**Machine Learning Process**

1. Data Preprocessing
   1. Import the data
   2. Clean the data
   3. Split the data into training and test sets (80% training 20% test sets)
2. Modelling
   1. Build the model
   2. Train the model
   3. Make predictions
3. Evaluation
   1. Calculate performance metrics
   2. Make a verdict

**Feature Scaling**

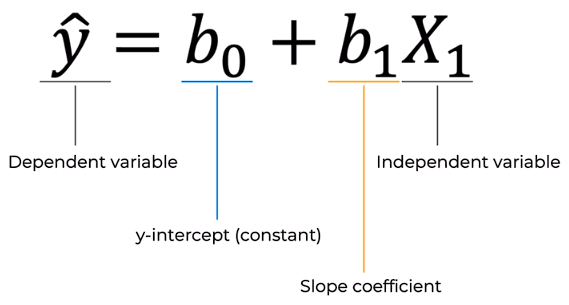
* Always applied to columns
* Normalization = ; end up with values between 0 and 1
* Standardization = ; end up with most values between -3 and 3 except extreme values

**Data Preprocessing**

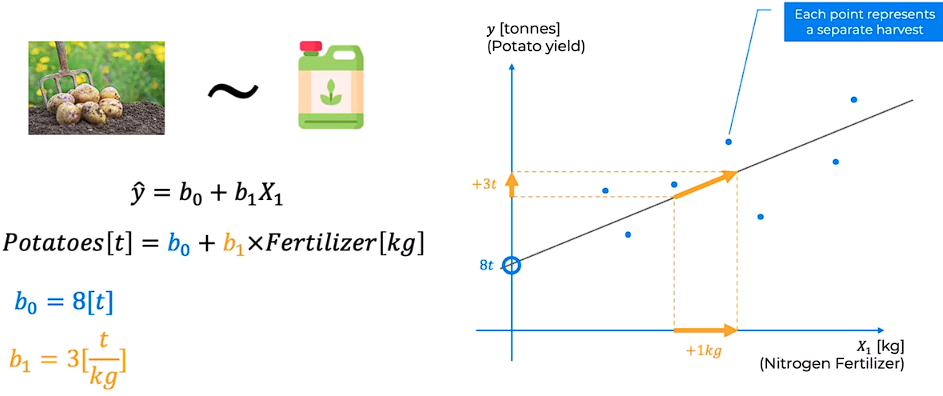
|  |
| --- |
| # ## Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # ## Importing the dataset  dataset = pd.read\_csv("Data.csv")  X = dataset.iloc[:, :-1].values # rows: all rows so ":" and column except the last one so ":-1"  Y = dataset.iloc[:, -1].values # rows all and column the last one  print(X)  print(Y)  # ## Taking care of missing data  from sklearn.impute import SimpleImputer  imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')  imputer.fit(X[:, 1:3])  X[:, 1:3] = imputer.transform(X[:, 1:3])  print(X)  # ## Encoding categorical data  # ### Encoding the Independent Variable  '''  Encoding country with vectors, These vectors are also called Dummy Variables  France: <1 0 0>  Spain: <0 0 1>  Germamy: <0 1 0>  '''  from sklearn.compose import ColumnTransformer  from sklearn.preprocessing import OneHotEncoder  ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [0])] , remainder='passthrough')  X = np.array(ct.fit\_transform(X))  print(X)  # ### Encoding the Dependent Variable  # Encoding yes: 1 no: 0  from sklearn.preprocessing import LabelEncoder  le=LabelEncoder()  Y = le.fit\_transform(Y)  print(Y)  # ## Splitting the dataset into the Training set and Test set  '''  Q:  \*\*Which is performed first splitting data set or feature scaling??  Ans:    #1 test set is supposed to be new data with new observation for ML model. So, split data set first then apply feature scaling          #2 this prevents information leakage  '''  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=1)  print(X\_train)  print(X\_test)  print(Y\_train)  print(Y\_test)  # ## Feature Scaling  '''  Q: Why is feature scaling done?  Ans: To prevent some features from dominating the ML model than other features  Q: What to choose Standardization or Normalization?  Ans: Standardization will work on all types of data       NormaliZation will work on data having normal distribution  Q: Do we need to apply Standardization on Dummy Variables?  Ans: No, they already have values 0 or 1 which is b/w -3 and +3. It we do it we will loose the fact which country represents which vector  '''  from sklearn.preprocessing import StandardScaler  sc = StandardScaler()  # fit() method will calculate mean and SD  # transform() methos will calculate the value using the formula (X - Mean(X))/SD(X)  X\_train[:, 3:] = sc.fit\_transform(X\_train[:, 3:])  X\_test[:, 3:] = sc.transform(X\_test[:, 3:])  print(X\_train)  print(X\_test) |

**REGRESSION**

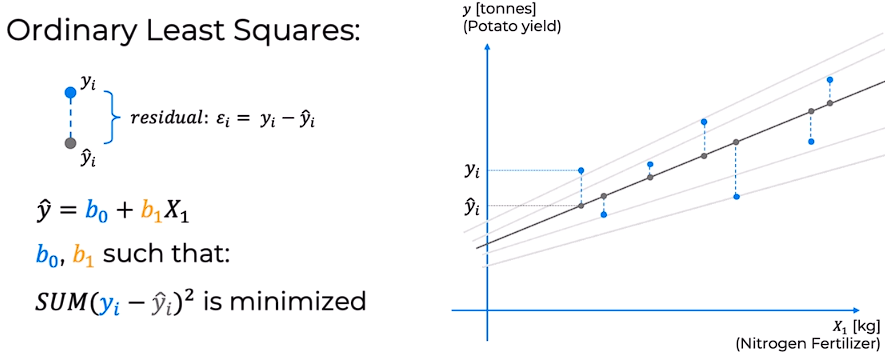
**Simple Linear Regression**

****

t=tonnes of potatoes, kg=kg of fertilizers

****

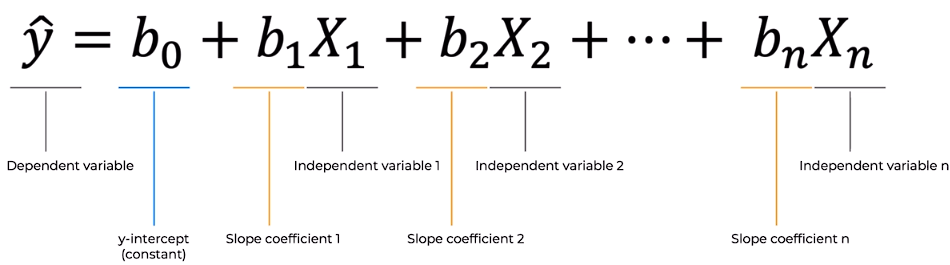
To find the best regression line we use **Ordinary Least Squares** method:

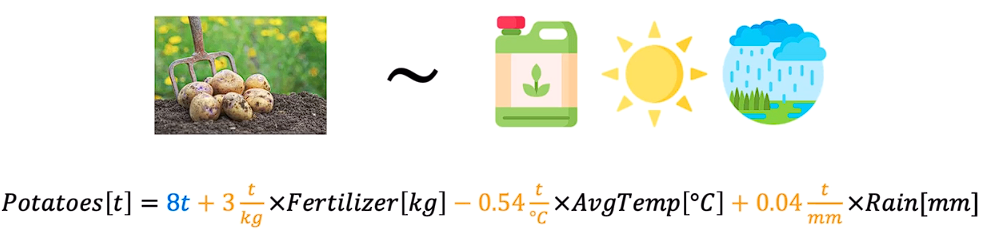


The line for which sum is minimized, is the best regression line

|  |
| --- |
| # # Simple Linear Regression  # ## Importing the libraries  import numpy as np  import matplotlib.pyplot as plt  import pandas as pd  # ## Importing the dataset  dataset = pd.read\_csv('Salary\_Data.csv')  X = dataset.iloc[:, :-1].values  y = dataset.iloc[:, -1].values  # ## Splitting the dataset into the Training set and Test set  from sklearn.model\_selection import train\_test\_split  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)  # ## Training the Simple Linear Regression model on the Training set  from sklearn.linear\_model import LinearRegression  regressor = LinearRegression()  regressor.fit(X\_train, y\_train)  # ## Predicting the Test set results  y\_pred = regressor.predict(X\_test)  # ## Visualising the Training set results  plt.scatter(X\_train, y\_train, color="red")  plt.plot(X\_train, regressor.predict(X\_train), color="blue")  plt.title("Salary vs Experience (Training Set)")  plt.xlabel("Years of Experience")  plt.ylabel("Salary")  plt.show()  # ## Visualising the Test set results  plt.scatter(X\_test, y\_test, color="red")  plt.plot(X\_test, y\_pred, color="blue")  plt.title("Salary vs Experience (Test Set)")  plt.xlabel("Years of Experience")  plt.ylabel("Salary")  plt.show() |

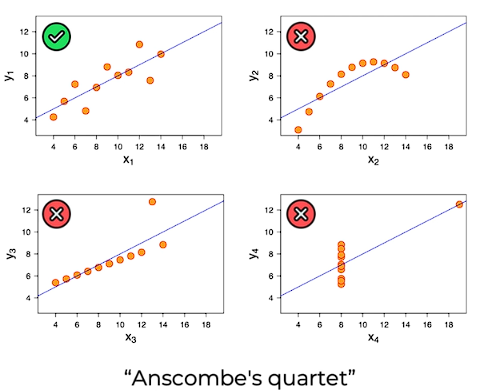
**Multiple Linear Regression**

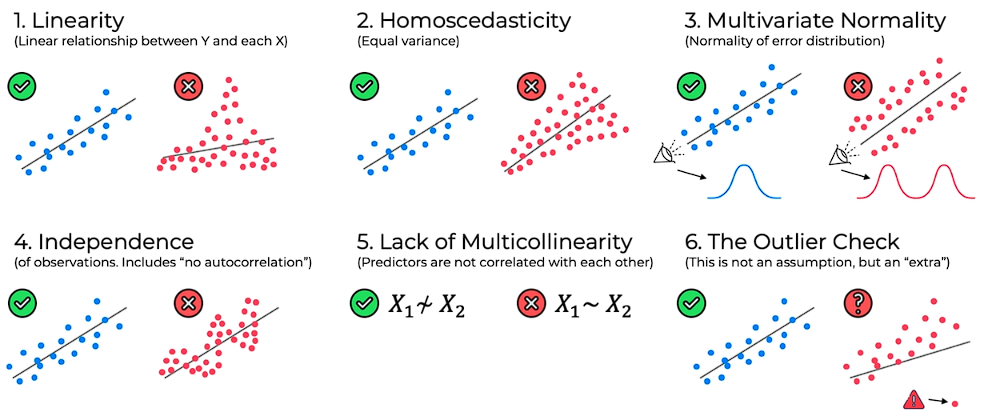




Research Paper of Potato Harvest: <https://www.mdpi.com/2073-4395/11/5/885>

**Assumptions of Linear Regression**

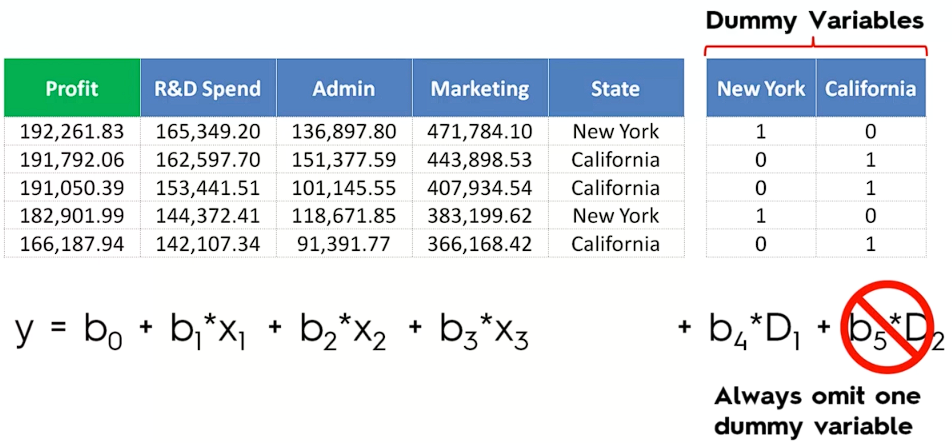




2. No cone shaped

4. No correlation between previous rows (eg. in Stock Market)

**Dummy Variable Trap**



* Dummy Variable are used for categorical data like states, what type of industry, etc.
* We can’t have a const and all the dummy variables in our regression so emit 1 dummy variable, i.e., it we have 8 dummy variables consider 7 in regression eq.

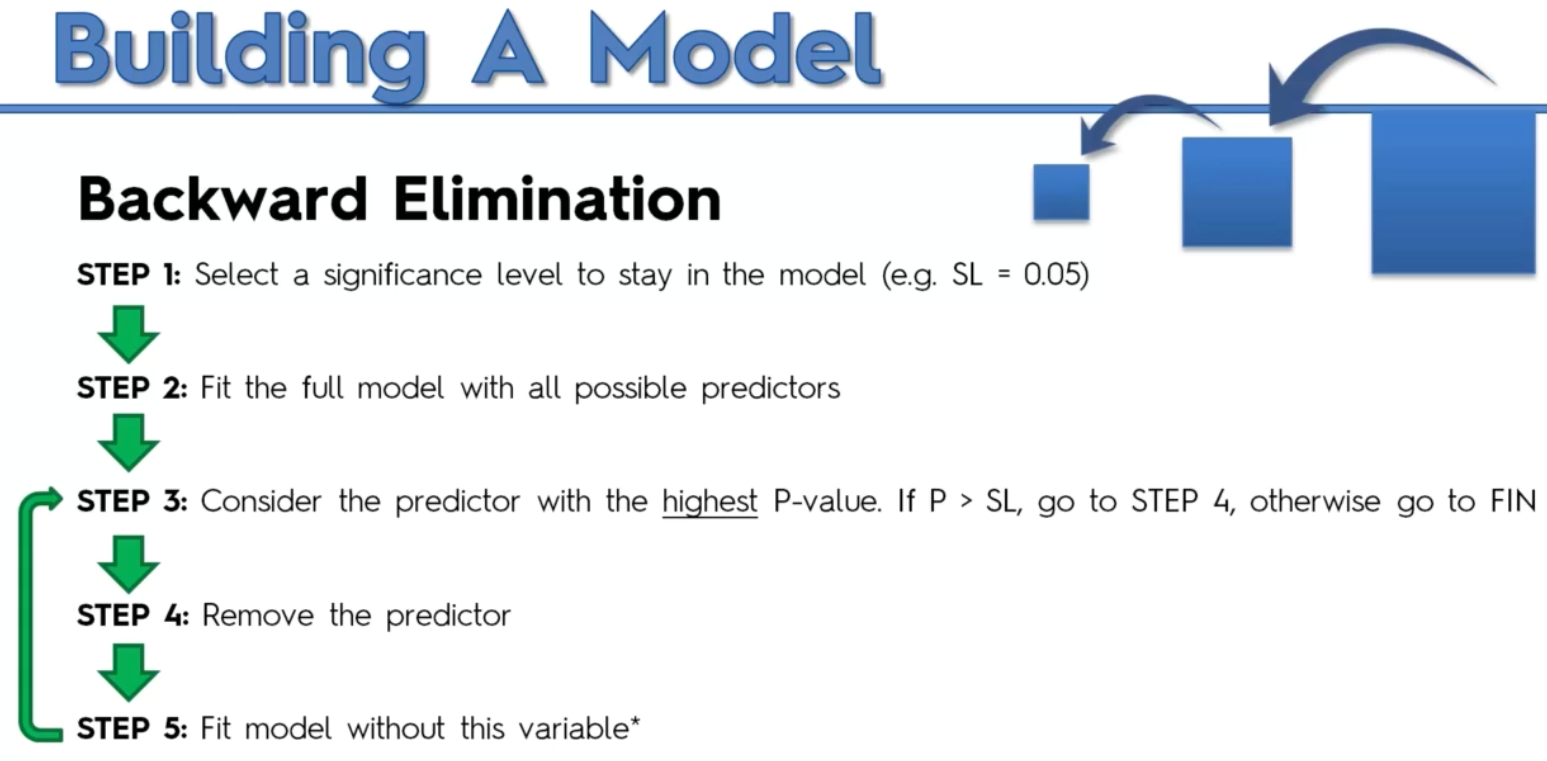
**Building A Model**

1. All-in

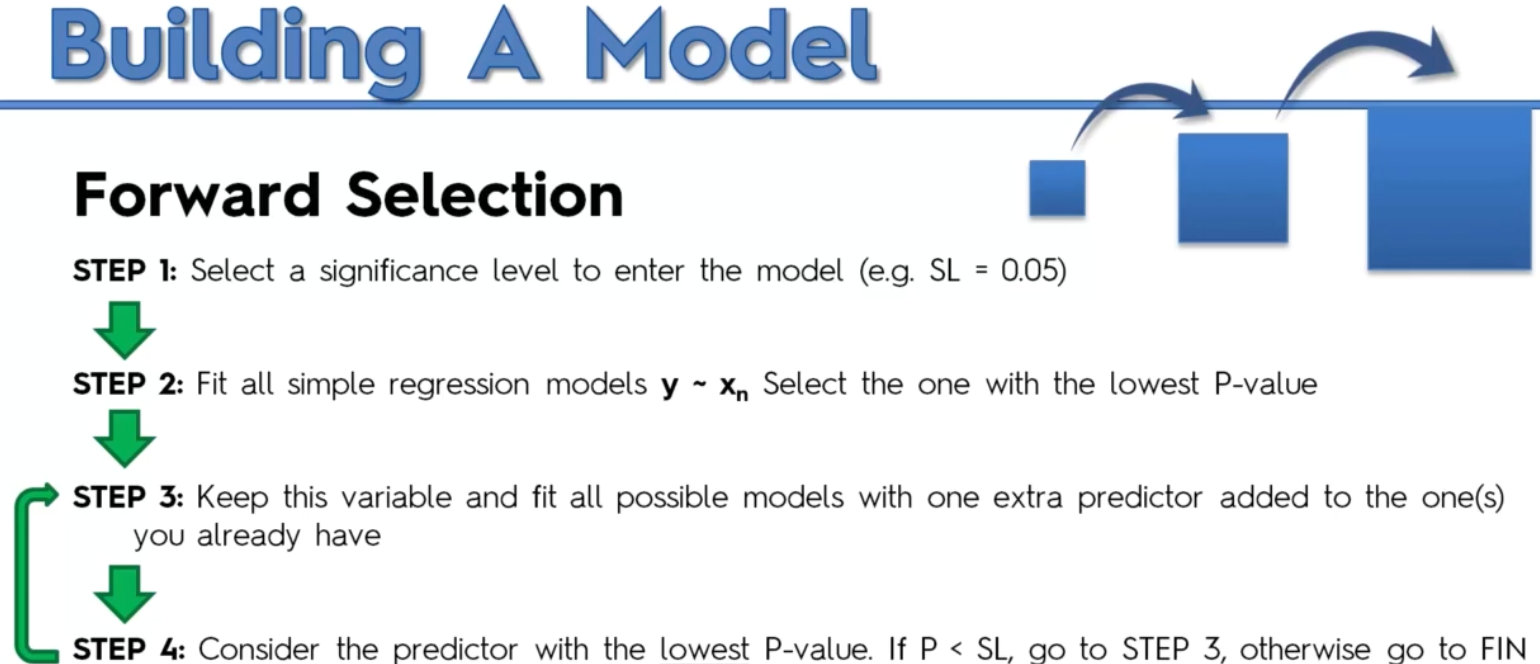
All-in cases:

* Prior-knowledge
* You have to select all variables
* Preparing for backward elimination

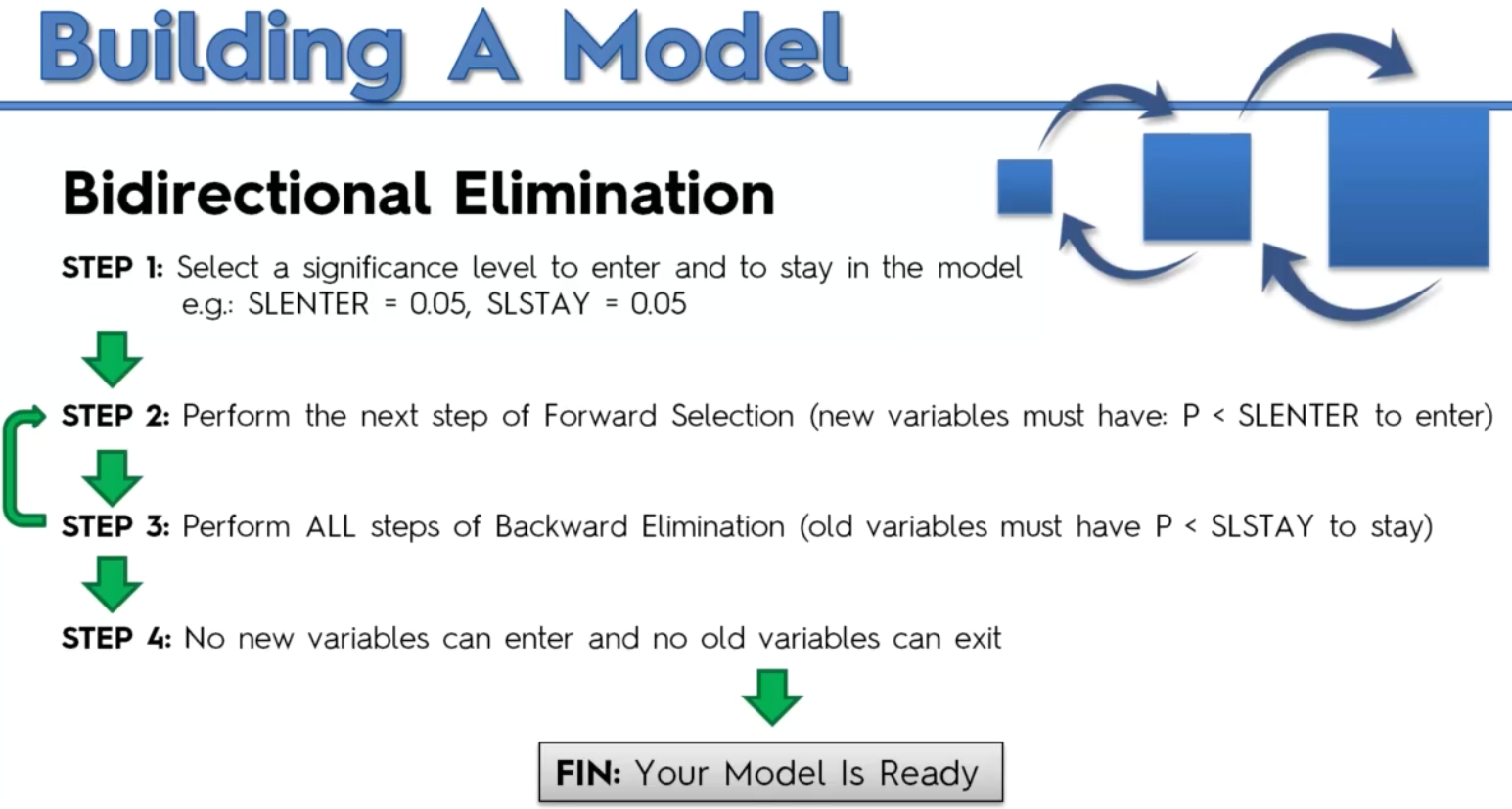
1. Backward Elimination
2. Forward Selection Stepwise regression
3. Bidirectional Elimination
4. Score Comparison

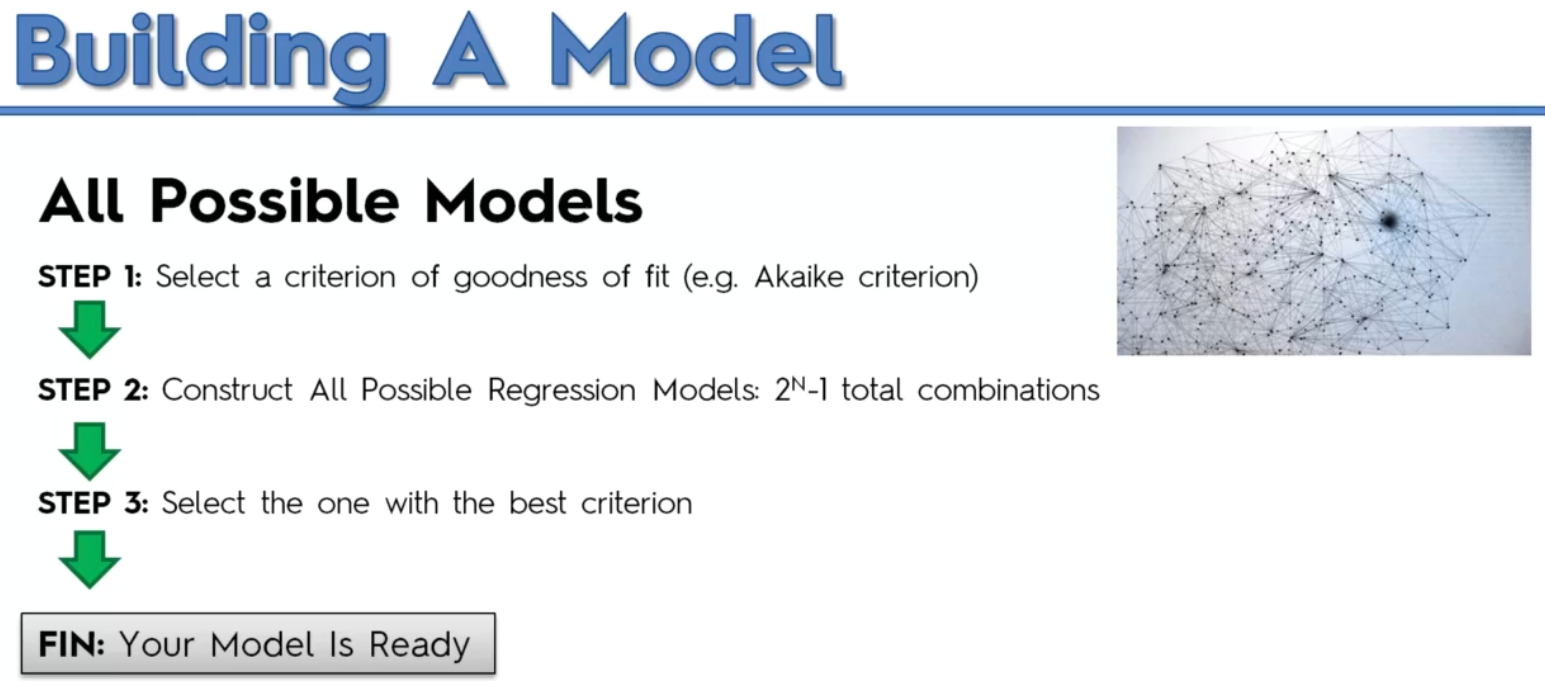


FIN (finish) means model is ready



FIN: keep the previous model not the current





\*\*In Multiple Linear Regression, we don’t need to apply feature scaling as the coefficients will compensate for it.

1. # # Multiple Linear Regression

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset = pd.read\_csv("50\_Startups.csv")

11. X = dataset.iloc[:, :-1].values

12. Y = dataset.iloc[:, -1].values

13.

14. print(X)

15.

16. print(Y)

17.

18.

19. # ## Encoding categorical data

20. from sklearn.compose import ColumnTransformer

21. from sklearn.preprocessing import OneHotEncoder

22. ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])] , remainder='passthrough')

23. X = np.array(ct.fit\_transform(X))

24.

25. print(X)

26.

27.

28. # ## Splitting the dataset into the Training set and Test set

29. from sklearn.model\_selection import train\_test\_split

30. X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.2, random\_state=1)

31.

32.

33. # ## Training the Multiple Linear Regression model on the Training set

34. from sklearn.linear\_model import LinearRegression

35. # this class will automatically avoid dummy variable trap

36. # it will automatically select the features with highest p-value

37. regressor = LinearRegression()

38. regressor.fit(X\_train, Y\_train)

39.

40.

41. # ## Predicting the Test set results

42. y\_pred = regressor.predict(X\_test)

43. np.set\_printoptions(precision=2)

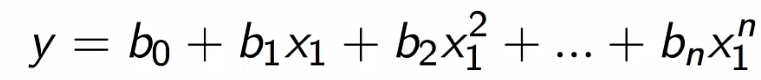
44. print(np.concatenate((y\_pred.reshape(len(y\_pred), 1), Y\_test.reshape(len(Y\_test), 1)), axis=1))

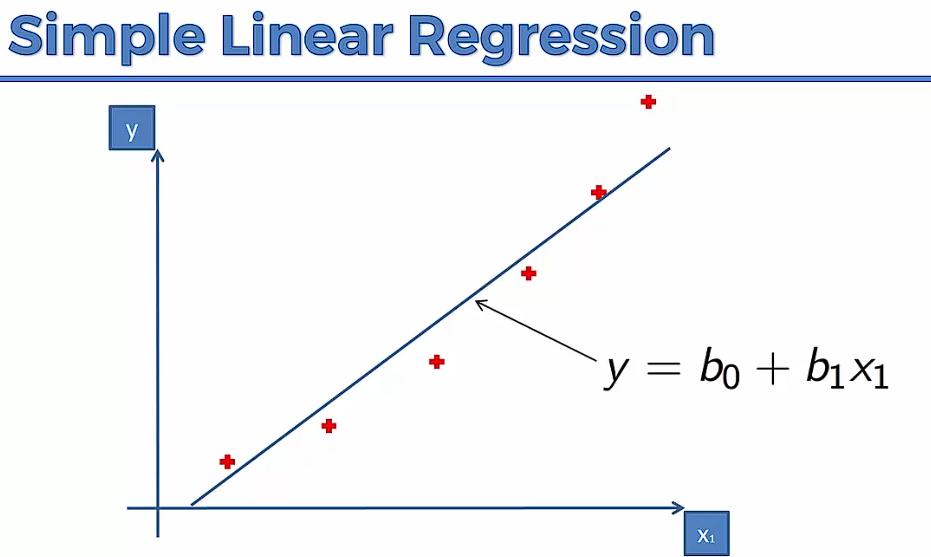
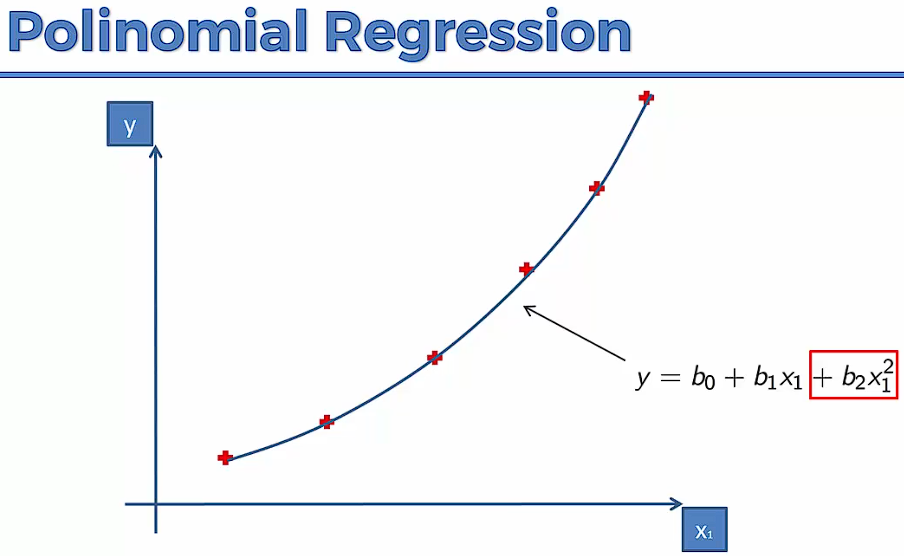
45.

46. plt.plot(y\_pred, color="blue")

47. plt.plot(Y\_test, color="red")

**Polynomial Linear Regression**



Use Cases – How diseases/epidemics spread in a territory

Why Linear – Coefficients are Linear, Linear and Non-linear are considered with the coefficients.

1. # # Polynomial Regression

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset = pd.read\_csv('Position\_Salaries.csv')

11. X = dataset.iloc[:, 1:-1].values

12. y = dataset.iloc[:, -1].values

13.

14. print(X)

15.

16. print(y)

17.

18.

19. # ## Training the Linear Regression model on the whole dataset

20. from sklearn.linear\_model import LinearRegression

21. lin\_reg = LinearRegression()

22. lin\_reg.fit(X, y)

23.

24.

25. # ## Training the Polynomial Regression model on the whole dataset

26. from sklearn.preprocessing import PolynomialFeatures

27. poly\_reg = PolynomialFeatures(degree=4)

28. X\_poly = poly\_reg.fit\_transform(X)

29.

30. print(X\_poly)

31.

32. lin\_reg\_2 = LinearRegression()

33. lin\_reg\_2.fit(X\_poly, y)

34.

35.

36. # ## Visualising the Linear Regression results

37. plt.scatter(X,y, color="red")

38. plt.plot(X, lin\_reg.predict(X), color="blue")

39. plt.xlabel("Position Level")

40. plt.ylabel("Salary")

41. plt.title("Truth or Bluff (Linear Regression)")

42. plt.show()

43.

44.

45. # ## Visualising the Polynomial Regression results

46. plt.scatter(X,y, color="red")

47. plt.plot(X, lin\_reg\_2.predict(poly\_reg.fit\_transform(X)), color="blue")

48. plt.xlabel("Position Level")

49. plt.ylabel("Salary")

50. plt.title("Truth or Bluff (Polynomial Regression)")

51. plt.show()

52.

53.

54. # ## Visualising the Polynomial Regression results (for higher resolution and smoother curve)

55. X\_grid = np.arange(min(X), max(X), 0.01)

56. X\_grid=X\_grid.reshape((len(X\_grid),1))

57. plt.scatter(X,y, color="red")

58. plt.plot(X\_grid, lin\_reg\_2.predict(poly\_reg.fit\_transform(X\_grid)), color="blue")

59. plt.xlabel("Position Level")

60. plt.ylabel("Salary")

61. plt.title("Truth or Bluff (Polynomial Regression)")

62. plt.show()

63.

64.

65. # ## Predicting a new result with Linear Regression

66. lin\_reg.predict([[6.5]]) # bad prediction 330k

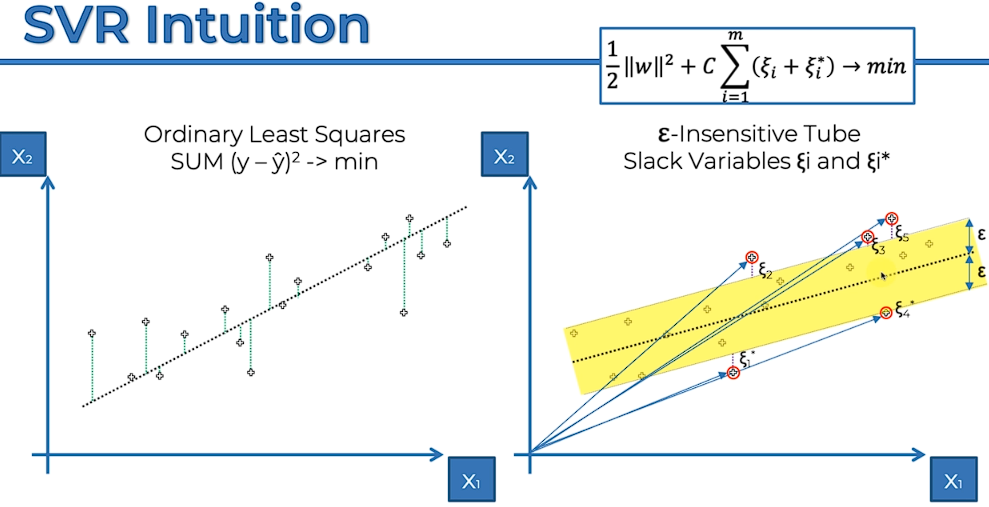
67.

68.

69. # ## Predicting a new result with Polynomial Regression

70. lin\_reg\_2.predict(poly\_reg.fit\_transform([[6.5]])) # good prediction 160k

**Support Vector Regression (SVR)**



It is a **Linear SVR Model**

The points within the tube are ignored

The points outside the tube for the support vectors which dictate the size of the tube.

1. # # Support Vector Regression (SVR)

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset = pd.read\_csv('Position\_Salaries.csv')

11. X = dataset.iloc[:, 1:-1].values

12. y = dataset.iloc[:, -1].values

13.

14. print(X)

15.

16. print(y)

17.

18. y = y.reshape(len(y),1) # standard scaling wants 2D array

19.

20. print(y)

21.

22.

23. # ## Feature Scaling

24. from sklearn.preprocessing import StandardScaler

25. sc\_X=StandardScaler()

26. sc\_y=StandardScaler()

27. X = sc\_X.fit\_transform(X)

28. y = sc\_y.fit\_transform(y)

29.

30. print(X)

31.

32. print(y)

33.

34.

35. # ## Training the SVR model on the whole dataset

36. from sklearn.svm import SVR

37. regressor = SVR(kernel = 'rbf')

38. regressor.fit(X, y)

39.

40.

41. # ## Predicting a new result

42. sc\_y.inverse\_transform(regressor.predict(sc\_X.transform([[6.5]])).reshape(-1,1))

43.

44.

45. # ## Visualising the SVR results

46. plt.scatter(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(y), color='red')

47. plt.plot(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(regressor.predict(X).reshape(-1,1)), color="blue")

48. plt.title('Truth or Bluff (SVR)')

49. plt.xlabel('Position level')

50. plt.ylabel('Salary')

51. plt.show()

52.

53.

54. # ## Visualising the SVR results (for higher resolution and smoother curve)

55. X\_grid = np.arange(min(sc\_X.inverse\_transform(X)), max(sc\_X.inverse\_transform(X)), 0.01)

56. X\_grid = X\_grid.reshape((len(X\_grid), 1))

57. plt.scatter(sc\_X.inverse\_transform(X), sc\_y.inverse\_transform(y), color = 'red')

58. plt.plot(X\_grid, sc\_y.inverse\_transform(regressor.predict(sc\_X.transform(X\_grid)).reshape(-1,1)), color = 'blue')

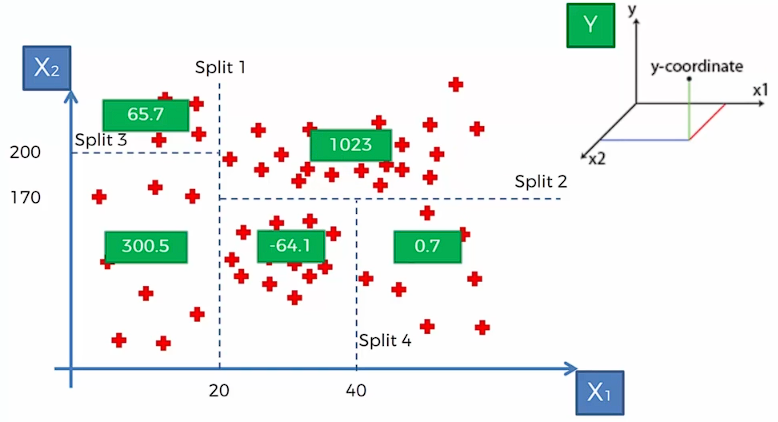
59. plt.title('Truth or Bluff (SVR)')

60. plt.xlabel('Position level')

61. plt.ylabel('Salary')

62. plt.show()

**Decision Tree Regression**



X1, X2 are independent variables, Y is dependent variable sticking out from the screen.

The averages of all the terminal leaves are taken (different for different segment), and for a given (X1, X2) Y value gives the average of the segment it falls under.

1. # # Decision Tree Regression

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset = pd.read\_csv('Position\_Salaries.csv')

11. x = dataset.iloc[:, 1:-1].values

12. y = dataset.iloc[:, -1].values

13.

14.

15. # ## Training the Decision Tree Regression model on the whole dataset

16. from sklearn.tree import DecisionTreeRegressor

17. regressor = DecisionTreeRegressor(random\_state=0)

18. regressor.fit(x, y)

19.

20.

21. # ## Predicting a new result

22. regressor.predict([[6.5]])

23.

24.

25. # ## Visualising the Decision Tree Regression results (higher resolution)

26. x\_grid=np.arange(min(x), max(x), 0.1)

27. x\_grid=x\_grid.reshape((len(x\_grid),1))

28. plt.scatter(x,y, color="red")

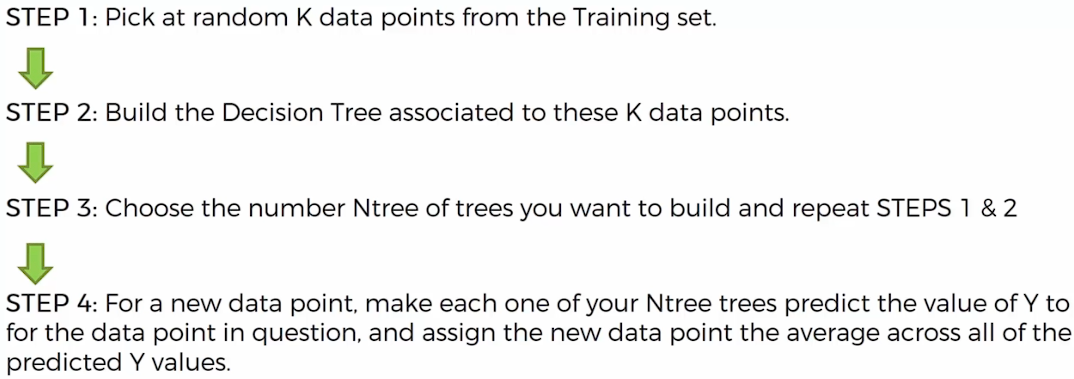
29. plt.plot(x\_grid, regressor.predict(x\_grid))

30. plt.show() # not recommended for 2D i.e single independent and single dependent variable

**Random Forest Regression**

Based on Ensemble Learning

***Ensemble Learning*** *– When we take multiple algo or the same algo multiple times to put together something more powerful than the original one.*



1. # Random Forest Regression

2.

3. # Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8. # Importing the dataset

9. dataset = pd.read\_csv('Position\_Salaries.csv')

10. X = dataset.iloc[:, 1:-1].values

11. y = dataset.iloc[:, -1].values

12.

13. # Training the Random Forest Regression model on the whole dataset

14. from sklearn.ensemble import RandomForestRegressor

15. regressor = RandomForestRegressor(n\_estimators = 10, random\_state = 0)

16. regressor.fit(X, y)

17.

18. # Predicting a new result

19. regressor.predict([[6.5]])

20.

21. # Visualising the Random Forest Regression results (higher resolution)

22. X\_grid = np.arange(min(X), max(X), 0.01)

23. X\_grid = X\_grid.reshape((len(X\_grid), 1))

24. plt.scatter(X, y, color = 'red')

25. plt.plot(X\_grid, regressor.predict(X\_grid), color = 'blue')

26. plt.title('Truth or Bluff (Random Forest Regression)')

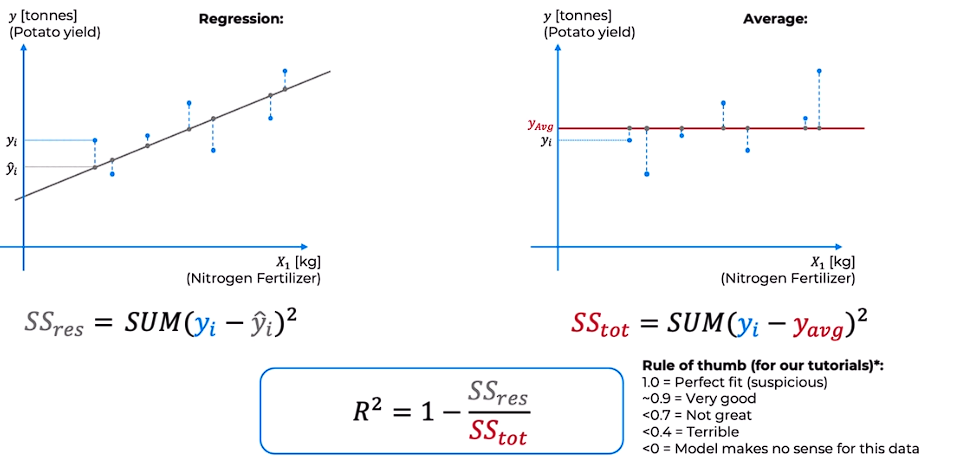
27. plt.xlabel('Position level')

28. plt.ylabel('Salary')

29. plt.show()

**Evaluating Regression Model Performance**

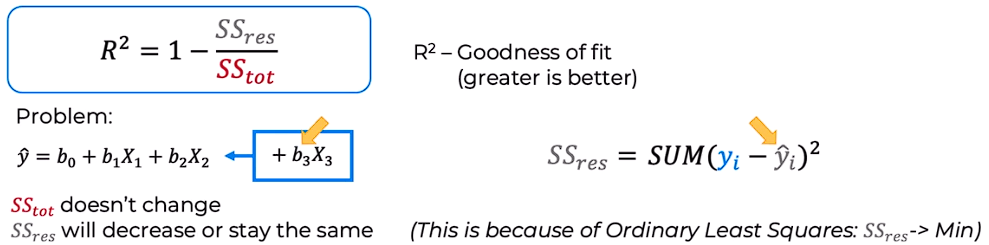
**R Squared**



SSres = Residual Sum of Squares, SStot = Total Sum of Squares

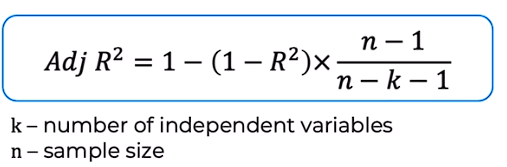
Generally, SSres < SStot

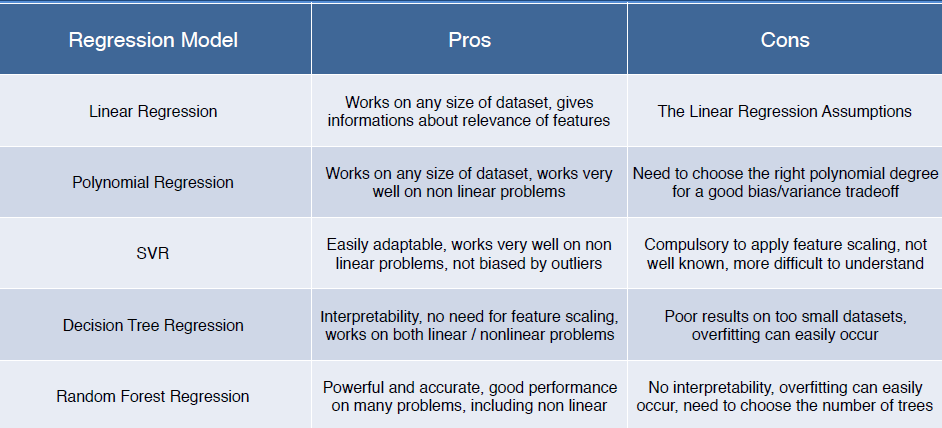
**Adjusted R Squared**



Situation arises that we keep adding new variables but SSres decreases every so slightly or never decrease at all (it can never increase) that is nothing to do with or model.

**Solution –** Formula that will penalize for every new independent variable added.





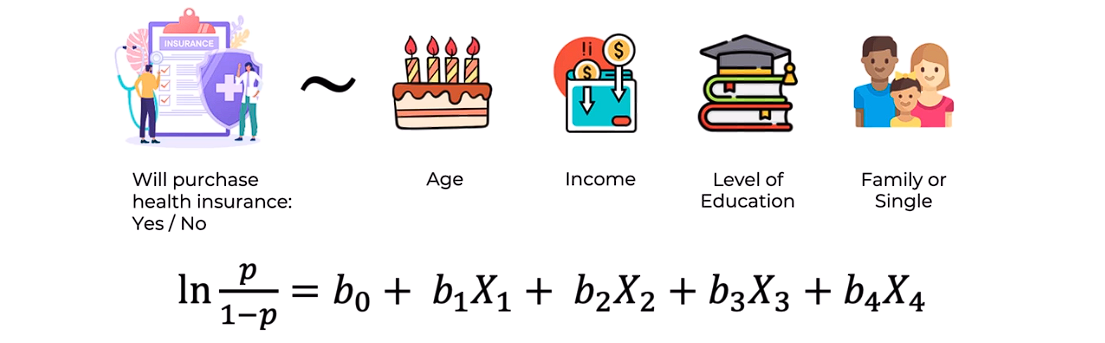
**CLASSIFICATION**

*A supervised learning algorithm to identify the category of new observation based on training data.*

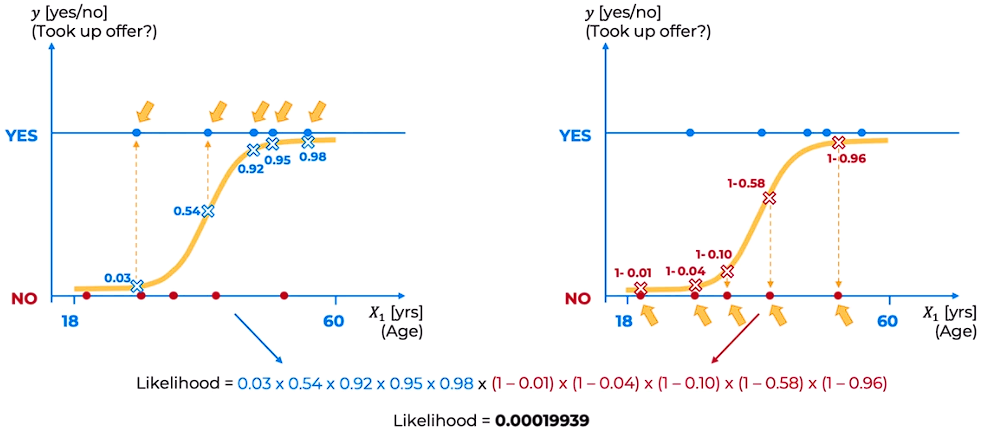
**Logistic Regression**



We can also have multiple independent variables



* **Maximum Likelihood**

****

Check maximum likelihood of each curve. The curve with the maximum likelihood is the best curve.

1. # # Logistic Regression

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset=pd.read\_csv('Social\_Network\_Ads.csv')

11. x=dataset.iloc[:,:-1].values

12. y=dataset.iloc[:,-1].values

13.

14. print(x)

15.

16. print(y)

17.

18.

19. # ## Splitting the dataset into the Training set and Test set

20. from sklearn.model\_selection import train\_test\_split

21. x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = 0.1, random\_state = 0)

22.

23. print(x\_train)

24.

25. print(y\_train)

26.

27. print(x\_test)

28.

29. print(y\_test)

30.

31.

32. # ## Feature Scaling

33. from sklearn.preprocessing import StandardScaler

34. sc=StandardScaler()

35. x\_train = sc.fit\_transform(x\_train)

36. x\_test=sc.fit\_transform(x\_test)

37.

38. print(x\_train)

39.

40. print(x\_test)

41.

42.

43. # ## Training the Logistic Regression model on the Training set

44. from sklearn.linear\_model import LogisticRegression

45. classifier = LogisticRegression(random\_state=1)

46. classifier.fit(x\_train, y\_train)

47.

48.

49. # ## Predicting a new result

50. print(classifier.predict(sc.transform([[30,87000]])))

51.

52.

53. # ## Predicting the Test set results

54. y\_pred = classifier.predict(x\_test)

55. print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

56.

57.

58. # ## Making the Confusion Matrix

59. # shows exactly the no. of correct predictions and no. of incorrect predictions

60. # it is a 2D matrix

61. # used to calculate accuracy

62. from sklearn.metrics import confusion\_matrix, accuracy\_score

63. cm=confusion\_matrix(y\_true=y\_test, y\_pred=y\_pred)

64. print(cm)

65. accuracy\_score(y\_true=y\_test, y\_pred=y\_pred)

66.

67.

68. # ## Visualising the Training set results

69. from matplotlib.colors import ListedColormap

70. X\_set, y\_set = sc.inverse\_transform(x\_train), y\_train

71. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

72. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

73. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

74. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

75. plt.xlim(X1.min(), X1.max())

76. plt.ylim(X2.min(), X2.max())

77. for i, j in enumerate(np.unique(y\_set)):

78. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

79. plt.title('Logistic Regression (Training set)')

80. plt.xlabel('Age')

81. plt.ylabel('Estimated Salary')

82. plt.legend()

83. plt.show()

84.

85.

86. # ## Visualising the Test set results

87. X\_set, y\_set = sc.inverse\_transform(x\_test), y\_test

88. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

89. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

90. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

91. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

92. plt.xlim(X1.min(), X1.max())

93. plt.ylim(X2.min(), X2.max())

94. for i, j in enumerate(np.unique(y\_set)):

95. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

96. plt.title('Logistic Regression (Test set)')

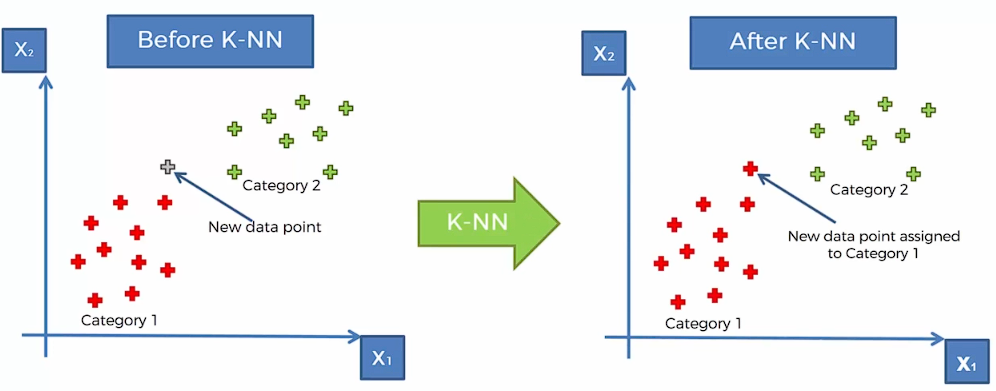
97. plt.xlabel('Age')

98. plt.ylabel('Estimated Salary')

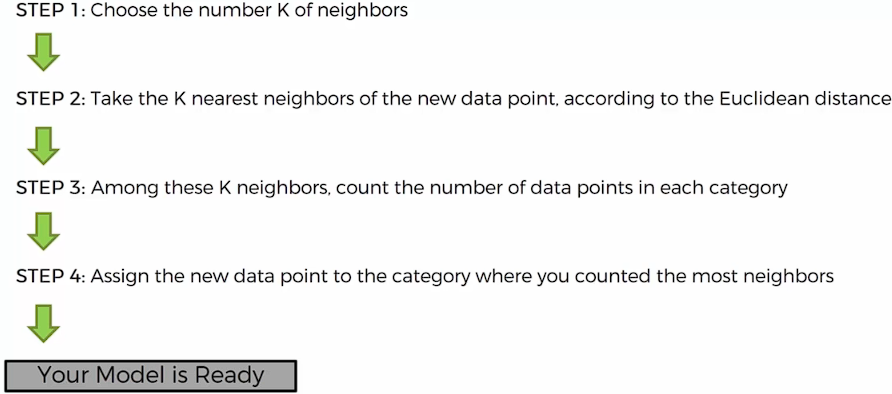
99. plt.legend()

100. plt.show()

**K-NN (K – Nearest Neighbour)**



Algorithm:



*Generally take k=5 neighbours*

1. # # K-Nearest Neighbors (K-NN)

2.

3. # ## Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8.

9. # ## Importing the dataset

10. dataset = pd.read\_csv('Social\_Network\_Ads.csv')

11. X = dataset.iloc[:, :-1].values

12. y = dataset.iloc[:, -1].values

13.

14.

15. # ## Splitting the dataset into the Training set and Test set

16. from sklearn.model\_selection import train\_test\_split

17. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

18.

19. print(X\_train)

20.

21. print(y\_train)

22.

23. print(X\_test)

24.

25. print(y\_test)

26.

27.

28. # ## Feature Scaling

29. from sklearn.preprocessing import StandardScaler

30. sc = StandardScaler()

31. X\_train = sc.fit\_transform(X\_train)

32. X\_test = sc.transform(X\_test)

33.

34. print(X\_train)

35.

36. print(X\_test)

37.

38.

39. # ## Training the K-NN model on the Training set

40. from sklearn.neighbors import KNeighborsClassifier

41. classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

42. classifier.fit(X\_train, y\_train)

43.

44.

45. # ## Predicting a new result

46. print(classifier.predict(sc.transform([[30,87000]])))

47.

48.

49. # ## Predicting the Test set results

50. y\_pred = classifier.predict(X\_test)

51. print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

52.

53.

54. # ## Making the Confusion Matrix

55. from sklearn.metrics import confusion\_matrix, accuracy\_score

56. cm = confusion\_matrix(y\_test, y\_pred)

57. print(cm)

58. accuracy\_score(y\_test, y\_pred)

59.

60.

61. # ## Visualising the Training set results

62. from matplotlib.colors import ListedColormap

63. X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

64. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

65. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

66. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

67. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

68. plt.xlim(X1.min(), X1.max())

69. plt.ylim(X2.min(), X2.max())

70. for i, j in enumerate(np.unique(y\_set)):

71. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

72. plt.title('K-NN (Training set)')

73. plt.xlabel('Age')

74. plt.ylabel('Estimated Salary')

75. plt.legend()

76. plt.show()

77.

78.

79. # ## Visualising the Test set results

80. from matplotlib.colors import ListedColormap

81. X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

82. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 1),

83. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 1))

84. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

85. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

86. plt.xlim(X1.min(), X1.max())

87. plt.ylim(X2.min(), X2.max())

88. for i, j in enumerate(np.unique(y\_set)):

89. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

90. plt.title('K-NN (Test set)')

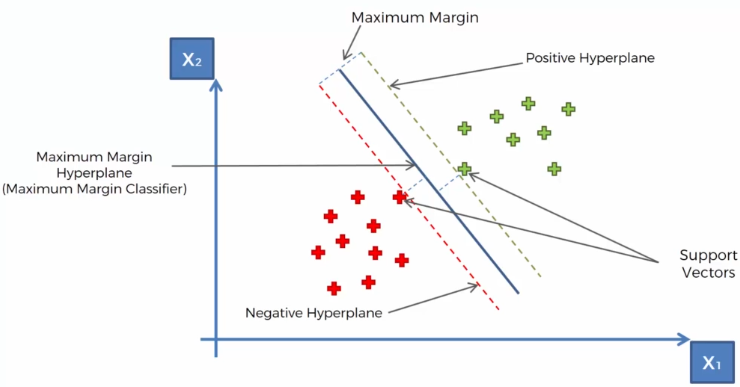
91. plt.xlabel('Age')

92. plt.ylabel('Estimated Salary')

93. plt.legend()

94. plt.show()

**Support Vector Machines (SVM)**



*The sum of distances between the* ***support vector*** *points and the line should be maximum*

1. # Support Vector Machine (SVM)

2.

3. # Importing the libraries

4. import numpy as np

5. import matplotlib.pyplot as plt

6. import pandas as pd

7.

8. # Importing the dataset

9. dataset = pd.read\_csv('Social\_Network\_Ads.csv')

10. X = dataset.iloc[:, :-1].values

11. y = dataset.iloc[:, -1].values

12.

13. # Splitting the dataset into the Training set and Test set

14. from sklearn.model\_selection import train\_test\_split

15. X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

16. print(X\_train)

17. print(y\_train)

18. print(X\_test)

19. print(y\_test)

20.

21. # Feature Scaling

22. from sklearn.preprocessing import StandardScaler

23. sc = StandardScaler()

24. X\_train = sc.fit\_transform(X\_train)

25. X\_test = sc.transform(X\_test)

26. print(X\_train)

27. print(X\_test)

28.

29. # Training the SVM model on the Training set

30. from sklearn.svm import SVC

31. classifier = SVC(kernel = 'linear', random\_state = 0)

32. classifier.fit(X\_train, y\_train)

33.

34. # Predicting a new result

35. print(classifier.predict(sc.transform([[30,87000]])))

36.

37. # Predicting the Test set results

38. y\_pred = classifier.predict(X\_test)

39. print(np.concatenate((y\_pred.reshape(len(y\_pred),1), y\_test.reshape(len(y\_test),1)),1))

40.

41. # Making the Confusion Matrix

42. from sklearn.metrics import confusion\_matrix, accuracy\_score

43. cm = confusion\_matrix(y\_test, y\_pred)

44. print(cm)

45. accuracy\_score(y\_test, y\_pred)

46.

47. # Visualising the Training set results

48. from matplotlib.colors import ListedColormap

49. X\_set, y\_set = sc.inverse\_transform(X\_train), y\_train

50. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

51. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

52. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

53. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

54. plt.xlim(X1.min(), X1.max())

55. plt.ylim(X2.min(), X2.max())

56. for i, j in enumerate(np.unique(y\_set)):

57. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

58. plt.title('SVM (Training set)')

59. plt.xlabel('Age')

60. plt.ylabel('Estimated Salary')

61. plt.legend()

62. plt.show()

63.

64. # Visualising the Test set results

65. from matplotlib.colors import ListedColormap

66. X\_set, y\_set = sc.inverse\_transform(X\_test), y\_test

67. X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 10, stop = X\_set[:, 0].max() + 10, step = 0.25),

68. np.arange(start = X\_set[:, 1].min() - 1000, stop = X\_set[:, 1].max() + 1000, step = 0.25))

69. plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).reshape(X1.shape),

70. alpha = 0.75, cmap = ListedColormap(('red', 'green')))

71. plt.xlim(X1.min(), X1.max())

72. plt.ylim(X2.min(), X2.max())

73. for i, j in enumerate(np.unique(y\_set)):

74. plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1], c = ListedColormap(('red', 'green'))(i), label = j)

75. plt.title('SVM (Test set)')

76. plt.xlabel('Age')

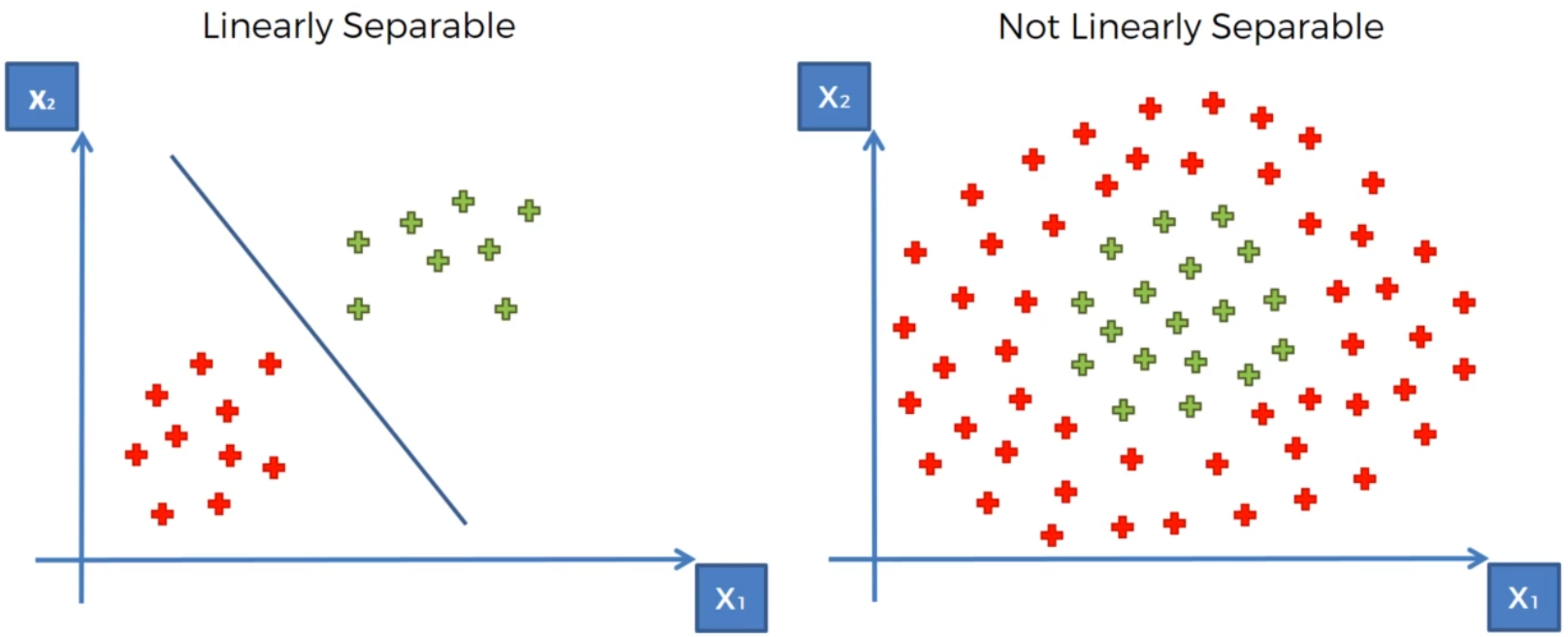
77. plt.ylabel('Estimated Salary')

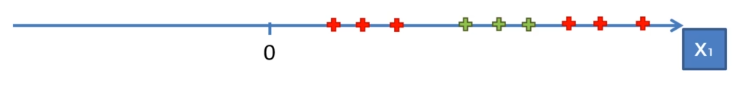
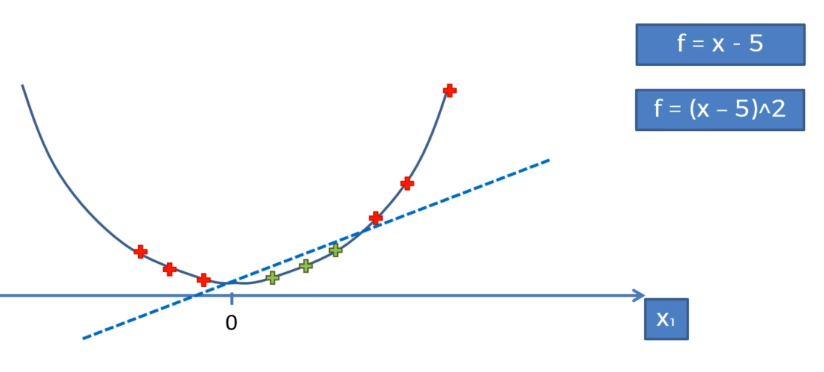
78. plt.legend()

79. plt.show()

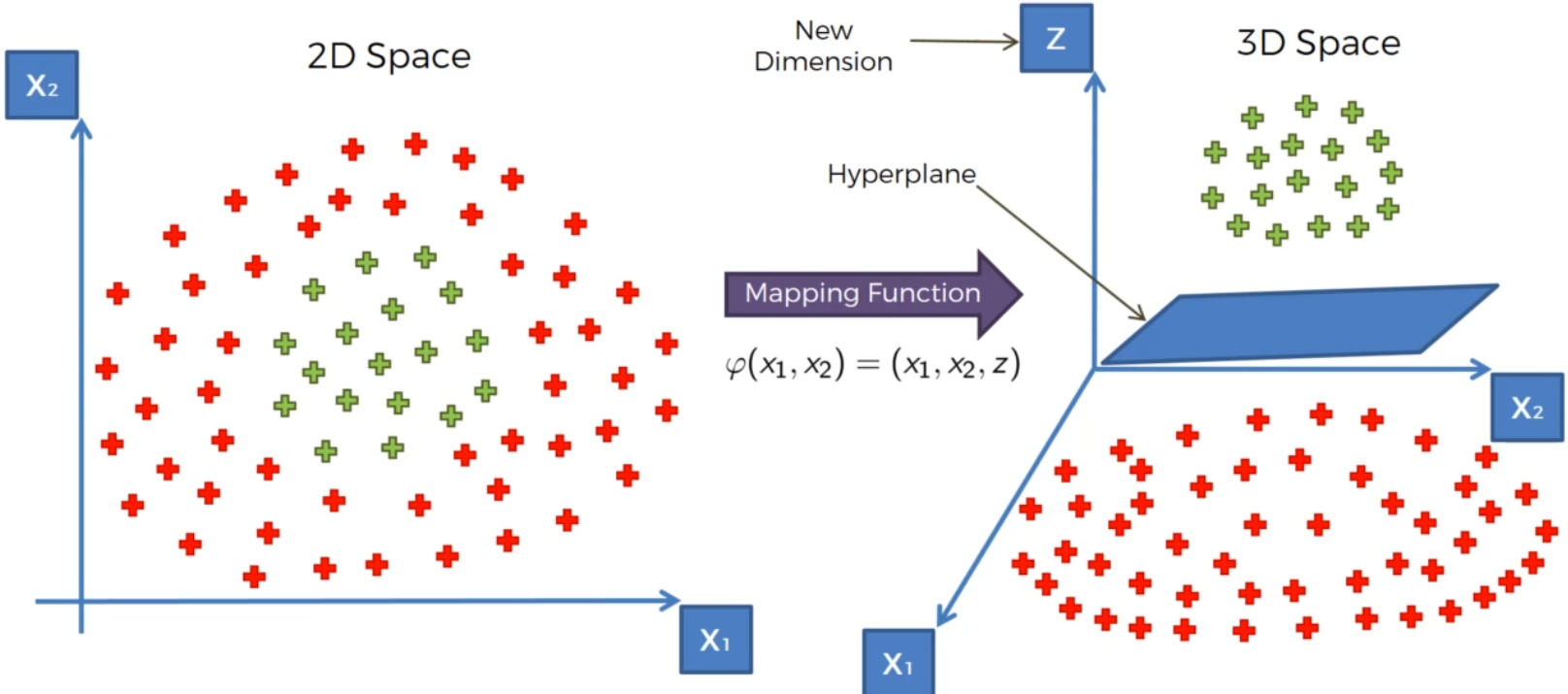
**Kerner SVM**

*Used when data is not linearly separable(Assuption)*



**Solution 1:** Using a higer dimensional space to apply linear separation 

*Conversion of 1D to 2D dataset by applying the mapping func to apply linear separation.*

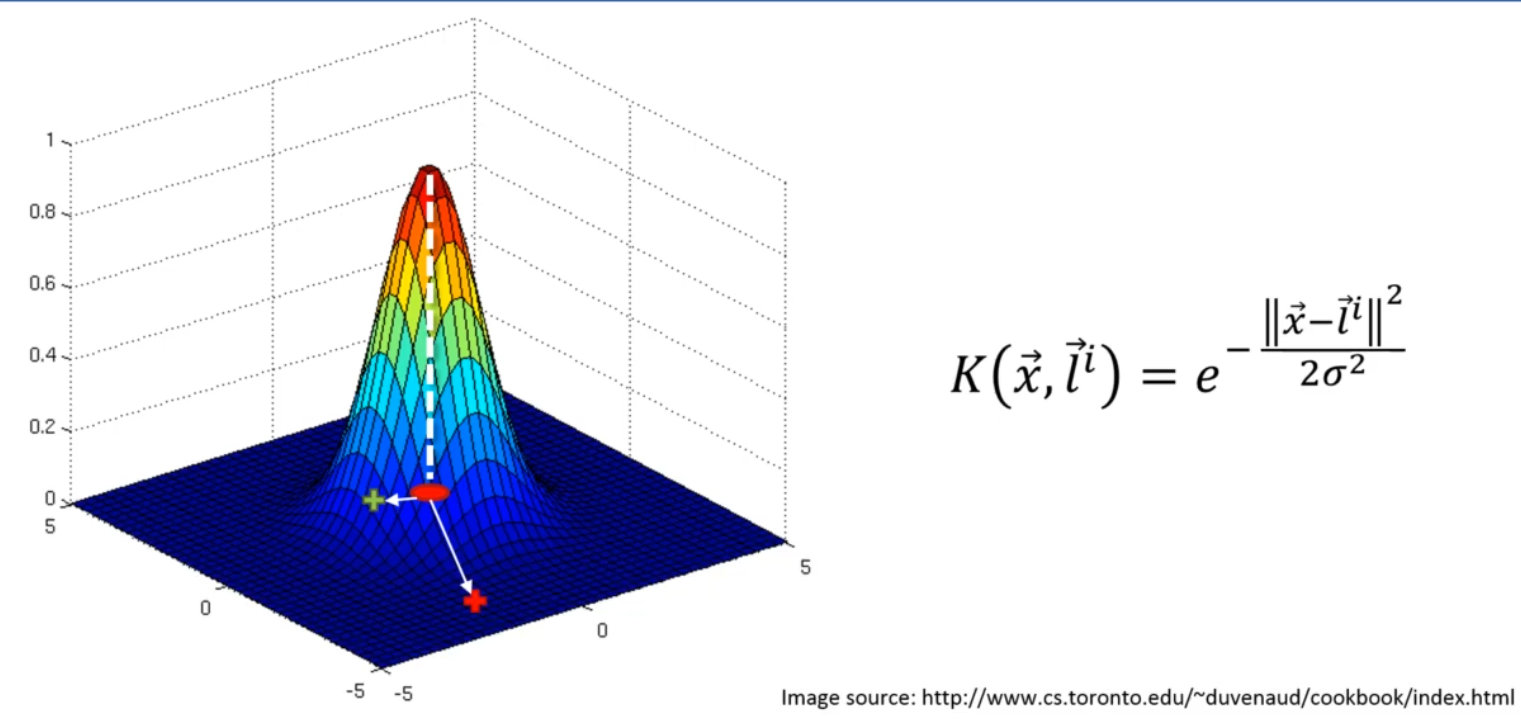


*Similar conversion from 2D space to 3D space*

*However it can be* ***compute intensive.***

**Note:** Linear separator in a 1D plane is a point, 2D plane is a line, in 3D plane a hyperplane.

**Solution 2:** Using kernel SVM

****

Gaussian RBF Kernel

***x*** *denotes a random point*

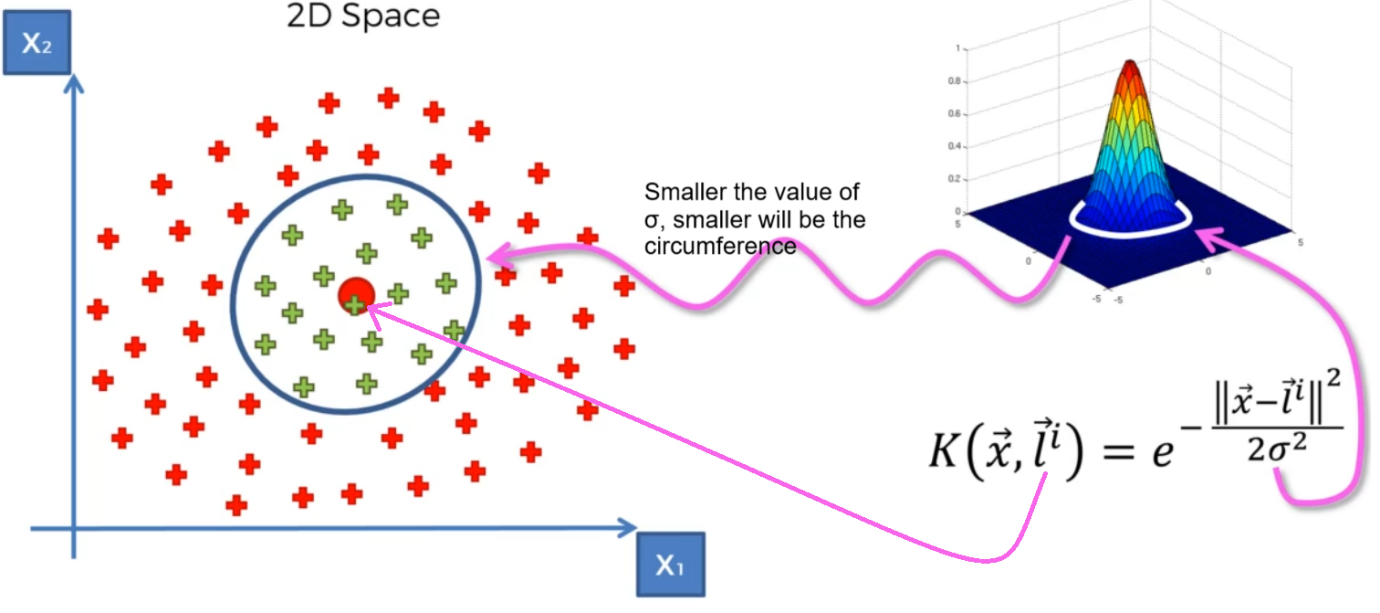
***l*** *denotes position of landmark,* ***li*** *means there can be I no. of landmarks*

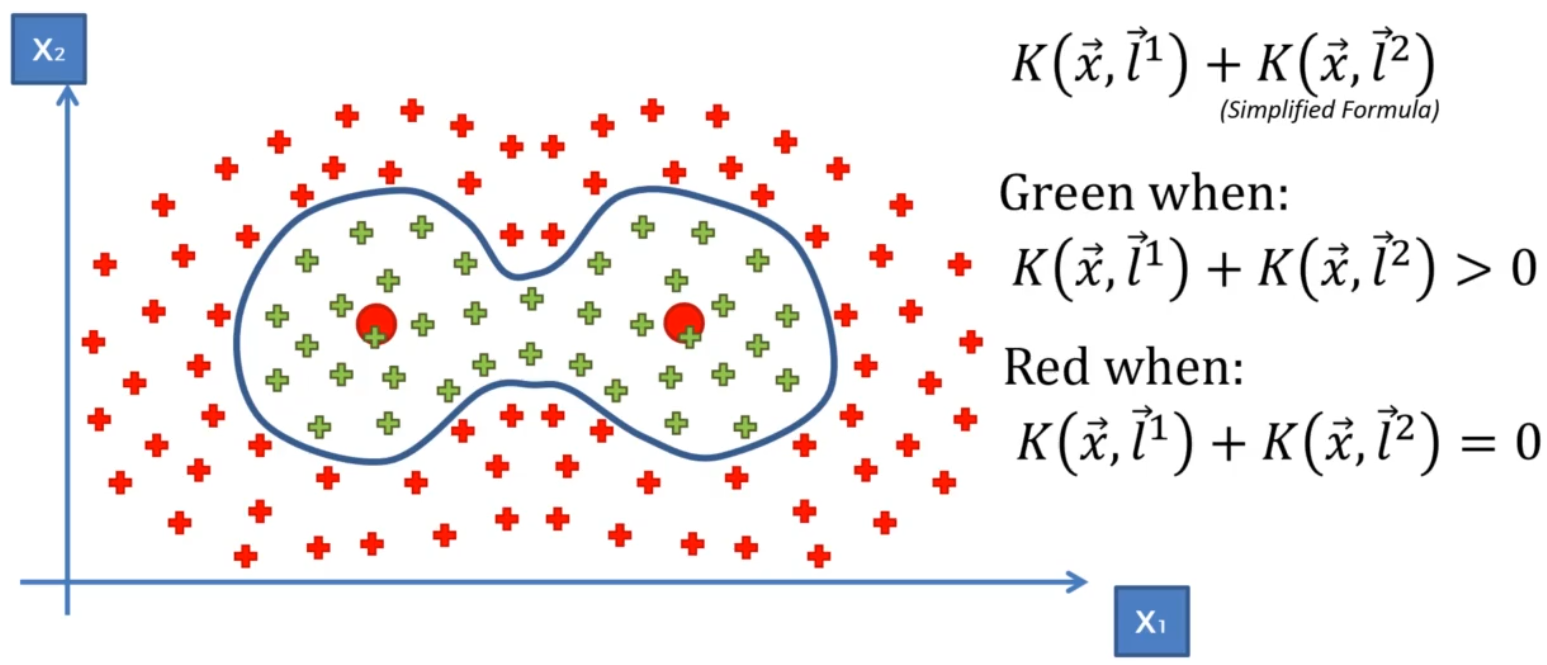
***σ*** *denotes a parameter*

***||x- li||2*** *denotes (distance of the point from the landmark)2*

***Points that are far from landmark will have vertical axis value close to 0.***

***Points that are near to landmark will have vertical axis value close to 1.***





Multiple Landmarks