Regression

Tushar B. Kute,
http://tusharkute.com

Regression?

- Regression analysis is a statistical method that helps us to analyse and understand the relationship between two or more variables of interest.
- The process that is adapted to perform regression analysis helps to understand which factors are important, which factors can be ignored and how they are influencing each other.

Regression?

- Introduction, types of regression. Simple regression- Types, Making predictions, Cost function, Gradient descent, Training, Model evaluation.
- Multivariable regression : Growing complexity, Normalization, Making predictions, Initialize weights, Cost function, Gradient descent, Simplifying with matrices, Bias term, Model evaluation

Taken From-
https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html

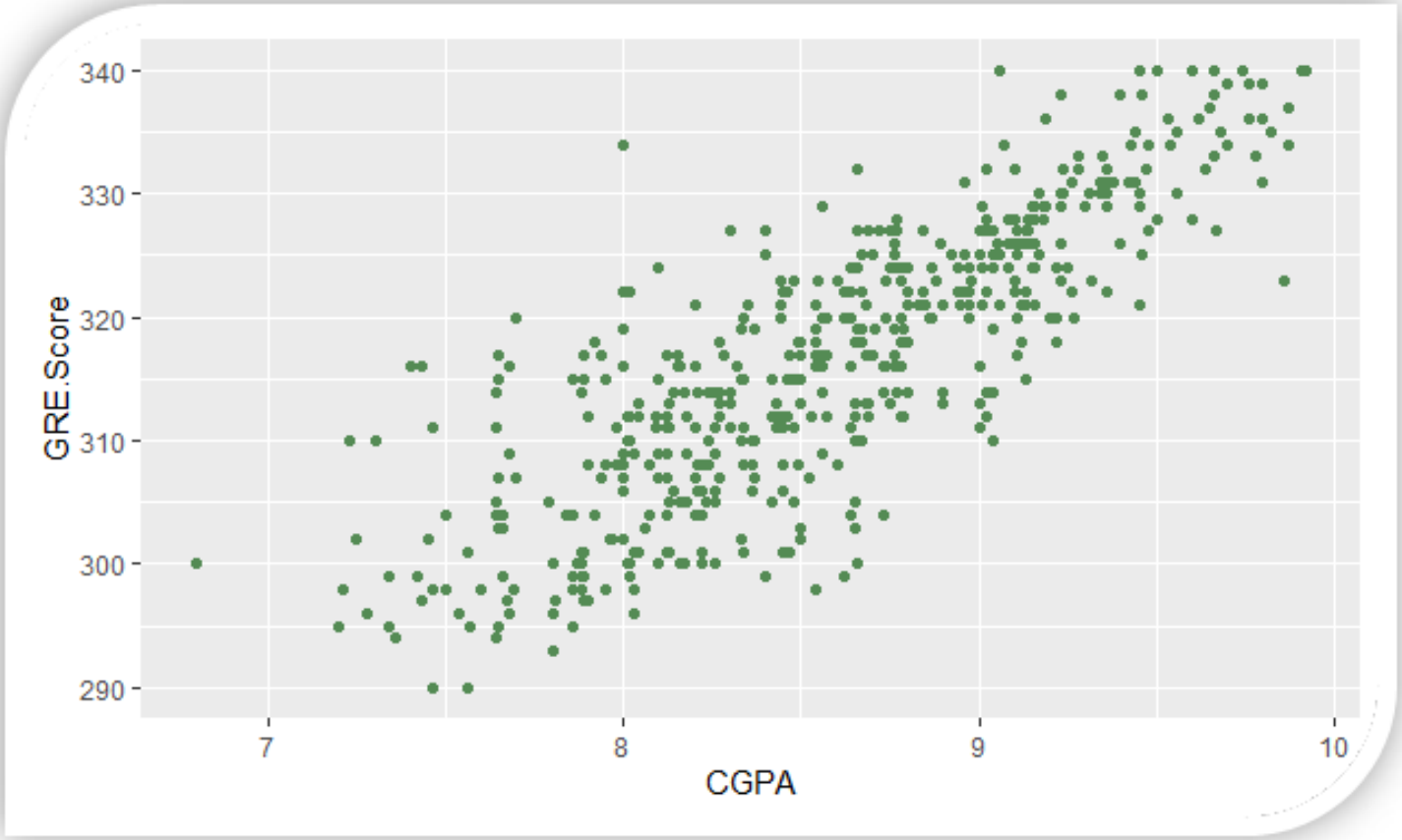
Regression?

- For the regression analysis is be a successful method, we understand the following terms:
  - Dependent Variable: This is the variable that we are trying to understand or forecast.
  - Independent Variable: These are factors that influence the analysis or target variable and provide us with information regarding the relationship of the variables with the target variable.

Example:

GRE Score	CGPA
337	9.65
324	8.87
316	8.00
322	8.67
314	8.21
330	9.34
321	8.20
308	7.90
302	8.00
323	8.60

Example:



Regression

- In regression, we normally have one dependent variable and one or more independent variables.
- Here we try to “regress” the value of dependent variable “Y” with the help of the independent variables.
- In other words, we are trying to understand, how does the value of ‘Y’ change w.r.t change in ‘X’.

$$Y = f(x)$$

Dependent Variable (GRE Score)

Independent Variable (CGPA)

Uses of Regression

- Regression analysis is used for prediction and forecasting. This has a substantial overlap to the field of machine learning. This statistical method is used across different industries such as,
  - Financial Industry- Understand the trend in the stock prices, forecast the prices, evaluate risks in the insurance domain
  - Marketing- Understand the effectiveness of market campaigns, forecast pricing and sales of the product.
  - Manufacturing- Evaluate the relationship of variables that determine to define a better engine to provide better performance
  - Medicine- Forecast the different combination of medicines to prepare generic medicines for diseases.

Terminologies

- Outliers
  - Suppose there is an observation in the dataset that has a very high or very low value as compared to the other observations in the data, i.e. it does not belong to the population, such an observation is called an outlier.
  - In simple words, it is an extreme value. An outlier is a problem because many times it hampers the results we get.

Terminologies

- Multicollinearity
  - When the independent variables are highly correlated to each other, then the variables are said to be multicollinear.
  - Many types of regression techniques assume multicollinearity should not be present in the dataset.
  - It is because it causes problems in ranking variables based on its importance, or it makes the job difficult in selecting the most important independent variable.

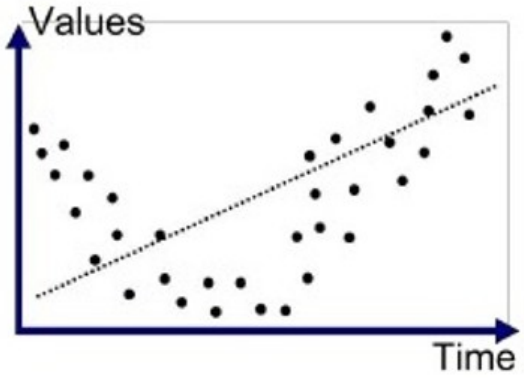
Terminologies

- Heteroscedasticity
  - When the variation between the target variable and the independent variable is not constant, it is called heteroscedasticity.
  - Example-As one's income increases, the variability of food consumption will increase.
  - A poorer person will spend a rather constant amount by always eating inexpensive food; a wealthier person may occasionally buy inexpensive food and at other times, eat expensive meals.
  - Those with higher incomes display a greater variability of food consumption.

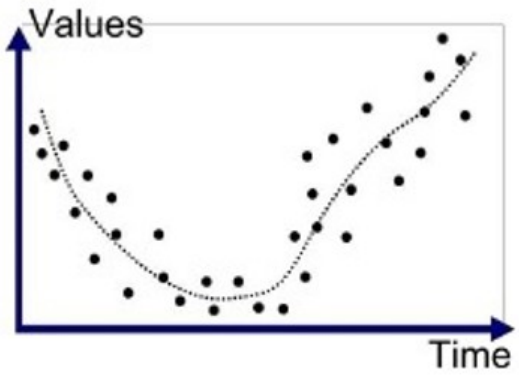
Terminologies

- When we use unnecessary explanatory variables, it might lead to overfitting.
- Overfitting means that our algorithm works well on the training set but is unable to perform better on the test sets. It is also known as a problem of high variance.
- When our algorithm works so poorly that it is unable to fit even a training set well, then it is said to underfit the data. It is also known as a problem of high bias.

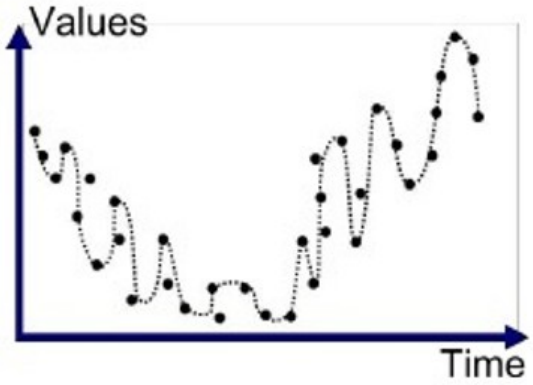
Terminologies



Underfitted



Good Fit/Robust



Overfitted

Types of Regression

- Linear Regression
- Multiple Regression
- Logistic Regression
- Polynomial Regression
- Regularized Models
  - Ridge Regression
  - Lasso Regression
  - ElasticNet Regression
- Outlier Based Model
  - RANSAC

Linear Regression

- The simplest of all regression types is Linear Regression where it tries to establish relationships between Independent and Dependent variables.
- The Dependent variable considered here is always a continuous variable.
- Linear Regression is a predictive model used for finding the linear relationship between a dependent variable and one or more independent variables.

$$Y = a + bx$$

Dependent Variable is continuous

Linear Regression

- Here, ‘Y’ is our dependent variable, which is a continuous numerical and we are trying to understand how does ‘Y’ change with ‘X’.
- So, if we are supposed to answer, the above question of “What will be the GRE score of the student, if his CCGPA is 8.32?” our go to option should be linear regression.

$$GRE = 261 + 6.8CGPA \rightarrow GRE = 261 + 6.8(8.32) \rightarrow GRE = 317.57$$

Simple Linear Regression

- As the model is used to predict the dependent variable, the relationship between the variables can be written in the below format.

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i$$

- Where,
  - Y_i – Dependent variable
  - β_0 — Intercept
  - β_1 – Slope Coefficient
  - X_i – Independent Variable
  - ε_i – Random Error Term

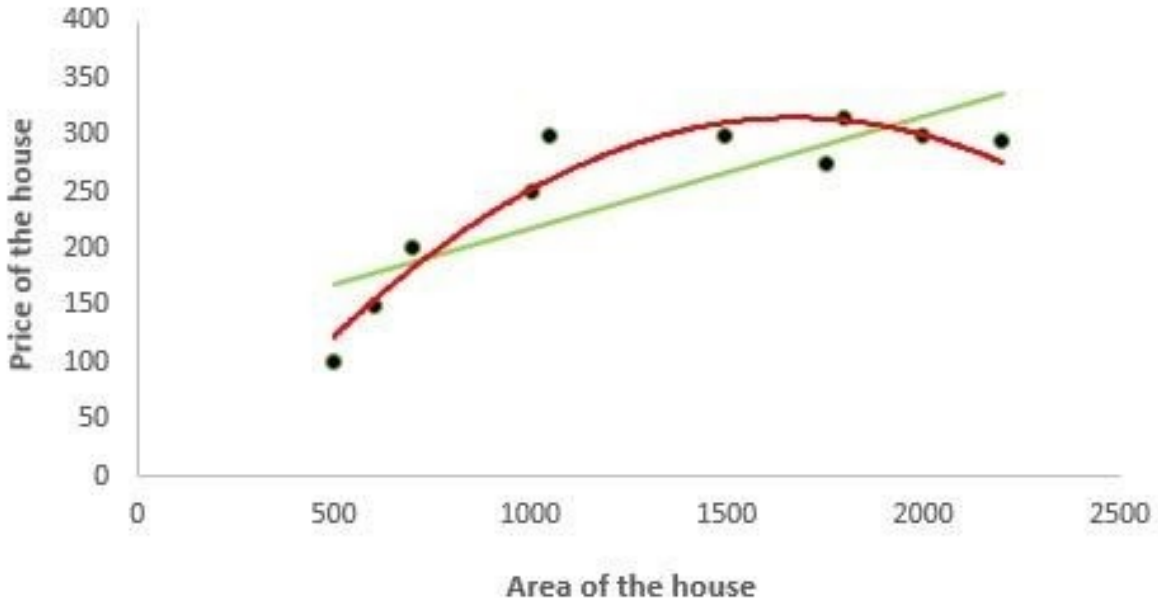
Simple Linear Regression

- The main factor that is considered as part of Regression analysis is understanding the variance between the variables. For understanding the variance, we need to understand the measures of variation.
  - SST = total sum of squares (Total Variation)
    - Measures the variation of the Y i values around their mean Y
  - SSR = regression sum of squares (Explained Variation)
    - Variation attributable to the relationship between X and Y
  - SSE = error sum of squares (Unexplained Variation)
    - Variation in Y attributable to factors other than X

Polynomial Regression

- This type of regression technique is used to model nonlinear equations by taking polynomial functions of independent variables.
- In the figure given below, you can see the red curve fits the data better than the green curve.
- Hence in the situations where the relationship between the dependent and independent variable seems to be non-linear, we can deploy Polynomial Regression Models.

Polynomial Regression



$$y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \varepsilon$$

Logistic Regression

- Logistic Regression is also known as Logit, Maximum-Entropy classifier is a supervised learning method for classification. It establishes a relation between dependent class variables and independent variables using regression.
- The dependent variable is categorical i.e. it can take only integral values representing different classes. The probabilities describing the possible outcomes of a query point are modelled using a logistic function.
- This model belongs to a family of discriminative classifiers. They rely on attributes which discriminate the classes well. This model is used when we have 2 classes of dependent variables. When there are more than 2 classes, then we have another regression method which helps us to predict the target variable better.

Linear Discriminant Analysis (LDA)

- Discriminant Analysis is used for classifying observations to a class or category based on predictor (independent) variables of the data.
- Discriminant Analysis creates a model to predict future observations where the classes are known.
- LDA comes to our rescue in situations when logistic regression is unstable when
  - Classed are well separated
  - Data is small
  - When we have more than 2 classes

Errors in Linear Regression

$$Loss = Error(y, \hat{y})$$

Loss function with no regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Loss function with L2 regularisation

Ridge Regression

- A regression model that uses L1 regularization technique is called Lasso Regression and model which uses L2 is called Ridge Regression.
- The key difference between these two is the penalty term.
- Ridge regression adds “squared magnitude” of coefficient as penalty term to the loss function. Here the highlighted part represents L2 regularization element.

Ridge Regression – Cost Function

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

- Here, if lambda is zero then you can imagine we get back OLS.
- However, if lambda is very large then it will add too much weight and it will lead to under-fitting.
- Having said that it's important how lambda is chosen. This technique works very well to avoid over-fitting issue.

Lasso Regression – Cost Function

- Lasso Regression (Least Absolute Shrinkage and Selection Operator) adds “absolute value of magnitude” of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

- Again, if lambda is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.

Comparing

- The key difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether.
- So, this works well for feature selection in case we have a huge number of features.

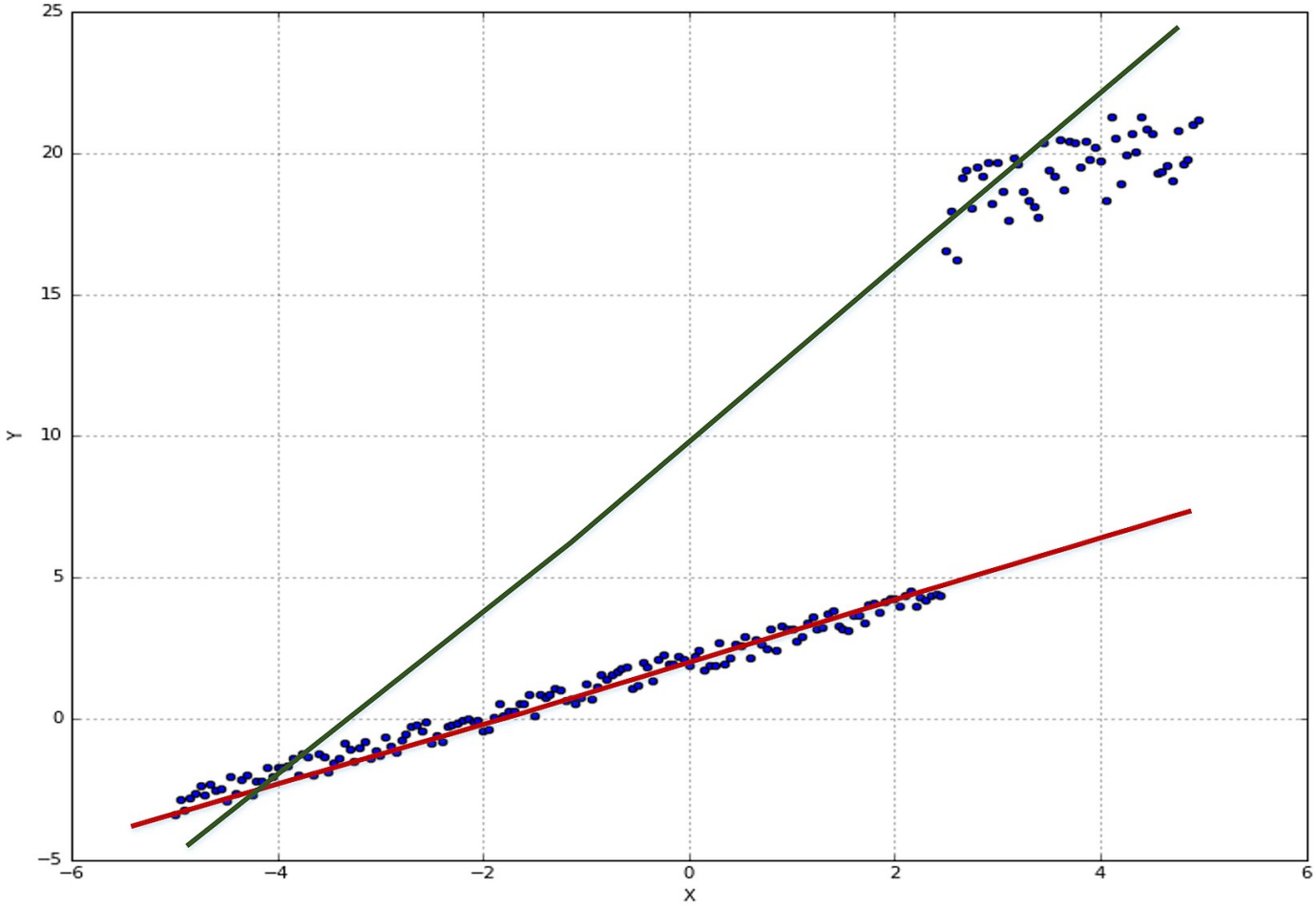
Elastic Net

- Elastic net is a popular type of regularized linear regression that combines two popular penalties, specifically the L1 and L2 penalty functions.
- a hyperparameter “alpha” is provided to assign how much weight is given to each of the L1 and L2 penalties.
- Alpha is a value between 0 and 1 and is used to weight the contribution of the L1 penalty and one minus the alpha value is used to weight the L2 penalty.
- elastic_net_penalty = (alpha * l1_penalty) + ((1 – alpha) * l2_penalty)

Robust Regression

- A common problem with linear regressions is caused by the presence of outliers.
- An ordinary least square approach will take them into account and the result (in terms of coefficients) will be therefore biased.
- In the following figure, there's an example of such a behavior:

Robust Regression



Well-Posed Learning Problems

- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Simple Regression

- Let’s say we are given a dataset with the following columns (features): how much a company spends on Radio advertising each year and its annual Sales in terms of units sold.
- We are trying to develop an equation that will let us to predict units sold based on how much a company spends on radio advertising.
- The rows (observations) represent companies.

Simple Regression

Company	Radio ($)	Sales
Amazon	37.8	22.1
Google	39.3	10.4
Facebook	45.9	18.3
Apple	41.3	18.5

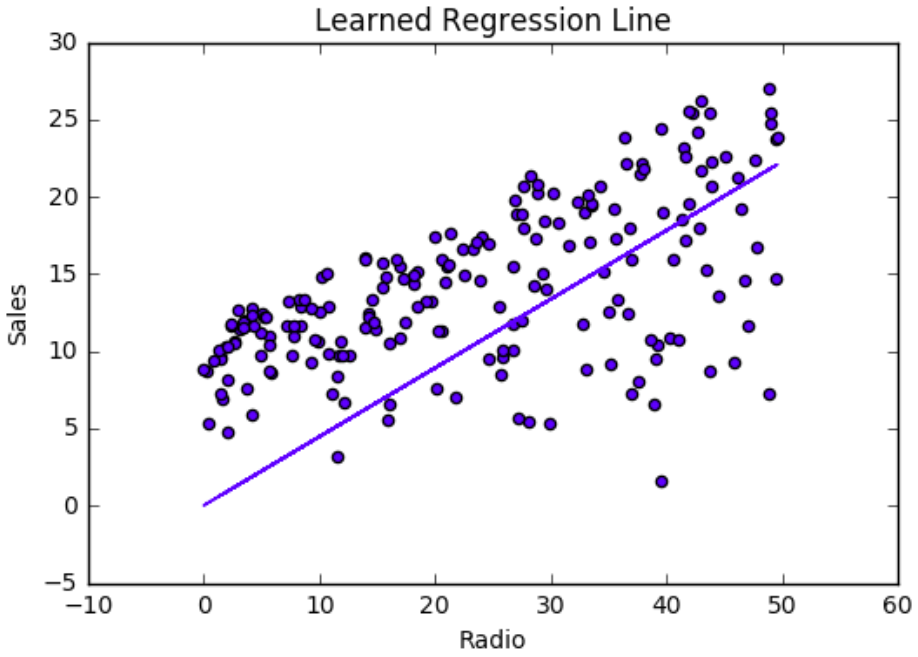
Making Predictions

- Our prediction function outputs an estimate of sales given a company's radio advertising spend and our current values for Weight and Bias.

  Sales=Weight·Radio+Bias
- Weight
- the coefficient for the Radio independent variable. In machine learning we call coefficients weights.
- Radio
- the independent variable. In machine learning we call these variables features.
- Bias
- the intercept where our line intercepts the y-axis. In machine learning we can call intercepts bias. Bias offsets all predictions that we make.

Making Predictions

- Our algorithm will try to learn the correct values for Weight and Bias. By the end of our training, our equation will approximate the line of best fit.



Cost Function

- The prediction function is nice, but for our purposes we don't really need it. What we need is a cost function so we can start optimizing our weights.
- Let's use MSE (L2) as our cost function. MSE measures the average squared difference between an observation's actual and predicted values.
- The output is a single number representing the cost, or score, associated with our current set of weights. Our goal is to minimize MSE to improve the accuracy of our model.

Math

- Given our simple linear equation y=mx+b, we can calculate MSE as:

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

Note

- N is the total number of observations (data points)
- $\frac{1}{N}\sum_{i=1}^{n}$ is the mean
- y_i is the actual value of an observation and $mx_i + b$ is our prediction

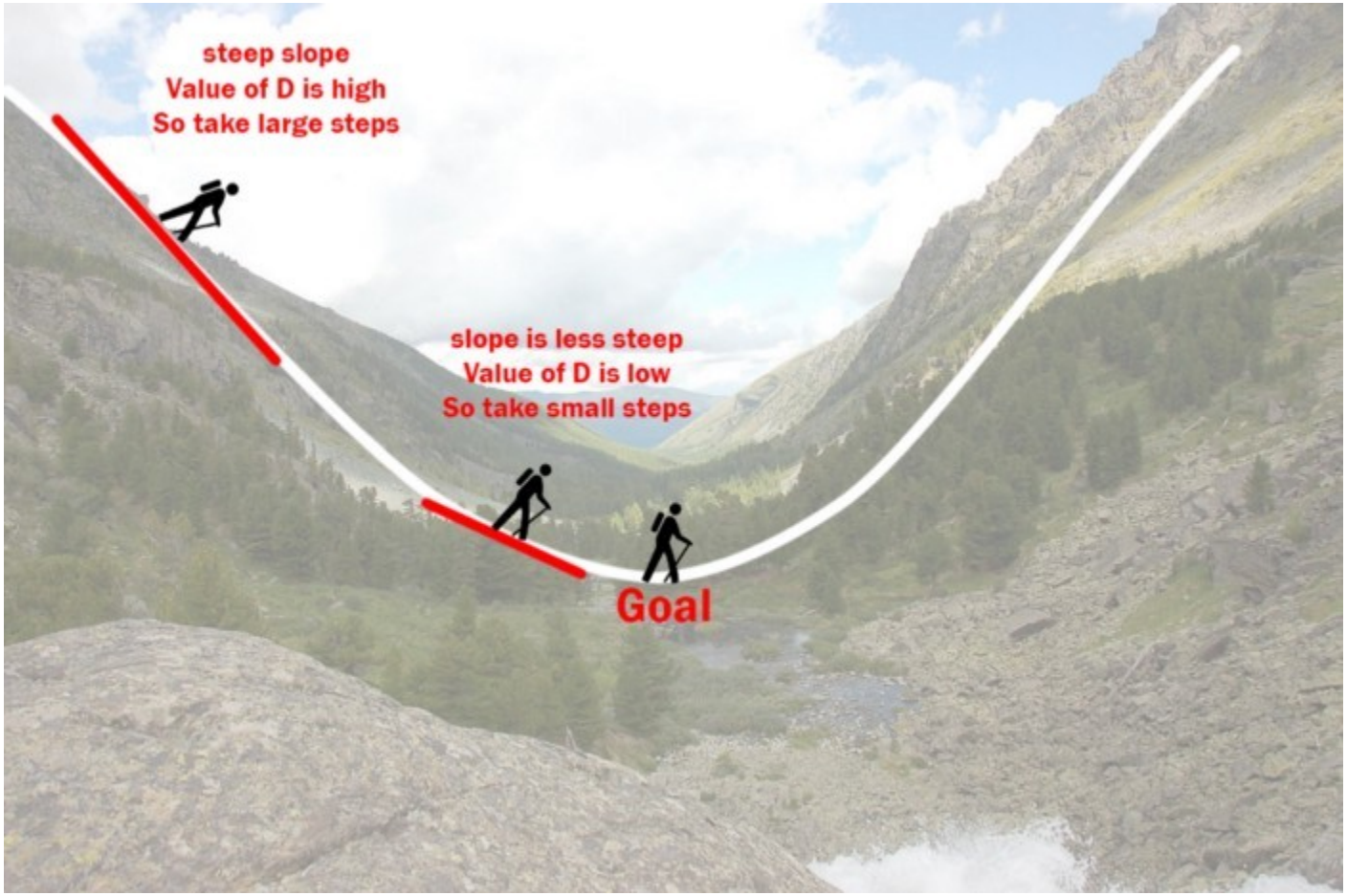
Code

```
def cost_function(radio, sales, weight, bias):
    companies = len(radio)
    total_error = 0.0
    for i in range(companies):
        total_error += (sales[i] - (weight*radio[i] + bias))**2
    return total_error / companies
```

Gradient Descent

- Gradient descent is an iterative optimization algorithm to find the minimum of a function. Here that function is our Loss Function.

Understanding Gradient Descent



Example

- Imagine a valley and a person with no sense of direction who wants to get to the bottom of the valley.
- He goes down the slope and takes large steps when the slope is steep and small steps when the slope is less steep.
- He decides his next position based on his current position and stops when he gets to the bottom of the valley which was his goal.

Example

- Let’s try applying gradient descent to m and c and approach it step by step:
  - Initially let m = 0 and c = 0. Let L be our learning rate. This controls how much the value of m changes with each step. L could be a small value like 0.0001 for good accuracy.
  - Calculate the partial derivative of the loss function with respect to m, and plug in the current values of x, y, m and c in it to obtain the derivative value D.

Example

$$D_m = \frac{1}{n}\sum_{i=0}^{n} 2(y_i - (mx_i + c))(-x_i)$$

$$D_m = \frac{-2}{n}\sum_{i=0}^{n} x_i(y_i - \bar{y}_i)$$

- D_m is the value of the partial derivative with respect to m. Similarly lets find the partial derivative with respect to c, Dc :

$$D_c = \frac{-2}{n}\sum_{i=0}^{n} (y_i - \bar{y}_i)$$

Example

Now we update the current value of m and c using the following equation:

$$m = m - L \times D_m$$

$$c = c - L \times D_c$$

- 4. We repeat this process until our loss function is a very small value or ideally 0 (which means 0 error or 100% accuracy). The value of m and c that we are left with now will be the optimum values.

Example

- Now going back to our analogy, m can be considered the current position of the person. D is equivalent to the steepness of the slope and L can be the speed with which he moves.
- Now the new value of m that we calculate using the above equation will be his next position, and L×D will be the size of the steps he will take.
- When the slope is more steep (D is more) he takes longer steps and when it is less steep (D is less), he takes smaller steps. Finally he arrives at the bottom of the valley which corresponds to our loss = 0.
- Now with the optimum value of m and c our model is ready to make predictions !

Types

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini Batch gradient descent

Batch Gradient Descent

- This is a type of gradient descent which processes all the training examples for each iteration of gradient descent.
- But if the number of training examples is large, then batch gradient descent is computationally very expensive.
- Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

Stochastic Gradient Descent

- This is a type of gradient descent which processes 1 training example per iteration.
- Hence, the parameters are being updated even after one iteration in which only a single example has been processed.
- Hence this is quite faster than batch gradient descent.
- But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

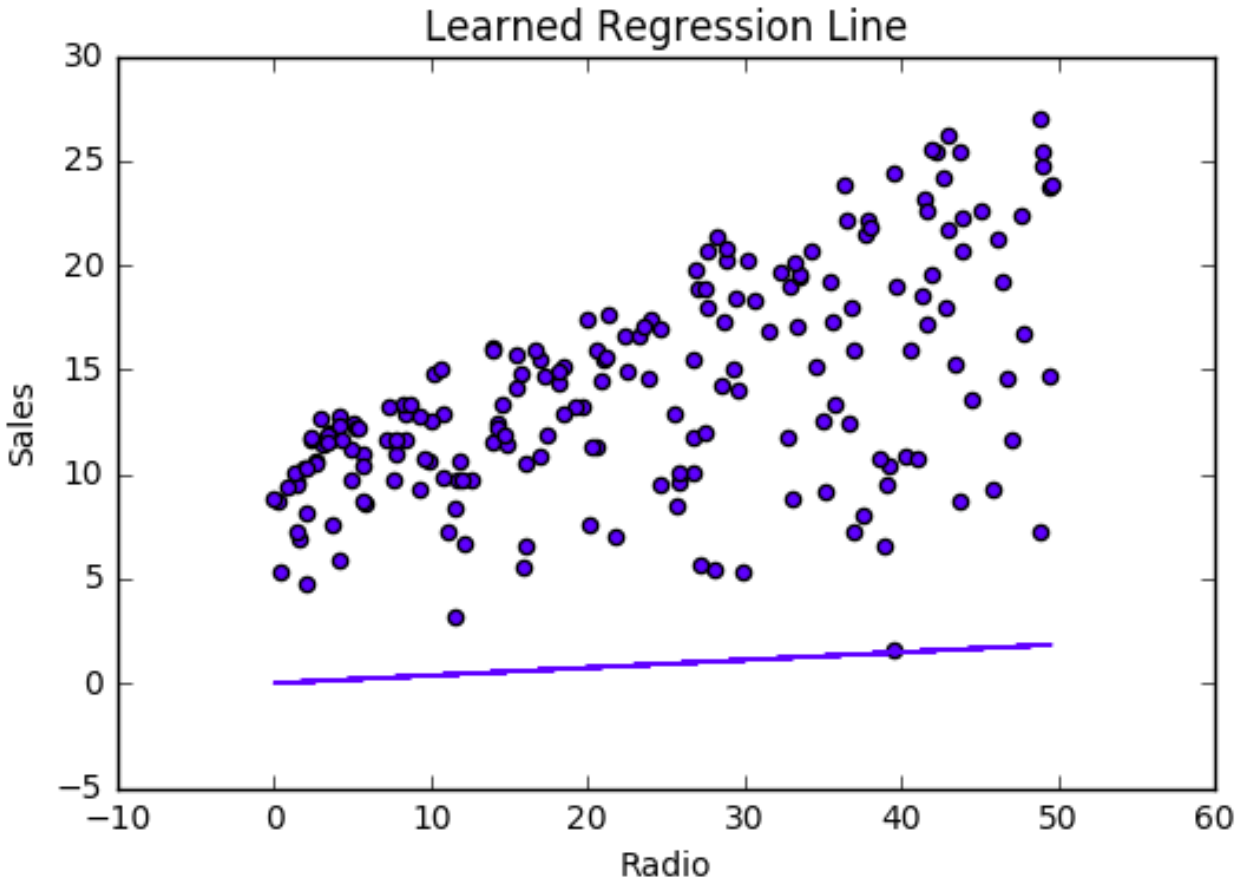
Mini Batch Gradient Descent

- This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent.
- Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go.
- Thus, it works for larger training examples and that too with lesser number of iterations.

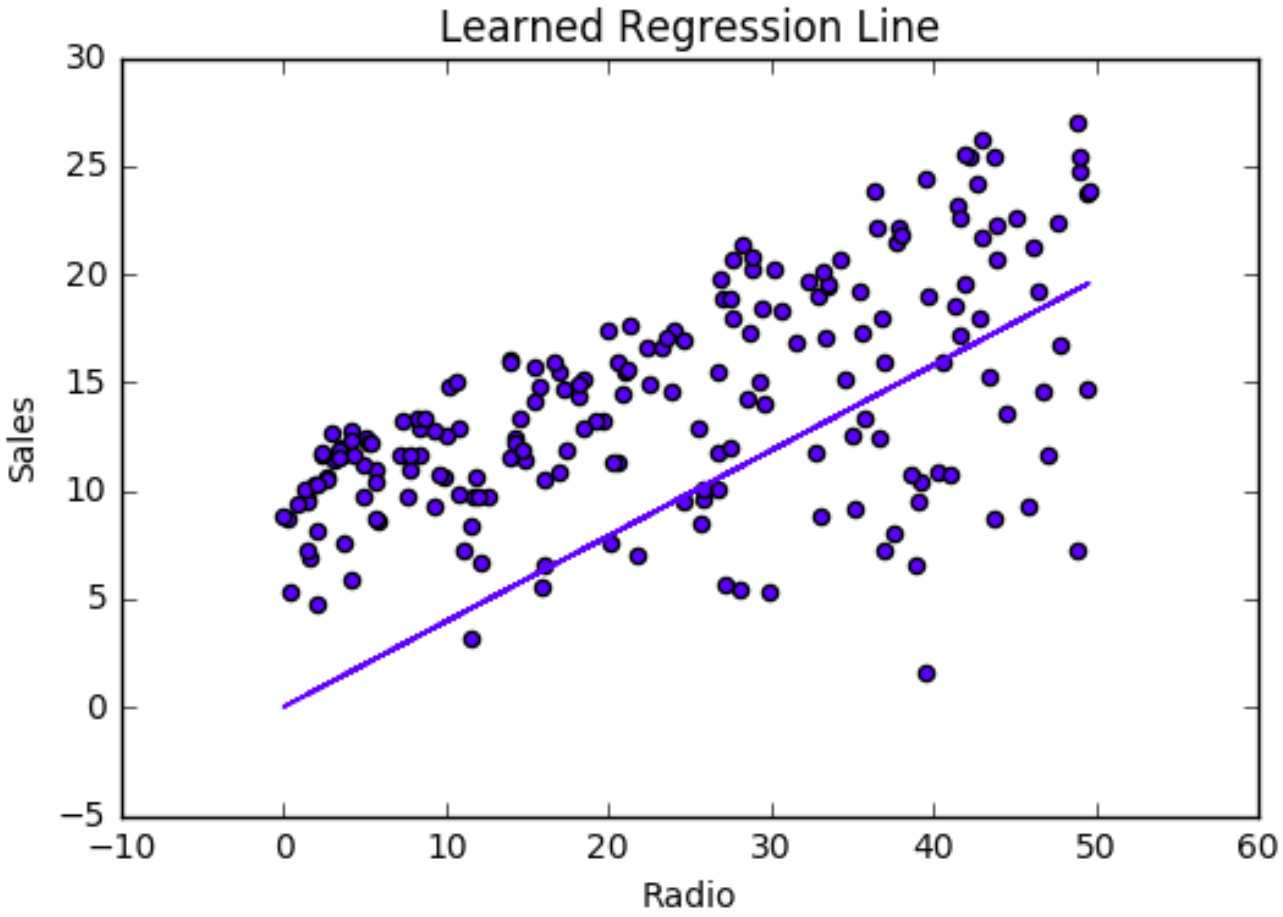
Training

- Training a model is the process of iteratively improving your prediction equation by looping through the dataset multiple times, each time updating the weight and bias values in the direction indicated by the slope of the cost function (gradient).
- Training is complete when we reach an acceptable error threshold, or when subsequent training iterations fail to reduce our cost.
- Before training we need to initialize our weights (set default values), set our hyperparameters (learning rate and number of iterations), and prepare to log our progress over each iteration.

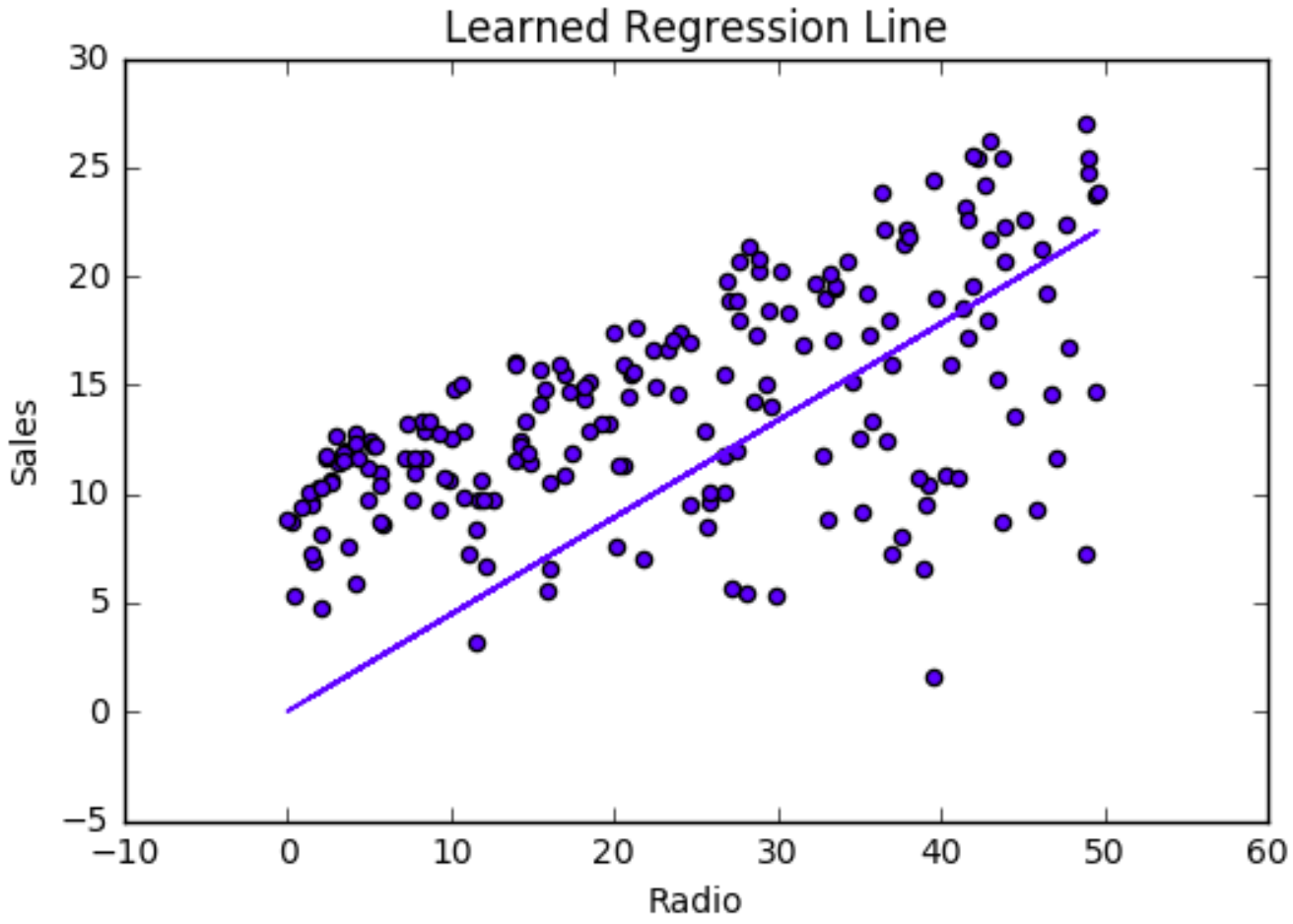
Visualizing



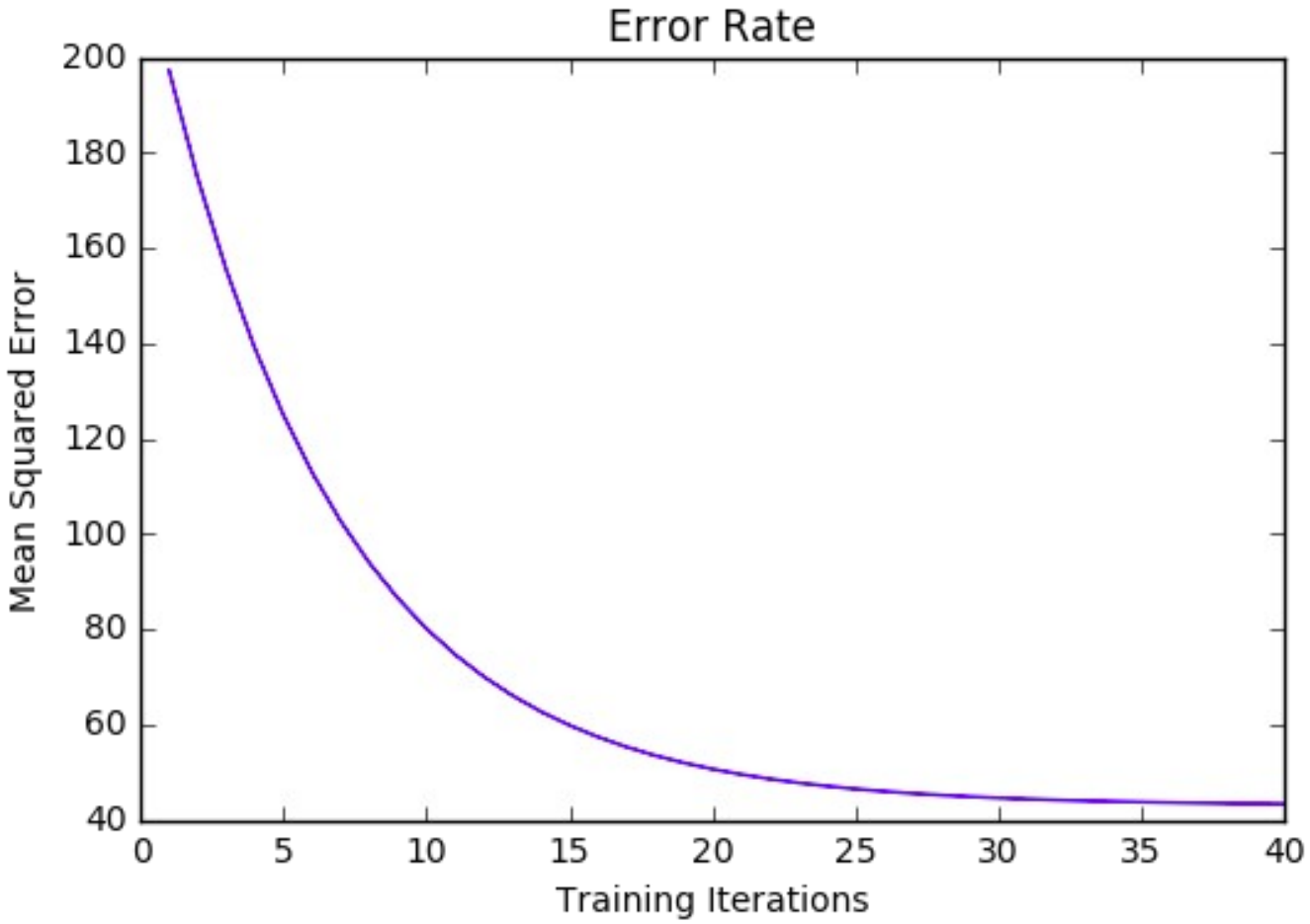
Visualizing



Visualizing



Cost History



Multiple Regression

- Multiple linear regression is used to estimate the relationship between two or more independent variables and one dependent variable. You can use multiple linear regression when you want to know:
- How strong the relationship is between two or more independent variables and one dependent variable (e.g. how rainfall, temperature, and amount of fertilizer added affect crop growth).
- The value of the dependent variable at a certain value of the independent variables (e.g. the expected yield of a crop at certain levels of rainfall, temperature, and fertilizer addition).

Multiple Regression

$$y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \varepsilon$$

- y = the predicted value of the dependent variable
- B_0 = the y-intercept (value of y when all other parameters are set to 0)
- B_1X_1= the regression coefficient (B_1) of the first independent variable (X_1) (a.k.a. the effect that increasing the value of the independent variable has on the predicted y value)
- … = do the same for however many independent variables you are testing
- B_nX_n = the regression coefficient of the last independent variable
- e = model error (a.k.a. how much variation there is in our estimate of y)

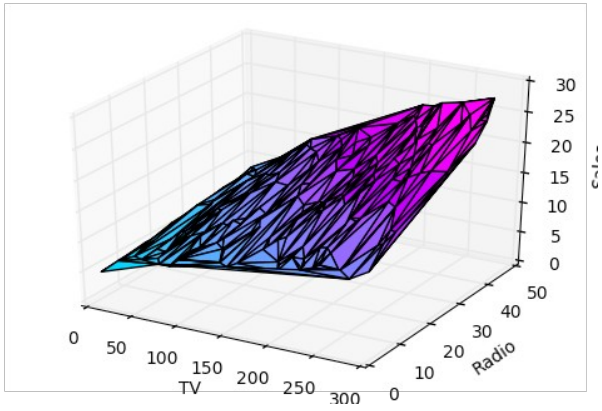
Multiple Regression

- Let’s say we are given data on TV, radio, and newspaper advertising spend for a list of companies, and our goal is to predict sales in terms of units sold.

Company	TV	Radio	News	Units
Amazon	230.1	37.8	69.1	22.1
Google	44.5	39.3	23.1	10.4
Facebook	17.2	45.9	34.7	18.3
Apple	151.5	41.3	13.2	18.5

Growing Complexity

- As the number of features grows, the complexity of our model increases and it becomes increasingly difficult to visualize, or even comprehend, our data.



- One solution is to break the data apart and compare 1-2 features at a time. In this example we explore how Radio and TV investment impacts Sales.

Normalization

- Real world dataset contains features that highly vary in magnitudes, units, and range.
- Normalisation should be performed when the scale of a feature is irrelevant or misleading and not should Normalise when the the scale is meaningful.
- The algorithms which use Euclidean Distance measure are sensitive to Magnitudes. Here feature scaling helps to weigh all the features equally.
- Formally, If a feature in the dataset is big in scale compared to others then in algorithms where Euclidean distance is measured this big scaled feature becomes dominating and needs to be normalized.

Normalization

- Techniques:
  - Feature Scaling or Standardization
  - Min Max Scaling
  - Robust Scaler

Feature Scaling

- Feature Scaling or Standardization: It is a step of Data Pre-Processing which is applied to independent variables or features of data.
- It basically helps to normalise the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.
- Package Used:
  - sklearn.preprocessing
- Import:
  - from sklearn.preprocessing import StandardScaler

Feature Scaling

- Formula used in Backend
  - Standardisation replaces the values by their Z scores.

$$z = \frac{x - \mu}{\sigma} \qquad \frac{x_i - \text{mean}(x)}{sd(x)}$$

  - Mostly the Fit method is used for Feature scaling.

Standard Scaler

- The idea behind StandardScaler is that it will transform your data such that its distribution will have a mean value 0 and standard deviation of 1.
- Given the distribution of the data, each value in the dataset will have the sample mean value subtracted, and then divided by the standard deviation of the whole dataset.

Standardization

- Standardization assumes that your data has a Gaussian (bell curve) distribution.
- This does not strictly have to be true, but the technique is more effective if your attribute distribution is Gaussian.
- Standardization is useful when your data has varying scales and the algorithm you are using does make assumptions about your data having a Gaussian distribution, such as linear regression, logistic regression, and linear discriminant analysis.

Min-Max Scaling Normalization

- Here your data Z is rescaled such that any specific z will now be $0 \leq z \leq 1$. and is done through this formula:

$$z = \frac{x \text{ - } \min(x)}{\max(x) \text{ - } \min(x)}$$

- It refers to rescaling real valued numeric attributes into the range 0 and 1.
- It is useful to scale the input attributes for a model that relies on the magnitude of values, such as distance measures used in k-nearest neighbors and in the preparation of coefficients in regression.

Min-Max Scaling Normalization

- Normalization is a good technique to use when you do not know the distribution of your data or when you know the distribution is not Gaussian (a bell curve).
- Normalization is useful when your data has varying scales and the algorithm you are using does not make assumptions about the distribution of your data, such as k-nearest neighbors and artificial neural networks.

Robust Scaler

- Robust Scaler algorithms scale features that are robust to outliers.
- The method it follows is almost similar to the MinMax Scaler but it uses the interquartile range (rather than the min-max used in MinMax Scaler).
- The median and scales of the data are removed by this scaling algorithm according to the quantile range.
- It, thus, follows the following formula:

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

- Where Q1 is the 1st quartile, and Q3 is the third quartile.

Making Predictions

- Our predict function outputs an estimate of sales given our current weights (coefficients) and a company's TV, radio, and newspaper spend. Our model will try to identify weight values that most reduce our cost function.

$$Sales = W_1 TV + W_2 Radio + W_3 Newspaper$$

Initialize Weights

```
W1 = 0.0
W2 = 0.0
W3 = 0.0
weights = np.array([
    [W1],
    [W2],
    [W3]
])
```

Code

```
def predict(features, weights):
    **
    features - (200, 3)
    weights - (3, 1)
    predictions - (200,1)
    **
    predictions = np.dot(features, weights)
    return predictions
```

Cost Function

- Now we need a cost function to audit how our model is performing.
- The math is the same, except we swap the mx+b expression for $W_1x_1+W_2x_2+W_3x_3$.
- We also divide the expression by 2 to make derivative calculations simpler.

$$MSE = \frac{1}{2N}\sum_{i=1}^{n}(y_i - (W_1x_1 + W_2x_2 + W_3x_3))^2$$

Cost Function

```
def cost_function(features, targets, weights):
    features:(200,3)
    targets: (200,1)
    weights:(3,1)
    returns average squared error among predictions
    N = len(targets)
    predictions = predict(features, weights)
    # Matrix math lets use do this without looping
    sq_error = (predictions - targets)**2
    # Return average squared error among predictions
    return 1.0/(2*N) * sq_error.sum()
```

Gradient descent

- Again using the Chain rule we can compute the gradient–a vector of partial derivatives describing the slope of the cost function for each weight.

$$f'(W_1) = -x_1(y-(W_1x_1+W_2x_2+W_3x_3))$$

$$f'(W_2) = -x_2(y-(W_1x_1+W_2x_2+W_3x_3))$$

$$f'(W_3) = -x_3(y-(W_1x_1+W_2x_2+W_3x_3))$$

Gradient descent

```
def update_weights(features, targets, weights, lr):
    '''
    Features:(200, 3)     Targets: (200, 1)     Weights:(3, 1)
    '''
    predictions = predict(features, weights)
    #Extract our features
    x1 = features[:,0]
    x2 = features[:,1]
    x3 = features[:,2]
    # Use matrix cross product (*) to simultaneously
    # calculate the derivative for each weight
    d_w1 = -x1*(targets - predictions)
    d_w2 = -x2*(targets - predictions)
    d_w3 = -x3*(targets - predictions)
    # Multiply the mean derivative by the learning rate
    # and subtract from our weights (remember gradient points in direction of steepest ASCENT)
    weights[0][0] -= (lr * np.mean(d_w1))
    weights[1][0] -= (lr * np.mean(d_w2))
    weights[2][0] -= (lr * np.mean(d_w3))
    return weights
```

Simplifying with matrices

- The gradient descent code above has a lot of duplication. Can we improve it somehow? One way to refactor would be to loop through our features and weights–allowing our function to handle any number of features. However there is another even better technique: vectorized gradient descent.
- Math
  - We use the same formula as above, but instead of operating on a single feature at a time, we use matrix multiplication to operative on all features and weights simultaneously. We replace the x_i terms with a single feature matrix X.

    gradient=–X(targets–predictions)

Bias Term

- Our train function is the same as for simple linear regression, however we're going to make one final tweak before running: add a bias term to our feature matrix.
- In our example, it's very unlikely that sales would be zero if companies stopped advertising.
- Possible reasons for this might include past advertising, existing customer relationships, retail locations, and salespeople.
- A bias term will help us capture this base case.

Bias Term

- Code
  - Below we add a constant 1 to our features matrix. By setting this value to 1, it turns our bias term into a constant.

    bias = np.ones(shape=(len(features),1))

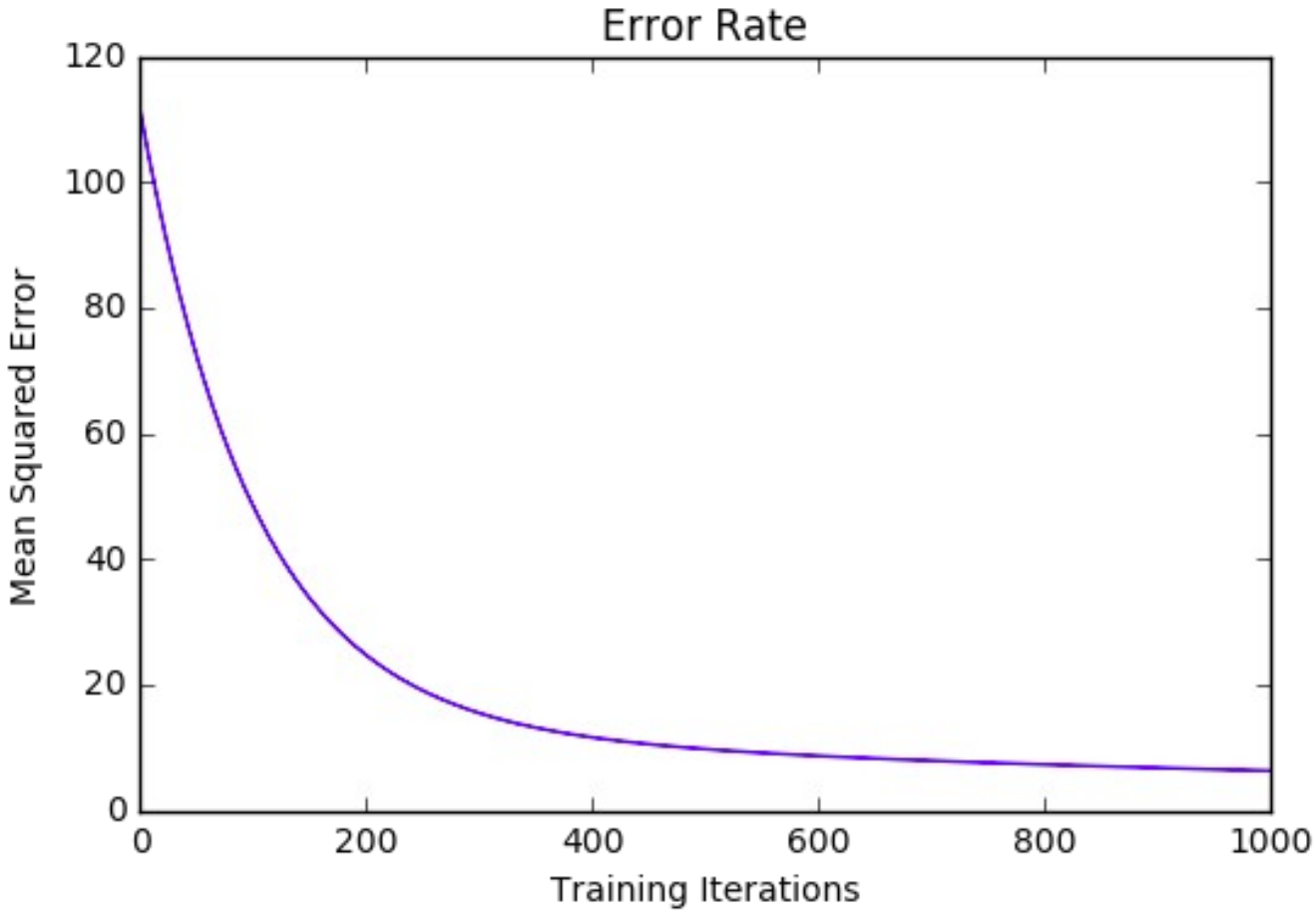
    features = np.append(bias, features, axis=1)

Model Evaluation

- After training our model through 1000 iterations with a learning rate of .0005, we finally arrive at a set of weights we can use to make predictions:

  Sales=4.7TV+3.5Radio+.81Newspaper+13.9
- Our MSE cost dropped from 110.86 to 6.25.

Model Evaluation



Useful web resources

- www.mitu.co.in
- www.scikit-learn.org
- www.towardsdatascience.com
- www.medium.com
- www.analyticsvidhya.com
- www.kaggle.com
- www.stephacking.com
- www.github.com

Thank you

This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License

@mitu_skillologies

/mITuSkillologies

@mitu_group

/company/mitu-skillologies

MITUSkillologies

Web Resources
https://mitu.co.in
http://tusharkute.com

contact@mitu.co.in

tushar@tusharkute.com