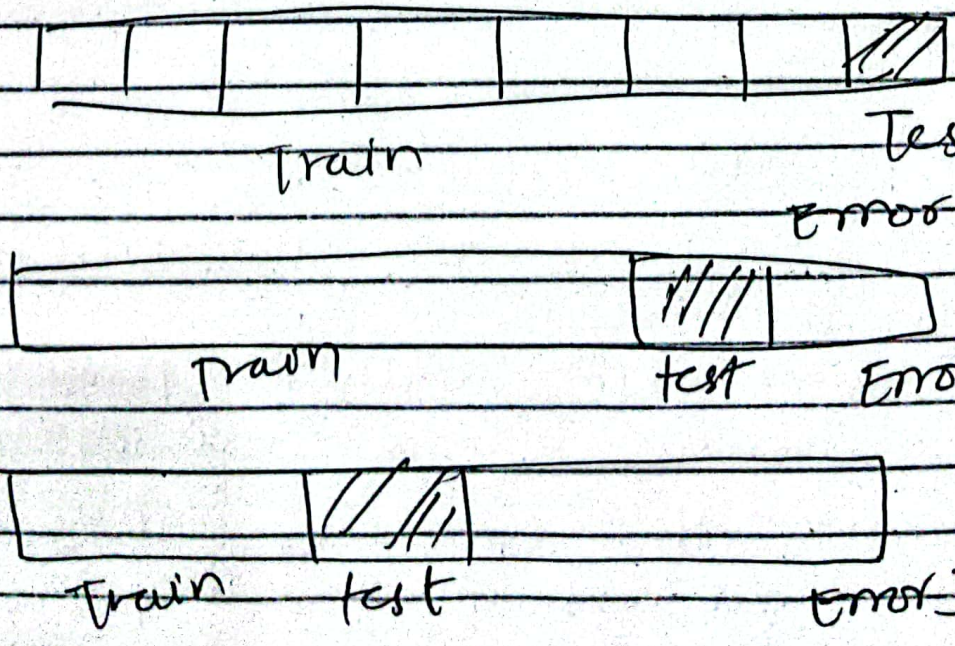
Do we need to scale the label

- ~~can negatively effect~~ Stochastic gradient descent.
- not advised

Cross-Validation:-

- Cross Validation is a more advanced set of methods for splitting data into training and testing sets-

Is there a way to:
Train on all data &
Evaluate on all data



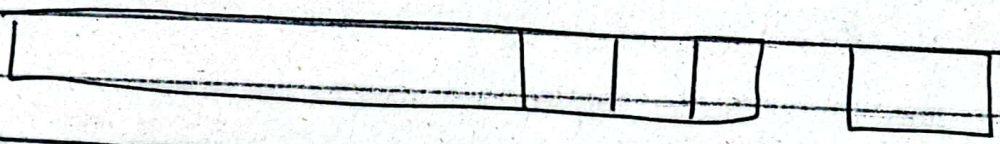
mean with the Error

computationally costly

- This is known as k-fold cross-validat
- Common choice for k is 10 so each test set is 10 percent of your total dataset.

- largest k possible would be k equal to the numbers of rows.
- This is known as leave one out cross validation.
  - computationally expensive.

Trian | Validation hold out test set :-



- Train | validation | test split

Data Setup

Regularization for Linear Regression Data setup

```
Import numpy as np
import pandas as pd
import ---        plt
impor             sns

df = pd.read_csv("Advc.csv")

X = df.drop('sales', axis=1)
y = df['sales']
```

From sklearn preprocessing
Import PolynomialFeatures

```
poly_converter = polynomialfeatures( degree=3,
                                     includebias=False)
poly_features = polynomial.conve.fit transfor
                                     (X)-

(X.shape)
(200, 19)
```

```
from sklear.model-selection import test
X_train, X_test, y_train, ytest = tn (poly, y, 42)

from sklear.preprocessing import Standardsca
    scaler = Standarscaler();
    Scaler.fit(X_train)
```

```
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

X_train[0]
```

Ridge Regression :-

Ridge Regression is a regularization technique that works by helping reduce the potential for overfitting to the training data.

- It does this by adding in a penalty term to the error that is based on the squared value of the coefficient.

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots + \hat{\beta}_p x_p$$

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

we could substite the beta coefficient

$$= \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} - \hat{\beta}_2 x_{i2} \cdots \hat{\beta}_p x_{ip})^2$$

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2$$

$$\text{Error} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij}\right)^2 + \left(\lambda \sum_{j=1}^{p} \beta_j^2\right)$$

Ridge Regression seeks to minimize this entire term RSS + penalty

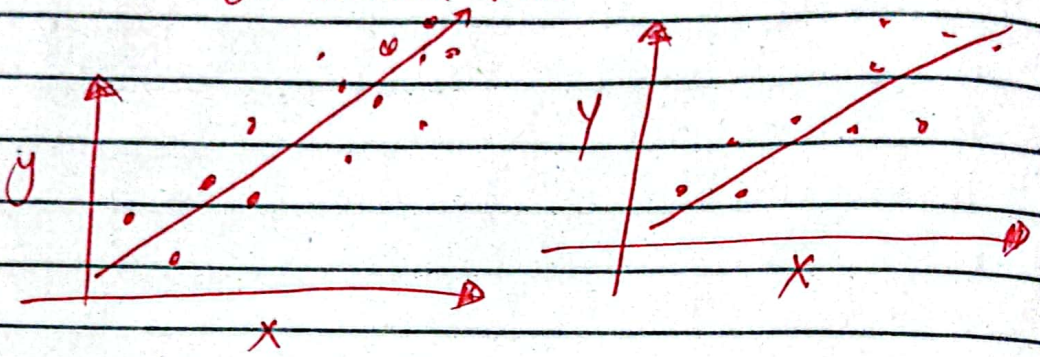
$$\lambda \sum_{j=1}^{p} \beta_j^2$$

shrinkage penalty has a tunable lambda parameters!

$\lambda = 0 - \infty$

↓ Simply

0 — mean RSS



#

L2 ~~regu~~ Regularization with python and scikit-learn

- **Important Note:-**

for cross-validation metrics, sklearn use a "scorer object"

All scorer object follows the convention that higher return values are better then the lower Return ~~val~~ value.

- But higher RMSE is actually worse!
- So scikit-learn fixes this by using a negative RMSE as its scorer metrics.
- This allows uniformaty across all scorer metrics, even across different task types.

The same idea of uniformity across model classes applied to referring to penalty strength parameters as alpha

```
from sklearn.linear_model import Ridge
# help(Ridge)

ridge_model = Ridge(alpha = 10)
ridge_model.fit(X_train, y_train)
test_prediction = ridge_model.predict(X_test

MAE = mean_absolute_error(ytest, prels)
RMSE = sqrt

from sklearn.linear_model import RidgeCV

ridge_cv_model = RidgeCV(alpha=(0.1, 1, 10.))
ridge_cv_model.fit(X_train, y_train)
ridge_cv_model.alpha_

from sklearn.metrics import SCORERS

SCORERS.keys()

new scoring = "neg_mean_absolute_error"

MAE:
RME:
ridge_cv_model.
```

Lasso Regression

L1 Regularization

- L1 regularization add an penalty term equal to absolute value of the magnitude of coefficients.

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda \sum_{j=1}^{p}|\beta_j| = RSS + \lambda \sum_{j=1}^{p}|\beta_j|$$

- limit the size of coefficient
- can yield sparse models where some coefficient can becomes zeros.

- Lasso can force some of the coefficient estimates to be exactly equal to zero when the the tunning parameter λ is sufficiently large.
- Similar to subset selection, the Lasso performs variable selection.
- Model generated from lasso are generally much easier to interpret.
- LassoCV with sklearn operates on default a number of alphas within a range instead of providing the alphas directly.
- Let explore the result of lasso in python and scikit learn.

Acynoym

LASSO – for least absolute shrinkage and selecton operator :-

```
from sklearn.linear_mode import lasso
from sklearn.linear_model import LassoCV

lasso_cv_cmodel = LassoCV(eps=0.001,
                  max_iteration=1m, n_alphas=100,
lasso_cv_model.fit(X_train, y_train, cv=5)

lasso_cv_model.alpha_
0.4945

test_predict5 = lassoCK.predict(X_test)
```

or

ElasticNet

L1 and L2 Regularization

We know that Lasso is able to shrink coefficients to zero, but we haven't taken a deeper dive to how and why that is

Ability becomes more clear when learning about elastic net which combines lasso and Ridge together.

Lasso

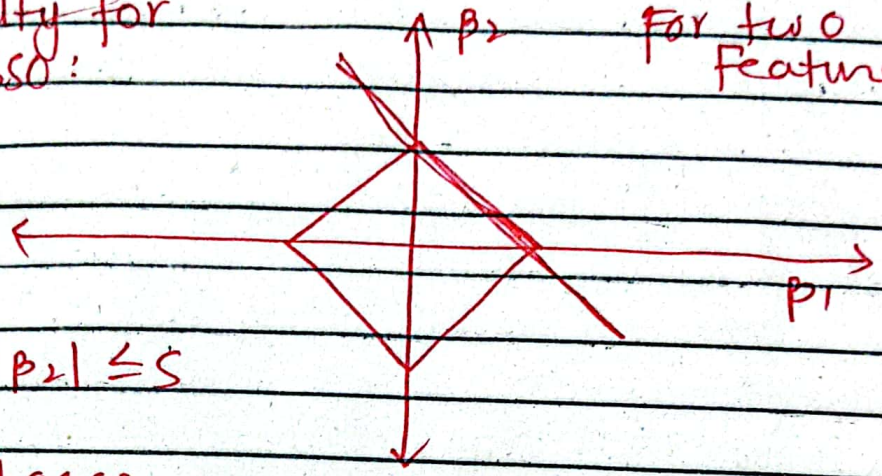
$$\underset{\beta}{\text{minimize}} \left\{ \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \right\} \text{ subject to } \sum_{j=1}^{p} |\beta_j| \leq s$$

and Ridge

$$\underset{\beta}{\text{minimize}} \left\{ \left\{ \sum_{i=1}^{n} \left(y_1 - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right) \right\}^2 \right\} \text{ subject to } \sum_{j=1}^{p} \beta_j^2 \leq s$$

- Lasso:
  - A convex object that lies tangent to the boundary, is likely to encounter a corner of a hypercube, for which some components of β are identically zero.

penalty for lasso:



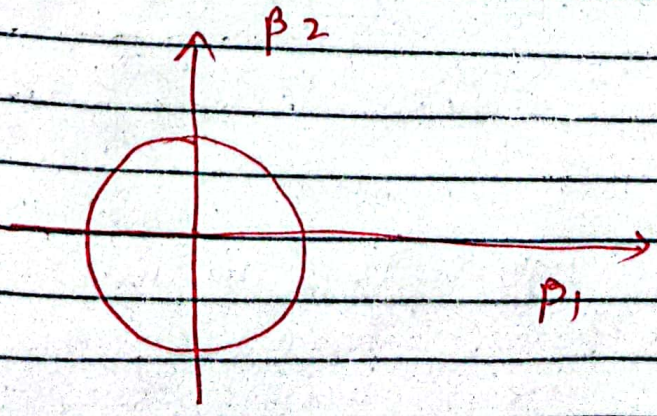
For two Features

$|\beta_1| + |\beta_2| \leq s$

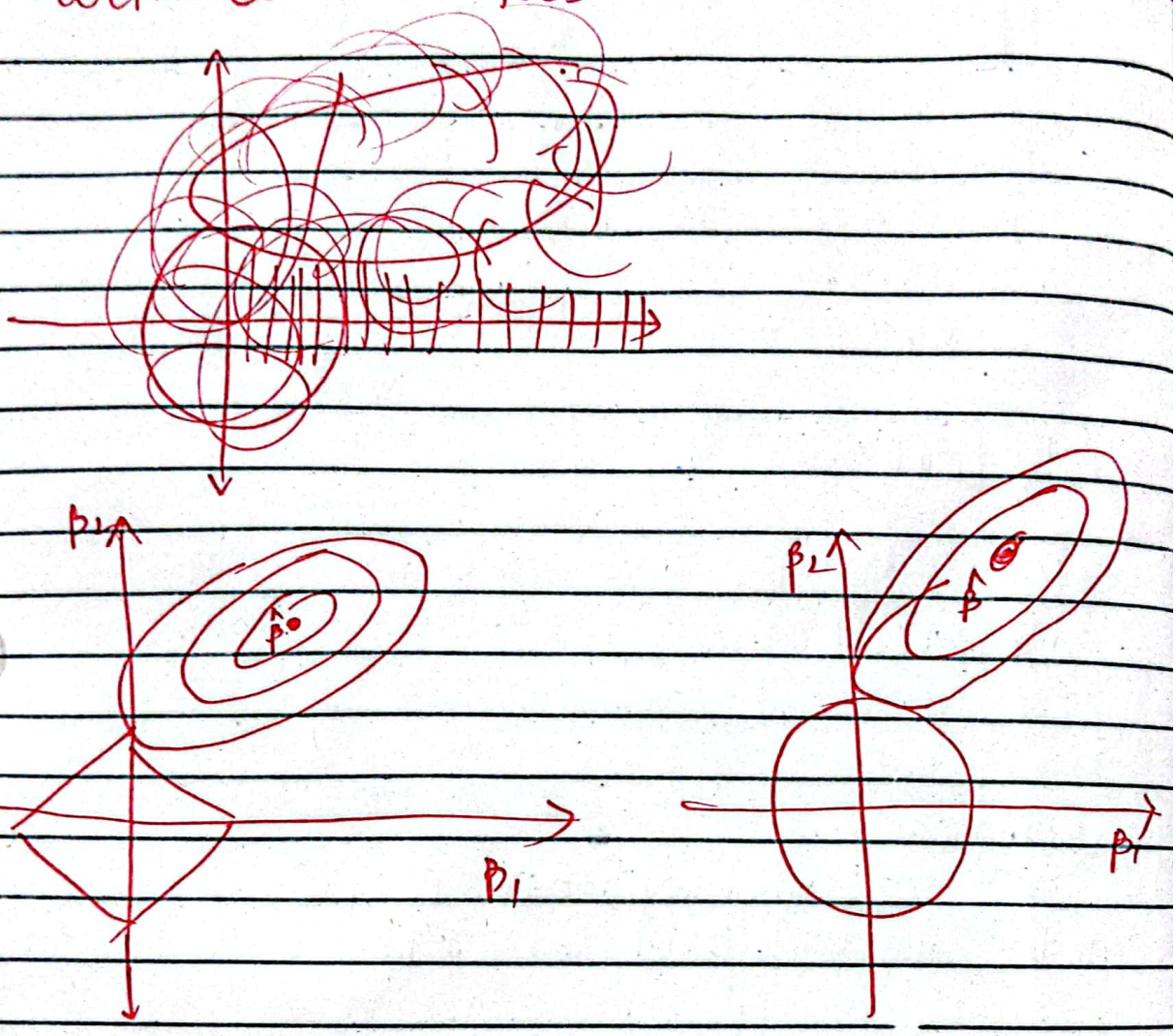
- In lasso we are dealing with hypercube.

Ridge:-

In case of an n-sphere, the points on the boundary for which some of the of β are zero are not distinguished from others and the convex object is no more likely to contact a point at which some components of β are zero than one for which none of them are.



This is why lasso is more likely to lead coefficient to zero -
This diagram is also commonly shown with contour RSS.



ElasticNet

- seeks to improve both L1 and L2 regularization by combining them:

$$\text{Error} = \sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda_1 \sum_{j=1}^{p}\beta_j^2 + \lambda_2 \sum_{j=1}^{p}|\beta_j|$$

Here we are trying to minimize both the squared ~~terms~~ and absolute value terms -
~~There~~ There are two distinct

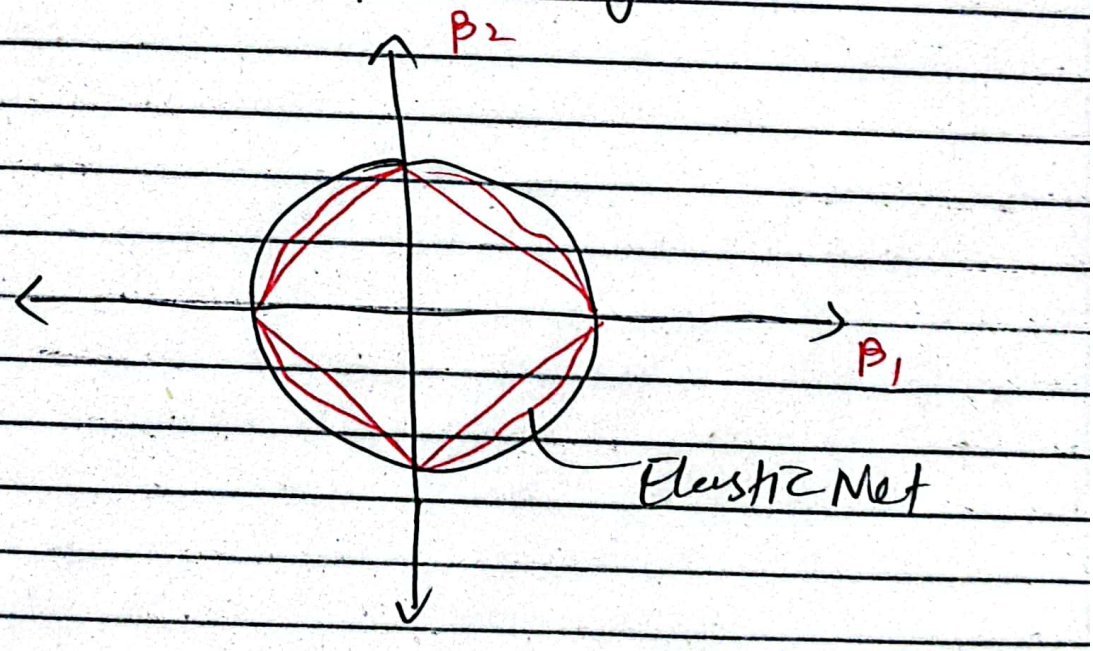
lambda λ_1, λ_2 values for each penalty.
Alternatively we can express this as Ratio between L1 and L2:

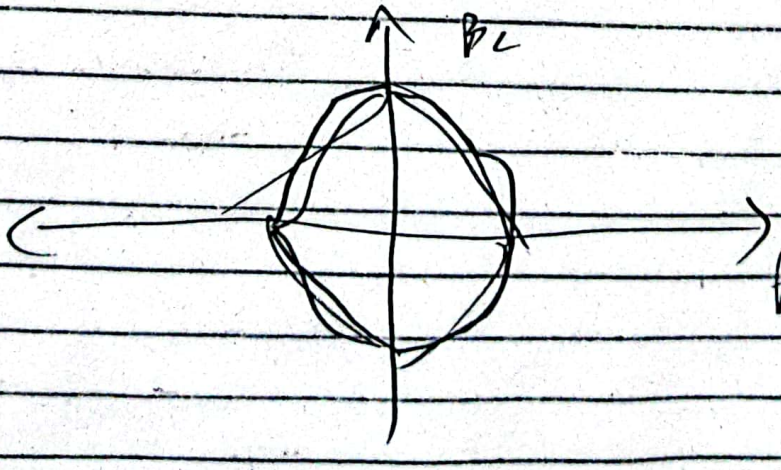
$$\frac{\sum_{i=1}^{n}(y_i - x_i^T\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\hat{\beta}_j|\right)$$

we can also use a simplified notation

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}\left(\|y - X\beta\|^2 + \lambda_2\|\beta\|^2 + \lambda_1\|\beta\|_1\right)$$

Elastic Net penalty Region:





Code #

```
from sklearn.linear_model import ElasticNetCV

elastic_model = ElasticNetCV(l1_ratio=[0.1,0.2
                0.3. 0.99, 0.1],
                eps=0.001, n_alphas=100,
                max_iteration=100000)

elastic_model.fit(X_Train, y_train)
elastic_model.l1_ratio

elastic_model.alpha_
```