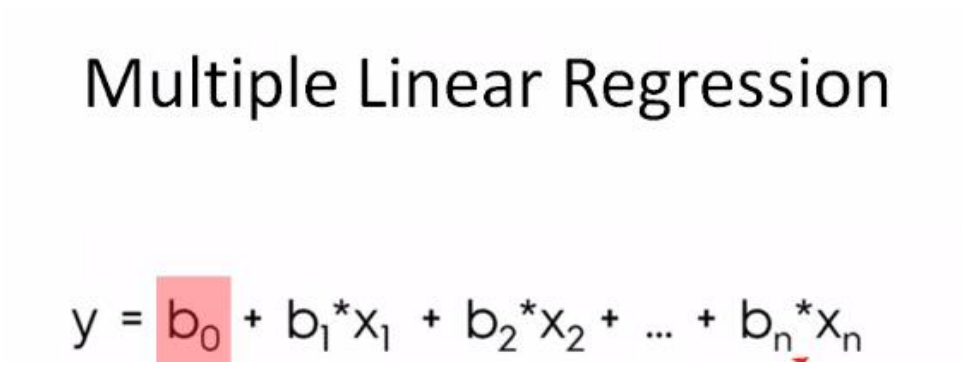
Data Science Training - Day 3

Multi-Linear Regression:

A	B	C	D	E
RNDSpend	Administration	MarketingSpend	State	Profit
165349.2	136897.8	471784.1	New York	192262
162597.7	151377.59	443898.53	California	191792
153441.51	101145.55	407934.54	Florida	191050
144372.41	118671.85	383199.62	New York	182902
142107.34	91391.77	366168.42	Florida	166188
131876.9	99814.71	362861.36	New York	156991
134615.46	147198.87	127716.82	California	156123
130298.13	145530.06	323876.68	Florida	155753
120542.52	148718.95	311613.29	New York	152212
123334.88	108679.17	304981.62	California	149760
101913.08	110594.11	229160.95	Florida	146122
100671.96	91790.61	249744.55	California	144259
93863.75	127320.38	249839.44	Florida	141586
91992.39	135495.07	252664.93	California	134307



The dataset won't get fit into the below formula:

$$b_0 = \text{intercept} = \frac{\text{sum}(y)*\text{sum}(x^2) - \text{sum}(x)\text{sum}(xy)}{n*\text{sum}(x^2) - \text{sum}(x)^2}$$

$$b_1 = \text{slope} = \frac{n*\text{sum}(xy) - \text{sum}(x)*\text{sum}(y)}{n*\text{sum}(x^2) - \text{sum}(x)^2}$$

To handle the state column (category), we need encoding. We will convert it to numeric format.

Encoding/dummy variable creation /One hot encoder:

Encoding takes unique values and alphabetically sorted. It will represented each unique value as a columns.

California	d0_C
Florida	d1_F
New York	d2_NY

State	d0_C	d1_F	d2_NY
New York	0	0	1
California	1	0	0
Florida	0	1	0

Machine learns through rows – identifying the pattern instead of columns.

You can remove the dummy variables.

State	d0_C	d1_F	d2_NY
New York	0	0	1
California	1	0	0
Florida	0	1	0

In order to have a balance learning, dummy variable is best suited. We won't use weighted labels.

numOfDummyVar = numOfUniqVal - 1

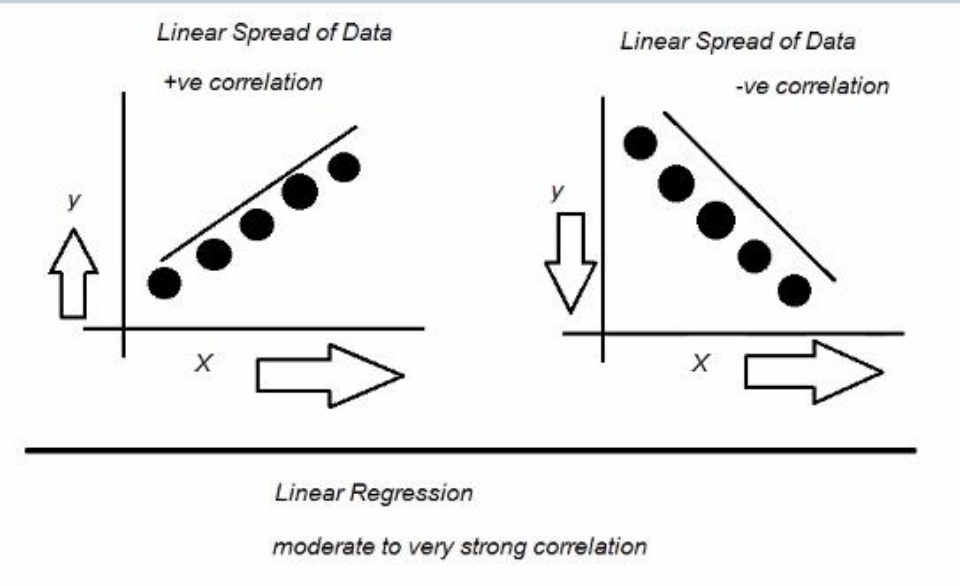
California	d0_C	0
Florida	d1_F	1
New York	d2_NY	2
Texas		3
		4
		5

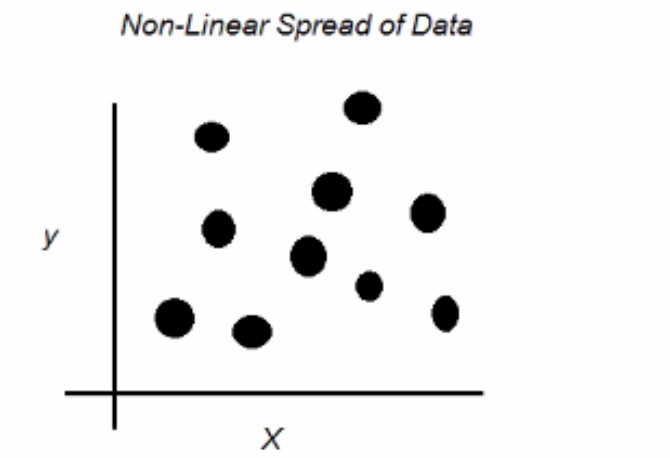
XNew - DataFrame

Index	RNDSpend	Administration	MarketingSpend	State_California	State_Florida	State_New York
0	165349	136898	471784	0	0	1
1	162598	151378	443899	1	0	0
2	153442	101146	407935	0	1	0
3	144372	118672	383200	0	0	1
4	142107	91391.8	366168	0	1	0
5	131877	99814.7	362861	0	0	1
6	134615	147199	127717	1	0	0
7	130298	145530	323877	0	1	0
8	120543	148719	311613	0	0	1
9	123335	108679	304982	1	0	0
10	101913	110594	229161	0	1	0
11	100672	91790.6	249745	1	0	0
12	93863.8	127320	249839	0	1	0
13	91992.4	135495	252665	1	0	0
14	119943	156547	256513	0	1	0
15	114524	122617	261776	0	0	1
16	78013.1	121598	264346	1	0	0
17	94657.2	145078	282574	0	0	1
18	91749.2	114176	294920	0	1	0
19	86419.7	153514	0	0	0	1
20	76253.9	113867	298664	1	0	0
21	78389.5	153773	299737	0	0	1
22	73994.6	122783	303319	0	1	0

XDrop - DataFrame

Index	RNDSpend	Administration	MarketingSpend	State_Florida	State_New York
0	165349	136898	471784	0	1
1	162598	151378	443899	0	0
2	153442	101146	407935	1	0
3	144372	118672	383200	0	1
4	142107	91391.8	366168	1	0
6	134615	147199	127717	0	0
5	131877	99814.7	362861	0	1
7	130298	145530	323877	1	0
9	123335	108679	304982	0	0
8	120543	148719	311613	0	1
14	119943	156547	256513	1	0
15	114524	122617	261776	0	1
10	101913	110594	229161	1	0
11	100672	91790.6	249745	0	0
17	94657.2	145078	282574	0	1





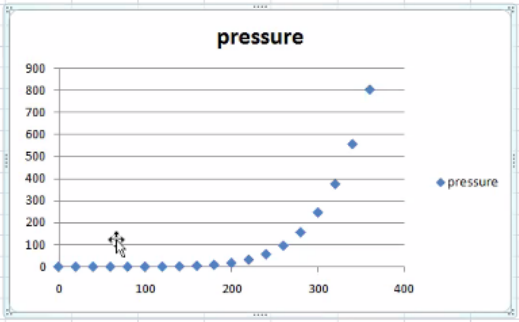
correlation of most of the X features will be in negligible to no-correlation

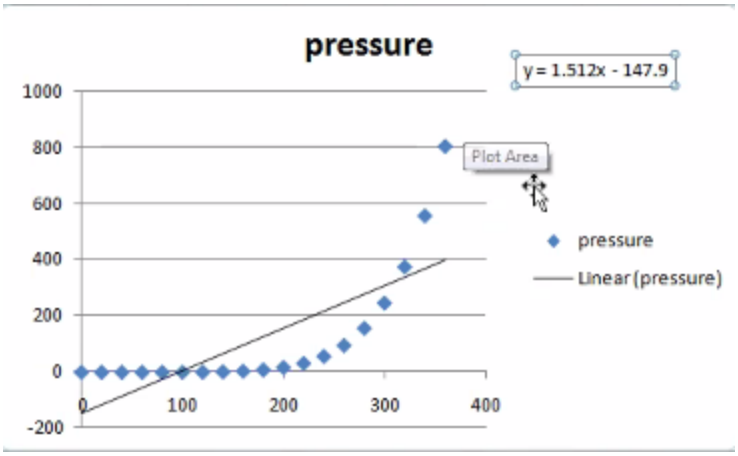
Polynomial Regression
DecisionTree Reg.
RandomForest Reg.

Take the example below:

This is not a linear problem.

tempreture	pressure
0	0.0002
20	0.0012
40	0.006
60	0.03
80	0.09
100	0.27
120	0.75
140	1.85
160	4.2
180	8.8
200	17.3
220	32.1
240	57
260	96
280	157
300	247
320	376
340	558
360	806



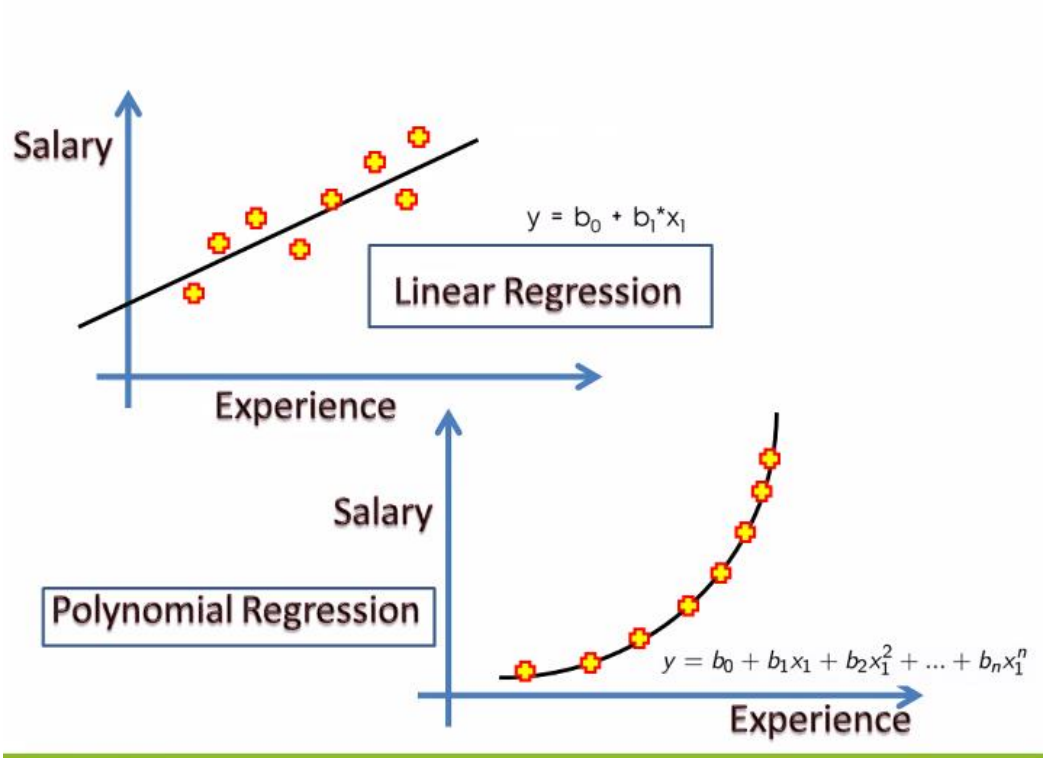


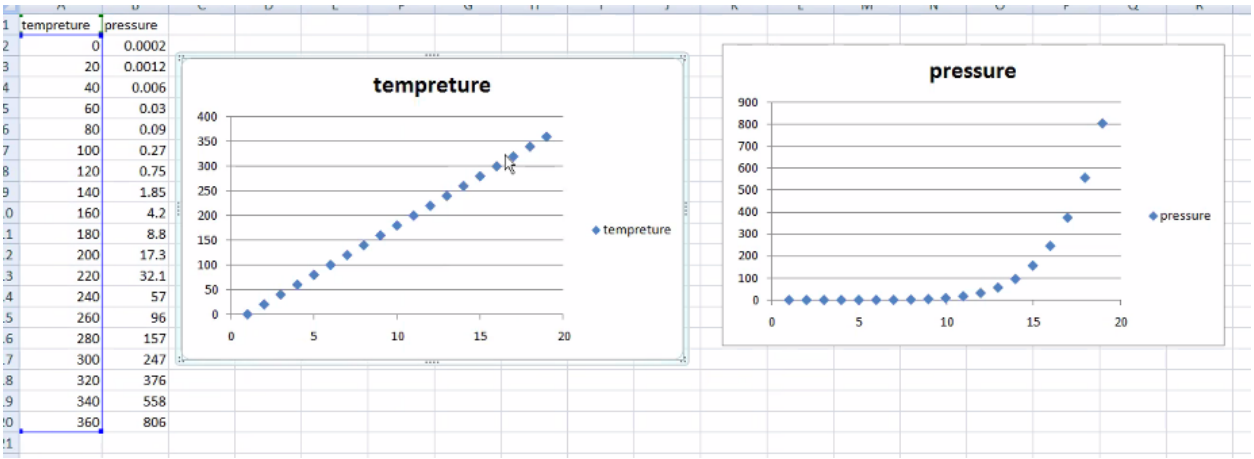
Here linear regression formula falls short. So, in this case you must use polynomial regression

Polynomial Regression

$$y = b_0 + b_1 x_1 + b_2 x_1^2 + \ldots + b_n x_1^n$$

By adding degree to each feature, we can solve it. This will curve the best fit line.

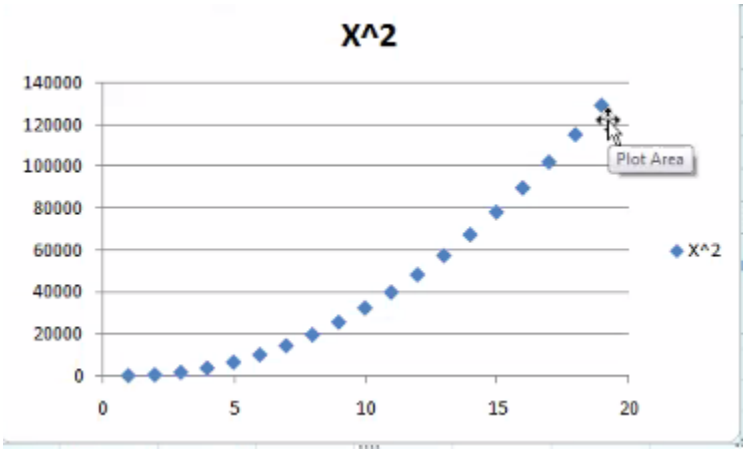


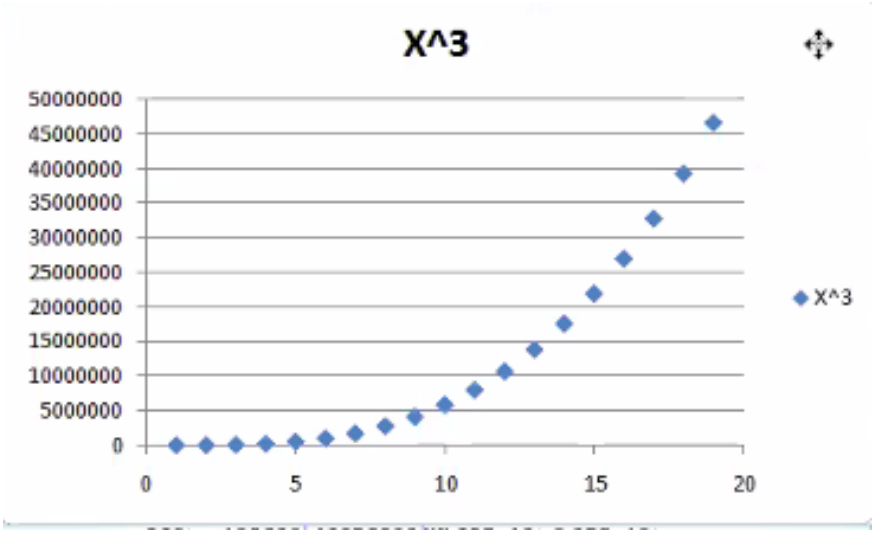


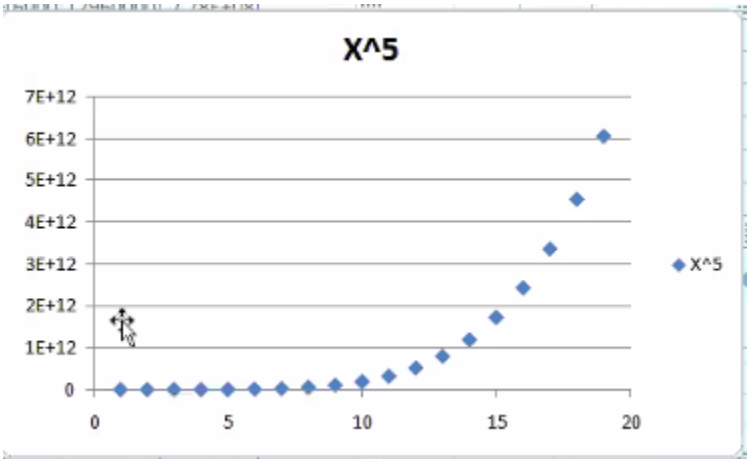
tempreture	pressure
0	0.0002
20	0.0012
40	0.006
60	0.03
80	0.09
100	0.27
120	0.75
140	1.85
160	4.2
180	8.8
200	17.3
220	32.1
240	57
260	96
280	157
300	247
320	376
340	558
360	806

You need to tweet the x value.

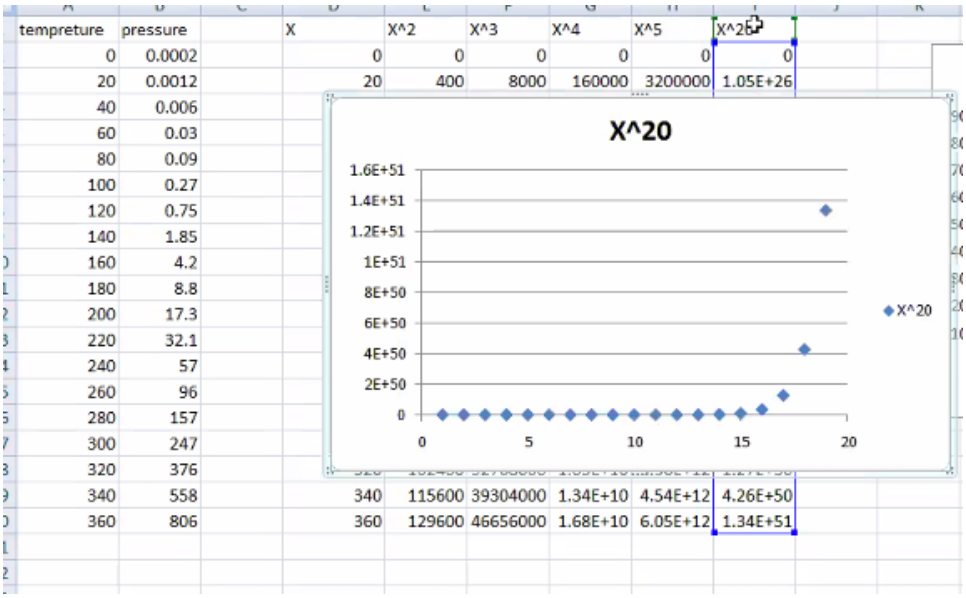
tempreture	pressure		X	X^2	X^3	X^4	X^5
0	0.0002		0	0	0	0	0
20	0.0012		20	400	8000	160000	3200000
40	0.006		40	1600	64000	2560000	1.02E+08
60	0.03		60	3600	216000	12960000	7.78E+08
80	0.09		80	6400	512000	40960000	3.28E+09
100	0.27		100	10000	1000000	1E+08	1E+10
120	0.75		120	14400	1728000	2.07E+08	2.49E+10
140	1.85		140	19600	2744000	3.84E+08	5.38E+10
160	4.2		160	25600	4096000	6.55E+08	1.05E+11
180	8.8		180	32400	5832000	1.05E+09	1.89E+11
200	17.3		200	40000	8000000	1.6E+09	3.2E+11
220	32.1		220	48400	10648000	2.34E+09	5.15E+11
240	57		240	57600	13824000	3.32E+09	7.96E+11
260	96		260	67600	17576000	4.57E+09	1.19E+12
280	157		280	78400	21952000	6.15E+09	1.72E+12
300	247		300	90000	27000000	8.1E+09	2.43E+12
320	376		320	102400	32768000	1.05E+10	3.36E+12
340	558		340	115600	39304000	1.34E+10	4.54E+12
360	806		360	129600	46656000	1.68E+10	6.05E+12





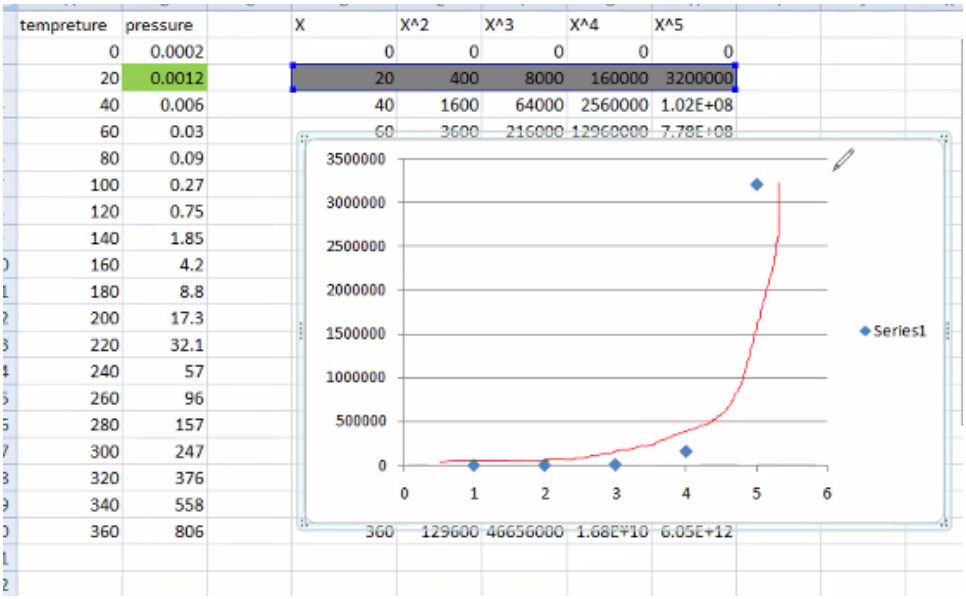


More the degree, better the result. But not too far!



tempreture	pressure	X	X^2	X^3	X^4	X^5	X^26
0	0.0002	0	0	0	0	0	0
20	0.0012	20	400	8000	160000	3200000	1.05E+26
40	0.006						
60	0.03						
80	0.09						
100	0.27						
120	0.75						
140	1.85						
160	4.2						
180	8.8						
200	17.3						
220	32.1						
240	57						
260	96						
280	157						
300	247						
320	376						
340	558	340	115600	39304000	1.34E+10	4.54E+12	4.26E+50
360	806	360	129600	46656000	1.68E+10	6.05E+12	1.34E+51

It’s a trial-and-error method. Until we try with different degree, we won’t know which is best.



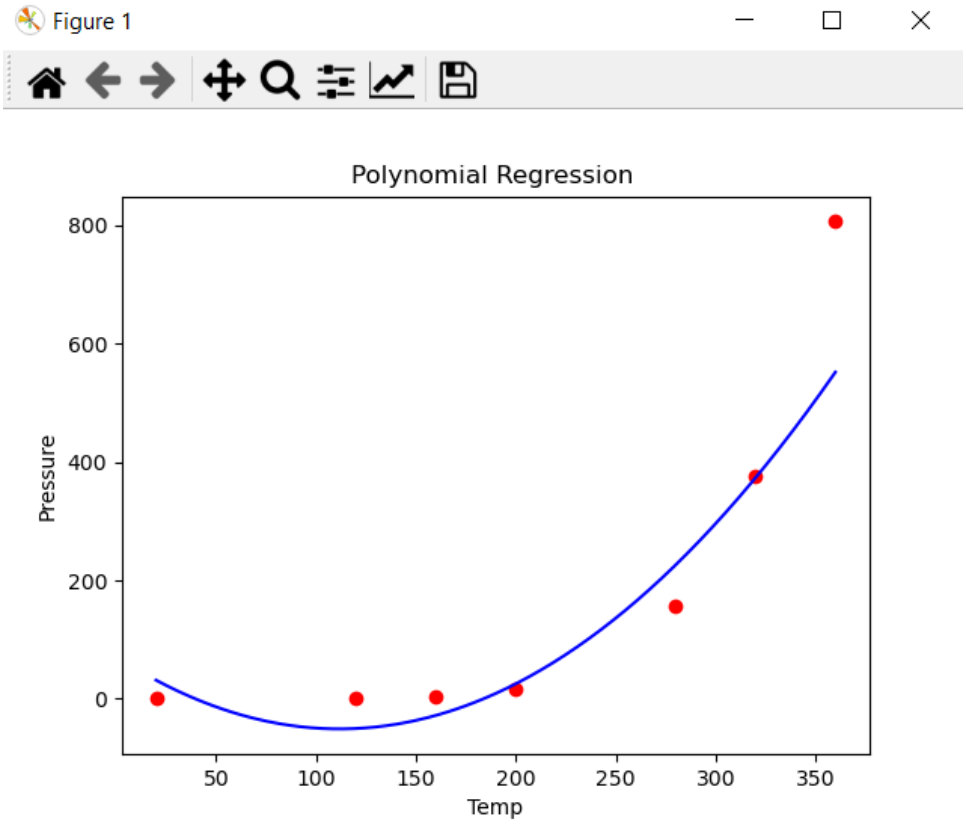
tempreture	pressure	X	X^2	X^3	X^4	X^5
0	0.0002	0	0	0	0	0
20	0.0012	20	400	8000	160000	3200000
40	0.006	40	1600	64000	2560000	1.02E+08
60	0.03	60	3600	216000	12960000	7.78E+08
80	0.09					
100	0.27					
120	0.75					
140	1.85					
160	4.2					
180	8.8					
200	17.3					
220	32.1					
240	57					
260	96					
280	157					
300	247					
320	376					
340	558					
360	806	360	129600	46656000	1.68E+10	6.05E+12

Machine is least interested in column names. Machine is not aware of which column it deals with. It considers it as feature 1,2,3,4, and so on.

tempreture	pressure	x1	x2	x3	x4	x5
0	0.0002	0	0	0	0	0
20	0.0012	20	400	8000	160000	3200000
40	0.006	40	1600	64000	2560000	1.02E+08
60	0.03	60	3600	216000	12960000	7.78E+08
80	0.09	80	6400	512000	40960000	3.28E+09
100	0.27	100	10000	1000000	1E+08	1E+10
120	0.75	120	14400	1728000	2.07E+08	2.49E+10
140	1.85	140	19600	2744000	3.84E+08	5.38E+10
160	4.2	160	25600	4096000	6.55E+08	1.05E+11
180	8.8	180	32400	5832000	1.05E+09	1.89E+11
200	17.3	200	40000	8000000	1.6E+09	3.2E+11
220	32.1	220	48400	10648000	2.34E+09	5.15E+11
240	57	240	57600	13824000	3.32E+09	7.96E+11
260	96	260	67600	17576000	4.57E+09	1.19E+12
280	157	280	78400	21952000	6.15E+09	1.72E+12
300	247	300	90000	27000000	8.1E+09	2.43E+12
320	376	320	102400	32768000	1.05E+10	3.36E+12
340	558	340	115600	39304000	1.34E+10	4.54E+12
360	806	360	129600	46656000	1.68E+10	6.05E+12

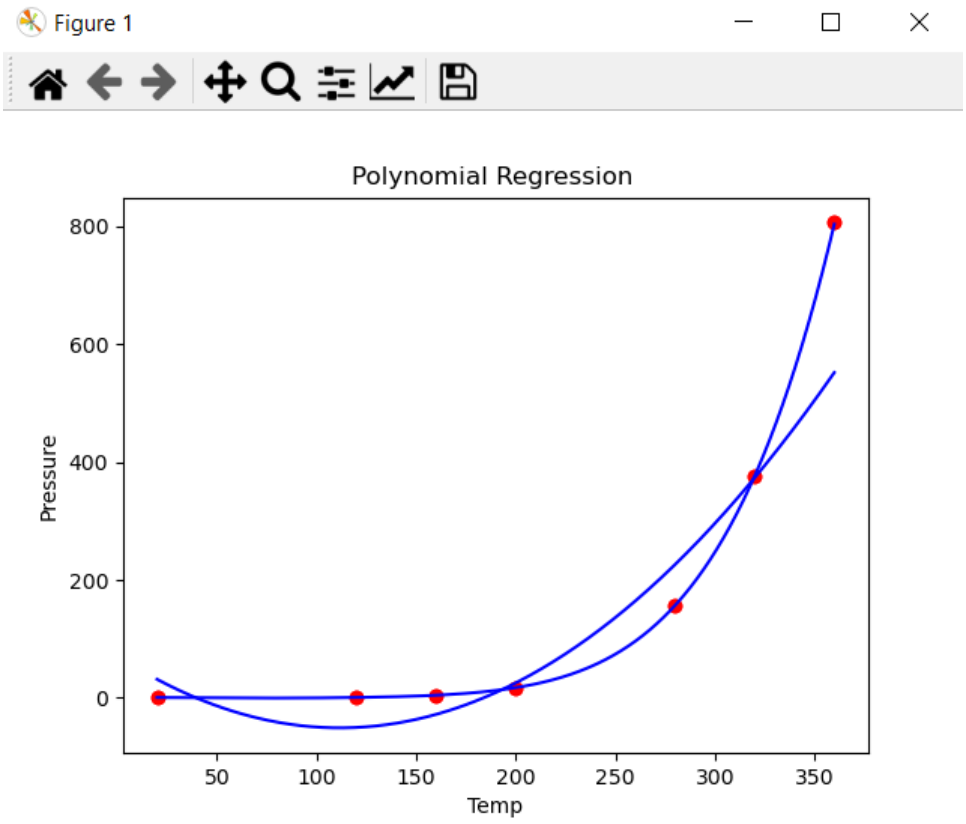
Degree 2 output:

Error is huge here.



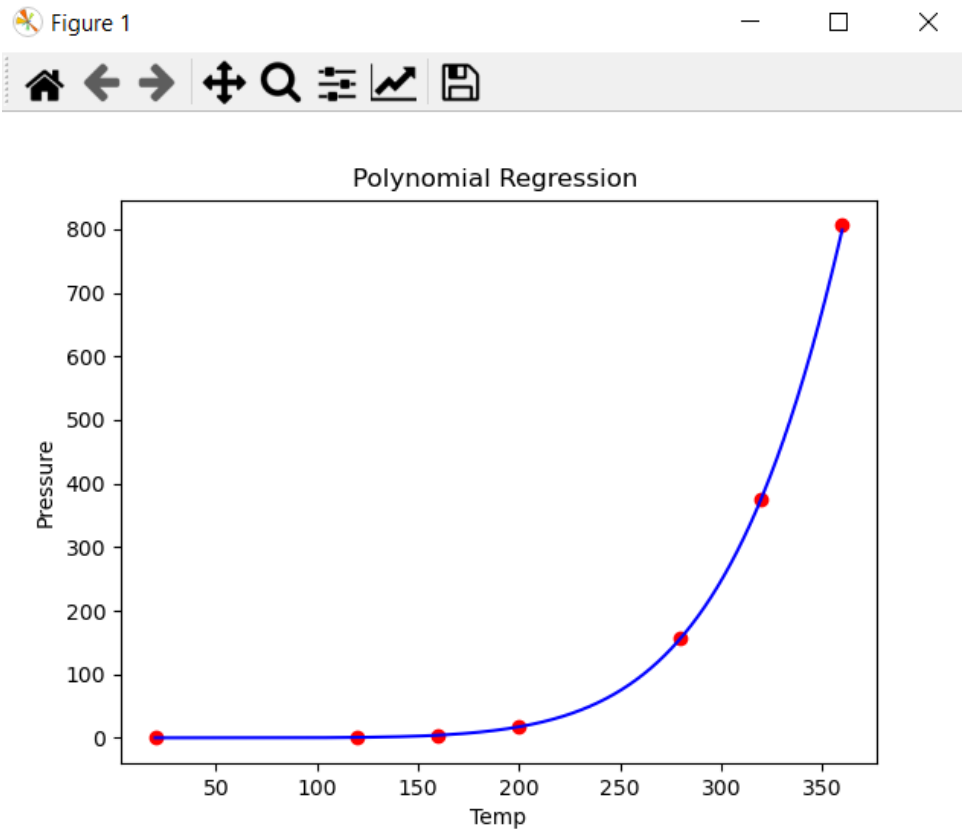
Degree 5:

Error is less. It’s fits 100%.



error_2	float64	1	10553.342229990534
error_5	float64	1	0.40559764248728847

Degree 10:



error_2	float64	1	10553.342229990534
error_5	float64	1	0.40559764248728847
error_10	float64	1	5.366908457395362

Degree 15:

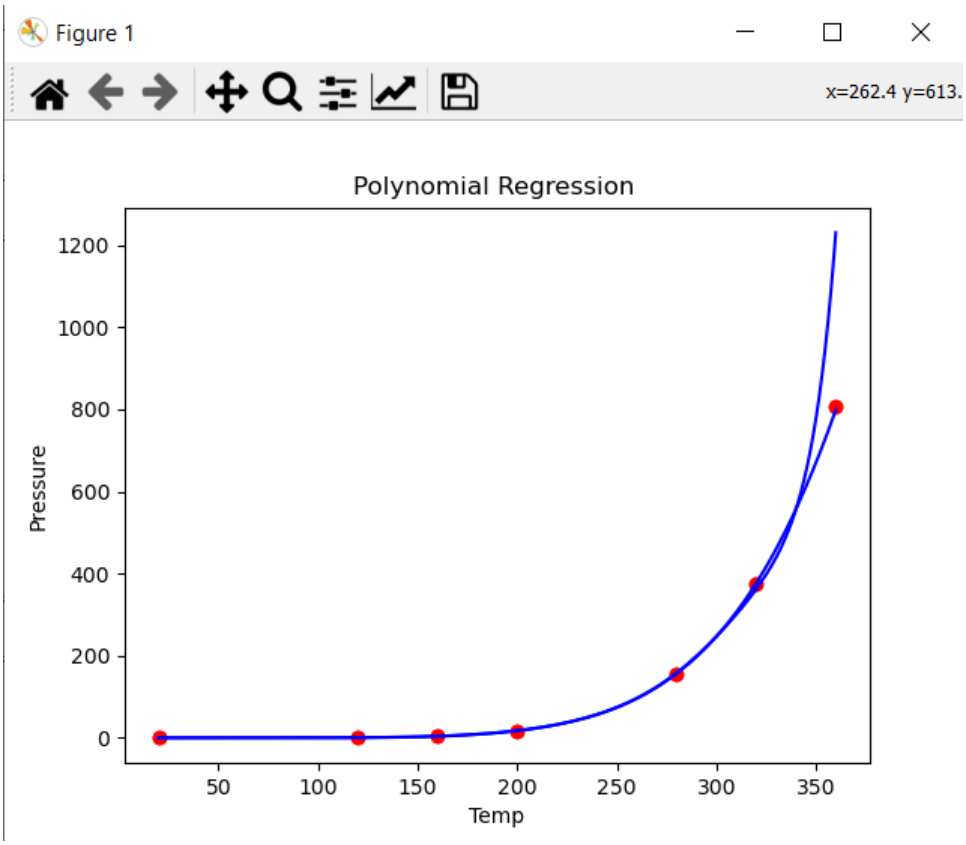
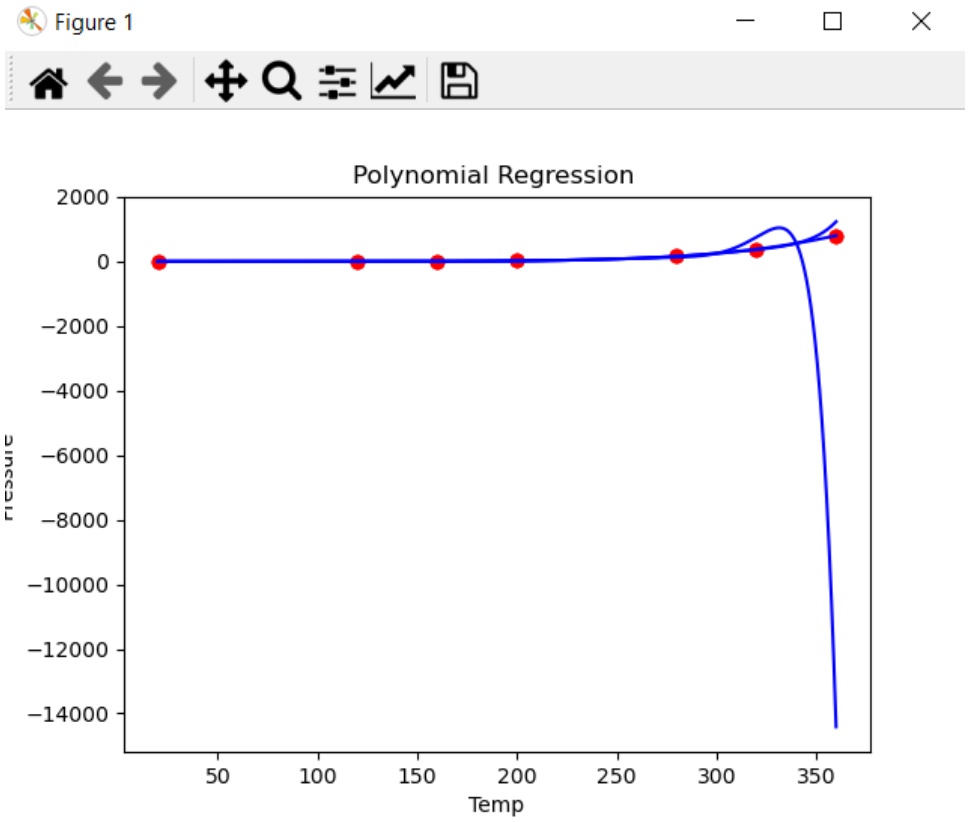


Figure 1
x=262.4 y=613.
Polynomial Regression
Pressure
Temp
0
200
400
600
800
1000
1200
50
100
150
200
250
300
350

Degree 20:



error_2	float64	1	10553.342229990534
error_5	float64	1	0.40559764248728847
error_10	float64	1	5.366908457395362
error_15	float64	1	26561.71204403531
error_20	float64	1	33961969.21293781

What is bias here?

```
# ------------------------------------------------------------------
from sklearn.preprocessing import PolynomialFeatures
poly_Obj = PolynomialFeatures(degree=20  ,  include_bias=False)
Xtrain_poly = poly_Obj.fit_transform(X_train)
Xtest_poly = poly_Obj.fit_transform(X_test)
```

We already discussed about the intercept. It helps to get the curve close to the spread of datapoints.

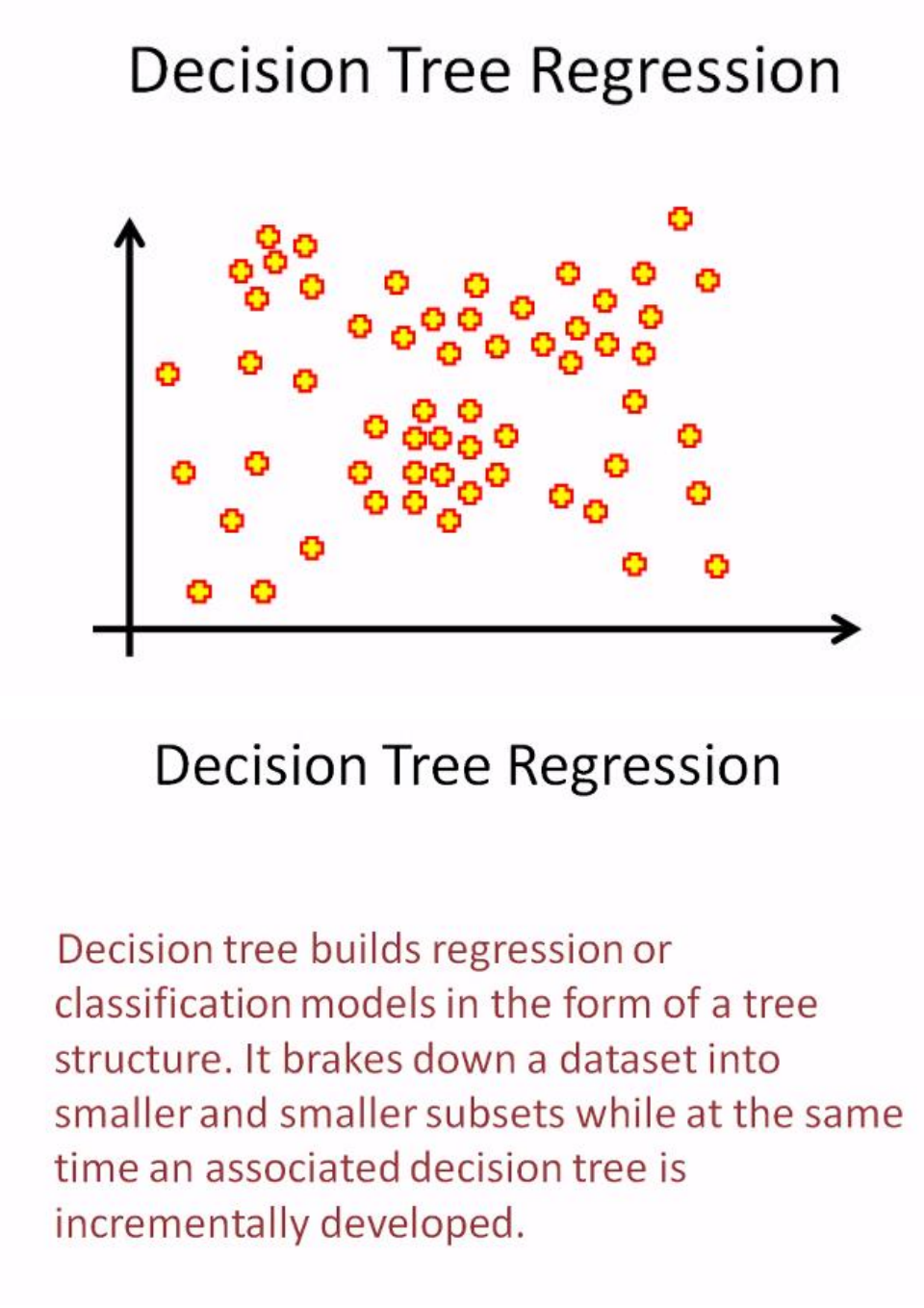
We can manually insert the intercept to keep it close to the spread of the information instead of machine calculating.

Feeding the intercept manually helps or not – range 0 to 1 will be good

Sometimes polynomial won't help. Go for Decision tree and Random Forest.

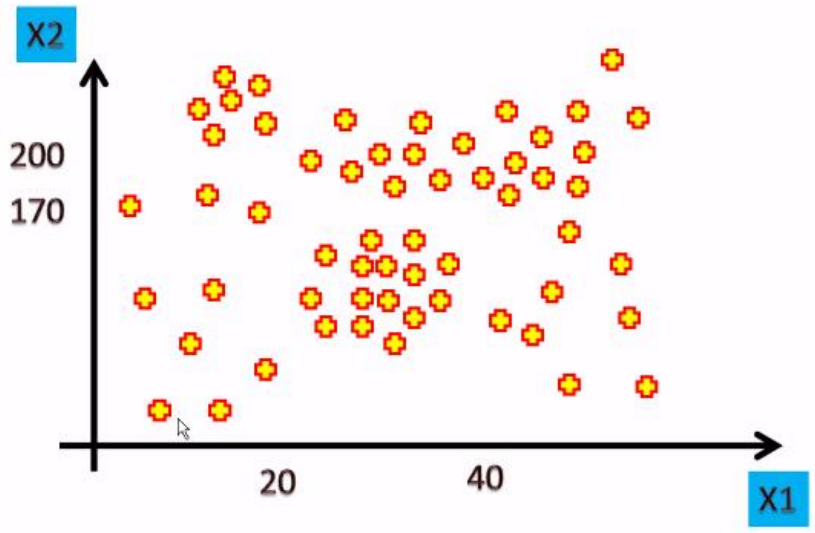
Decision tree regression:

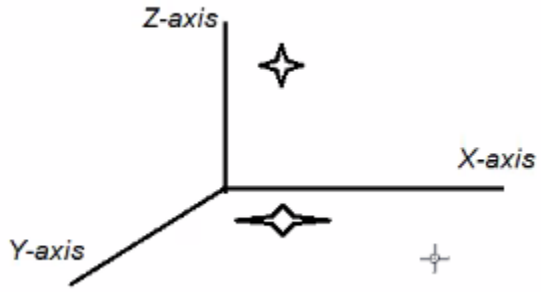
Forms in a tree structure. Breaks down into smaller and smaller subsets.



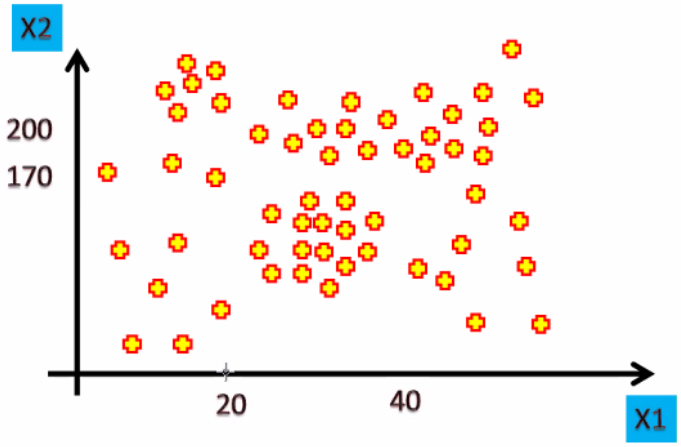
3 D Plotting looks like:

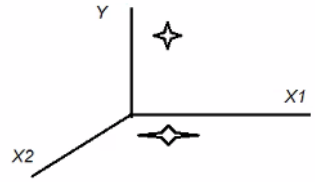
Decision Tree Regression



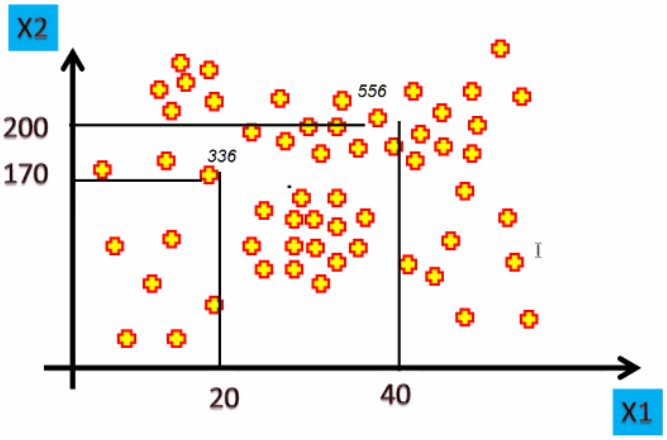


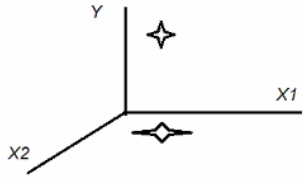
	input	*output*
X1	*X2*	*y*
20	*170*	*336*
40	*200*	*556*





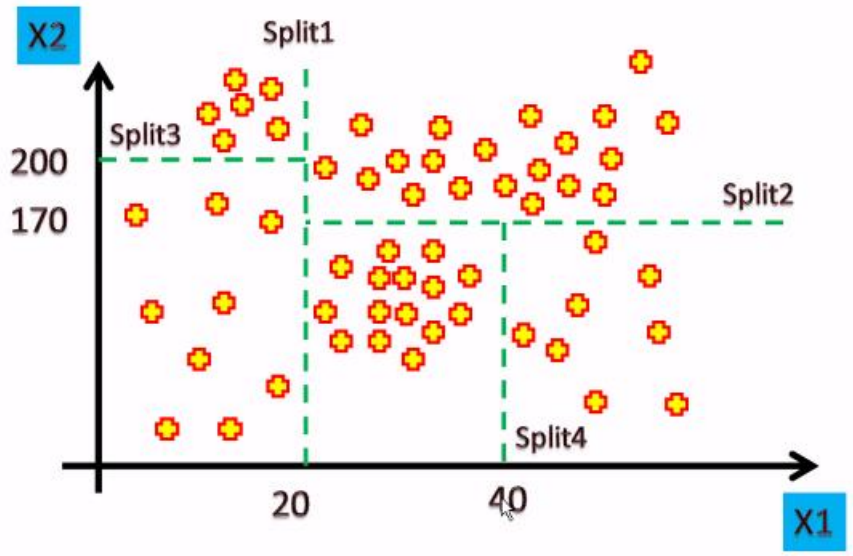
X1	X2	y
20	170	336
40	200	556



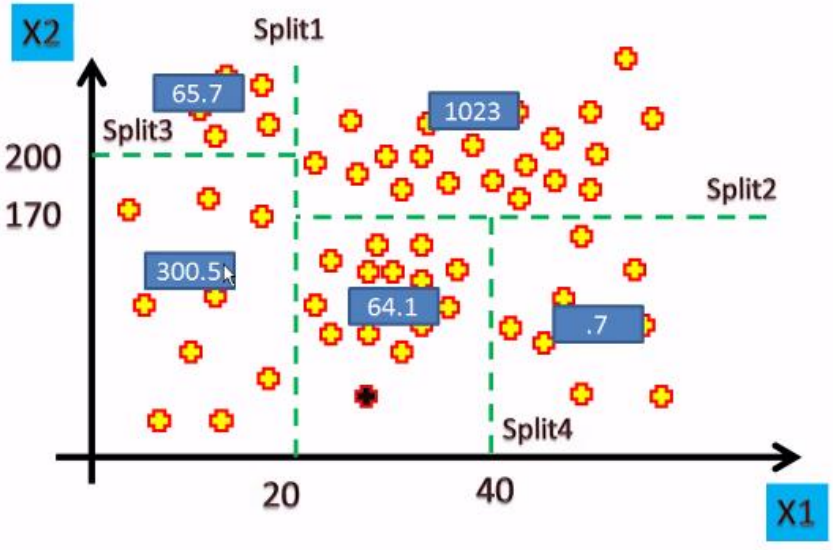


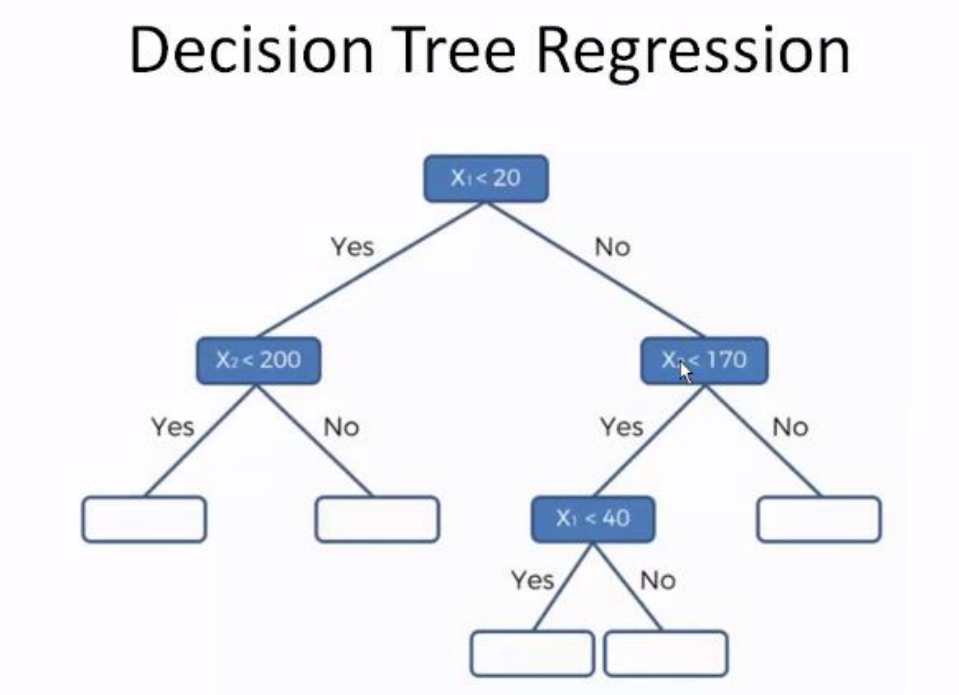
X1	X2	y
20	170	336
40	200	556

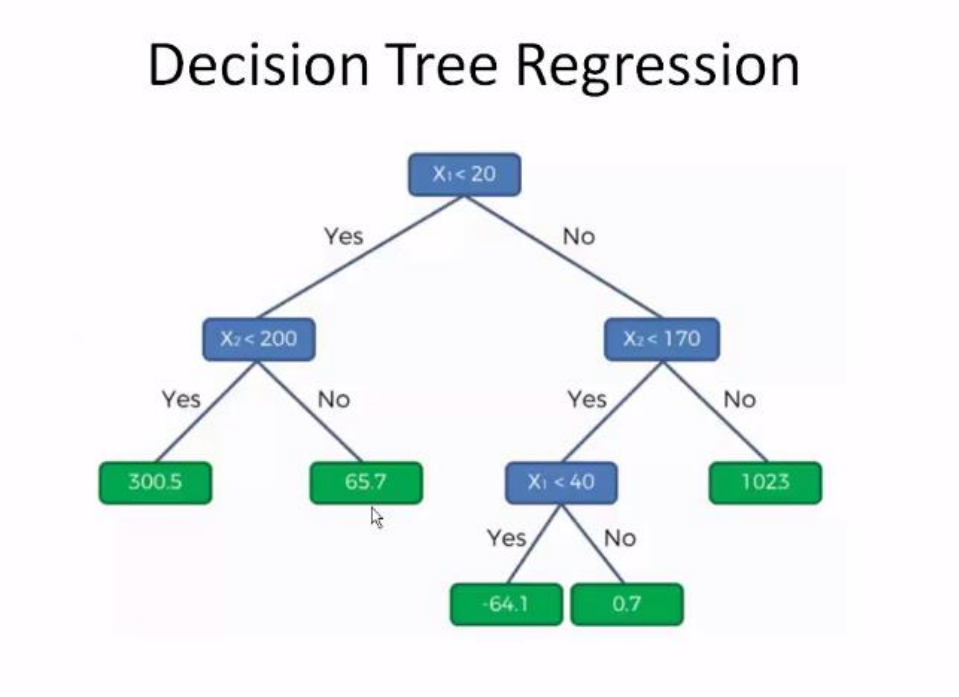
Decision Tree Regression



Decision Tree Regression





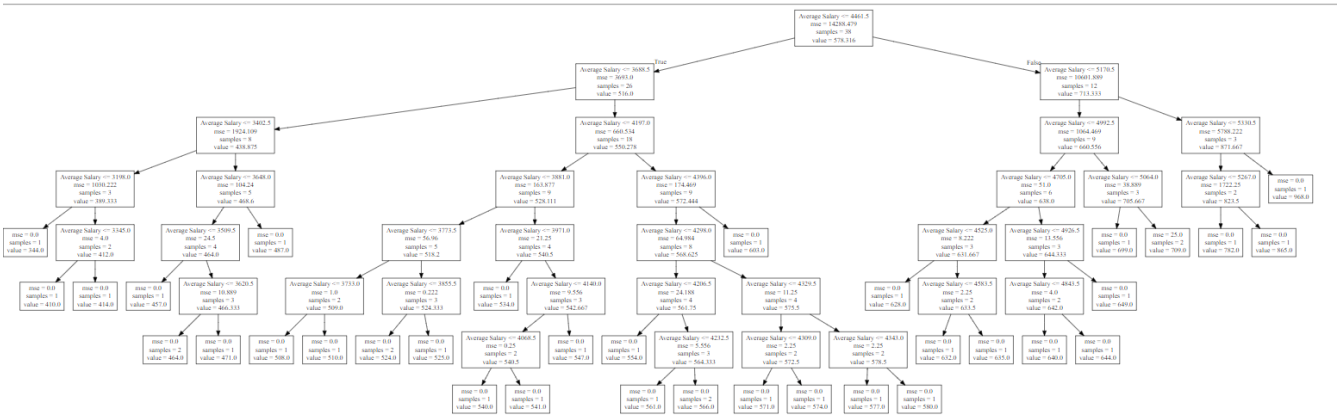


- Iterative in nature and time consuming
- Handle the complexity better than the polynomial regression.
- Good solution
- Performance – takes extra time.
- We can see the actual tree as well
- You will come here only when the polynomial regression is not good.

- Simple is better
- Prediction is just substituting values into the formula and getting it.
- Decision tree is good when we have good amount of data.
- Less data – not good to go with decision tree
- If the points falls under the split region, then the predicted value will be the average of that split region.

Web Graphviz: To visualize the tree structure

http://www.webgraphviz.com/



There will be overfitting problem! More splitting is done here.

Splitting has done in a granular level.

Overfitting:

Machine learns much on the training data and does only good only on training data.

It lacks generalization capabilities

Gives high error/ low accuracy on testing data

Multiple test sets tried against such model will lead to unstable error or accuracy and it won't be able to give appropriate results.

Can we control the number of splits? Whether to split or not depends on the value of N?

Value of N is considered for taking this decision.

Min_samples_split = n Deafult set to 2

If we have 2 or more data points in the split we divide it further.

```
from sklearn.tree import DecisionTreeRegressor
regressor = DecisionTreeRegressor()

regressor.fit(X_train,y_train)

# ================================
# Model Testing
# ================================
y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_s
error_DT = mean_squared_error(y_te


# ================================
# Tree plotting
# ================================
```

Arguments

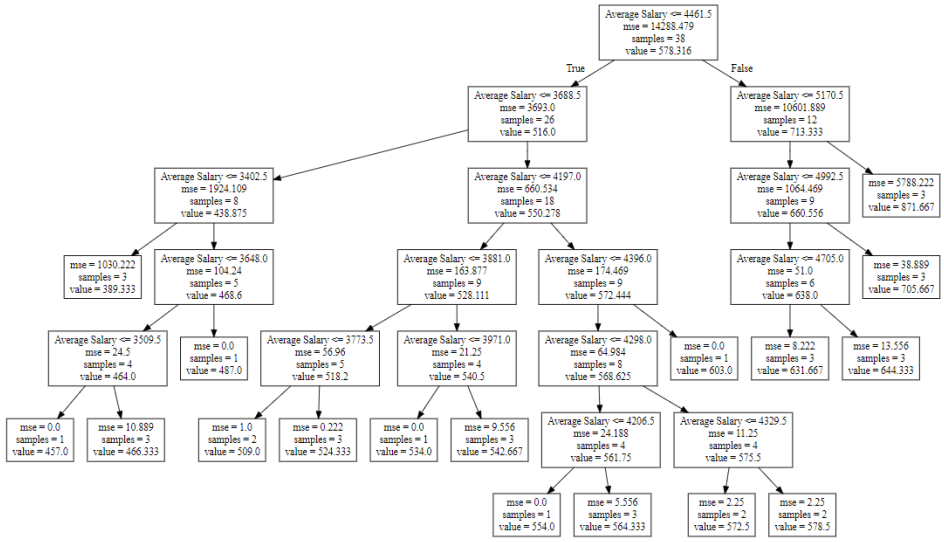
```
DecisionTreeRegressor(criterion="mse",
                      splitter="best",
                      max_depth=None,
                      min_samples_split=2,
                      min_samples_leaf=1,
                      min_weight_fraction_leaf=0.,
                      max_features=None,
                      random_state=None,
                      max_leaf_nodes=None,
                      min_impurity_decrease=0.,
                      min_impurity_split=None,
                      presort='deprecated',
                      ccp_alpha=0.0)
```

Should we split or confirm it as leaf node?

In the above case, it has spitted more. Results in overfitting!

Now, provide the min sample split as 4

error_DT	float64	1	39.9
error_DT4	float64	1	46.1722222222222



If the split value is more, then it results in **underfitting! (Too crisp and too much generalization)**

Step 1: Train model with min sample slit = n

Step 2: Test sample	Error
Sample 1	39.0
Sample 2	45.0
Sample 3	47
Sample 4	46.0

underfitting	Not learning Enough Too much of generalization the model is learning to do testing error will be very high / acc will be very low
Overfitting	Machine learns too much on training data Does only good on training data it lacks generalization capabilities gives high error / low accuracy on testing data multiple test sets tried against such model will lead to unstable error or accuracy

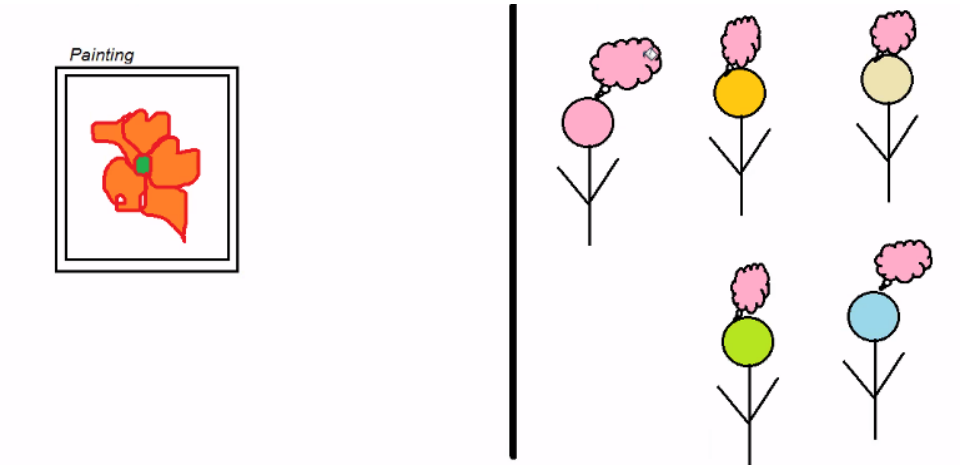
Good fit

- Enough learning
- Has good generalization capability
- Testing error low / accuracy high, and stable

Note: The error should be stable. You should be able to reproduce the same output. It should do good in all machines. So, stability check is a must!

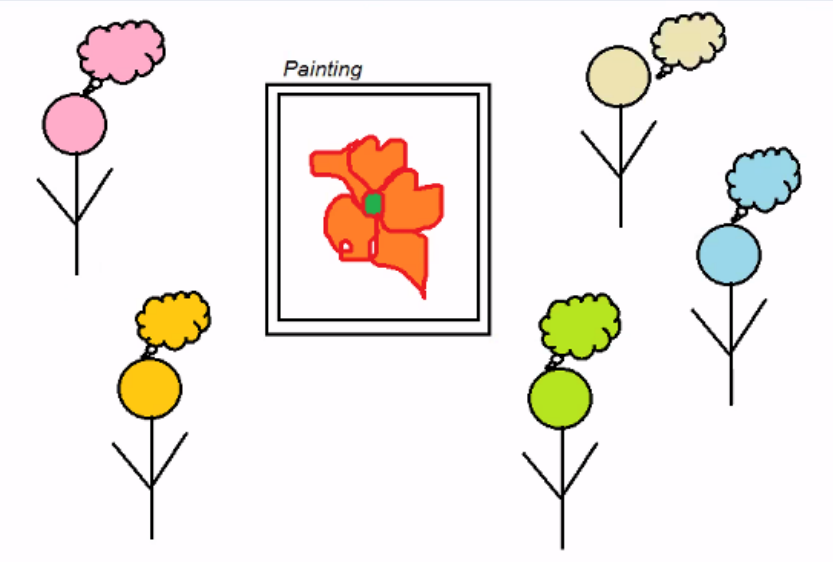
Random Forest:

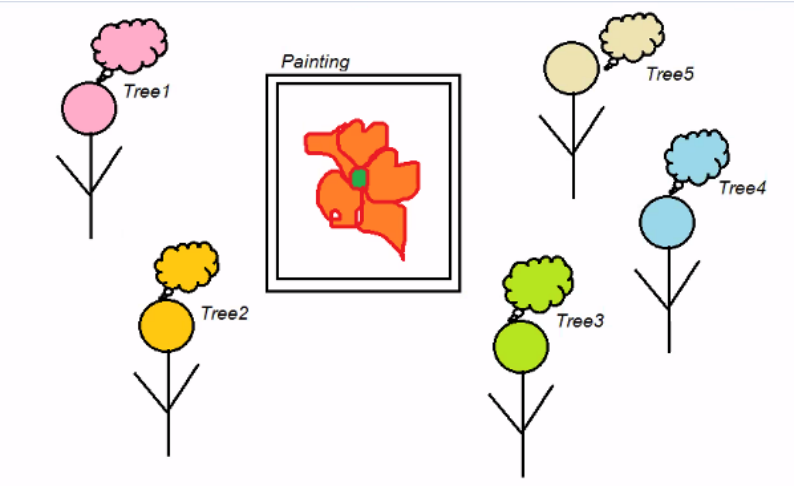
One tree is one perspective of looking at the data. That is the problem -> Inappropriate results



Now everyone is going inside the art gallery and having their own perspective. Better idea about the painting. Multiple perspective out of same painting -> Random Forest.

It decides from multiple tress.

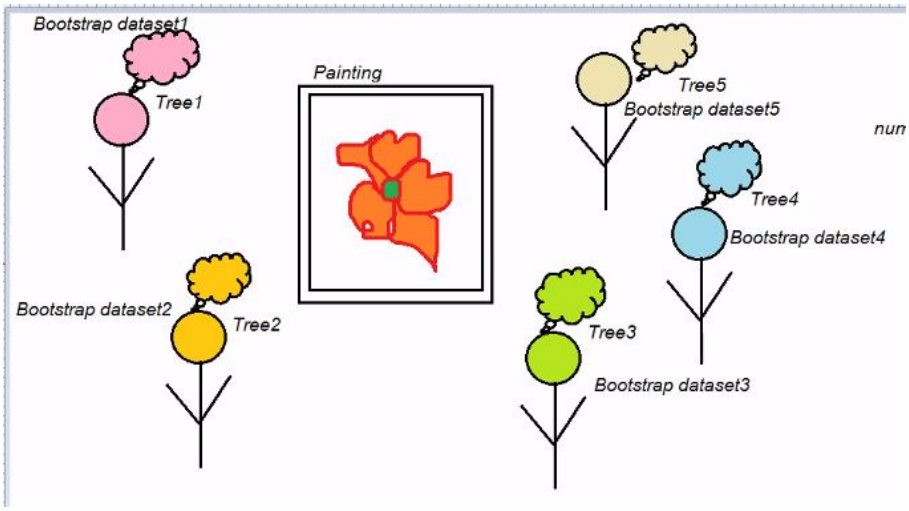




Overall Dataset will be same

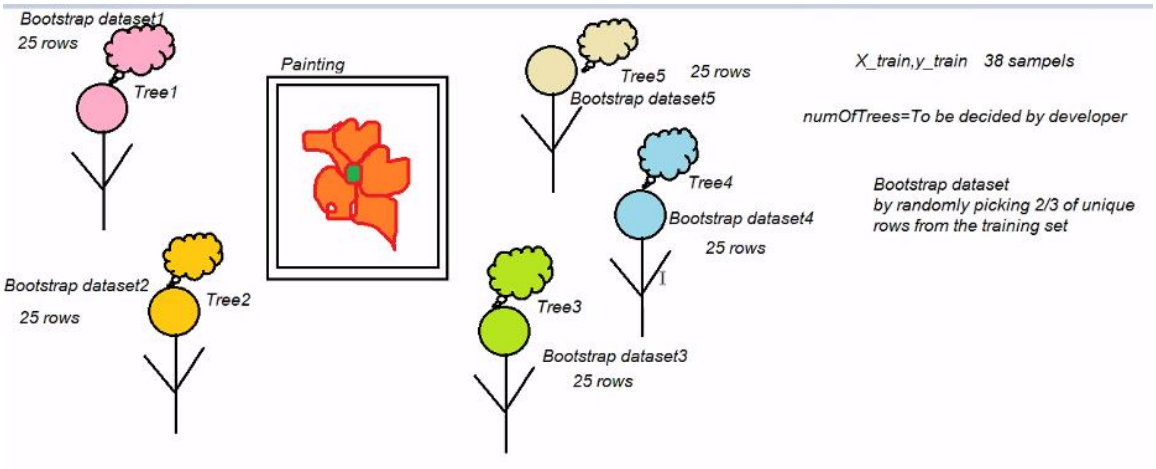
Number of trees = to be decided by developer

Each tree will have its own bootstrap dataset



How is the bootstrap dataset is constructed?

- Method to split the dataset: Bootstrap.
- Without replacement and its pure random
- By randomly picking 2/3 of unique rows from the training dataset.
- Each tree is different as it picks up different bootstrap dataset.



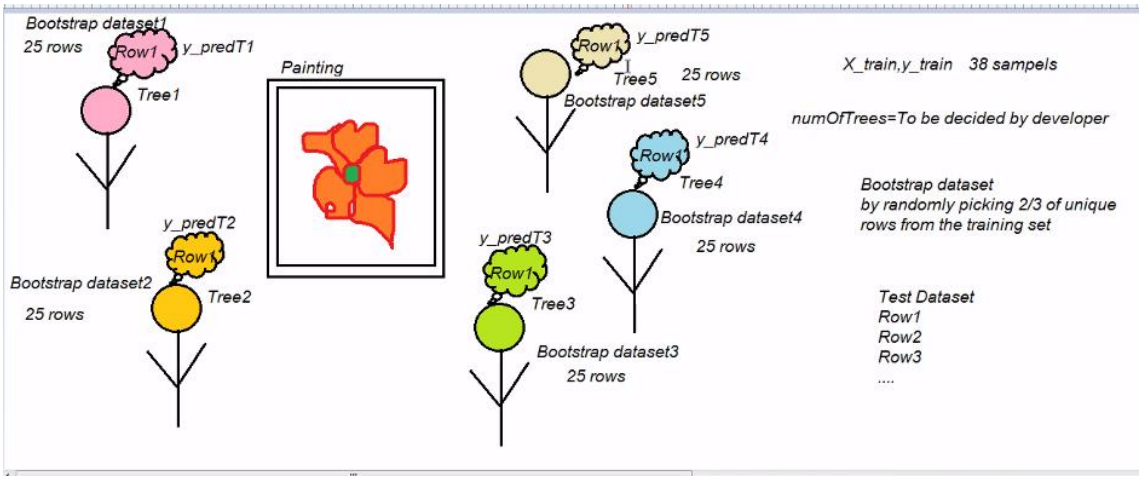
Test Dataset:

Row 1

Row 2

Row 3

…….



What am I going to do with all these predictions?

Test Dataset

Row1 = y_pred = avg(y_predT1,y_predT2,y_predT3,y_predT4,y_predT5)

Row2 = y_pred = avg(y_predT1,y_predT2,y_predT3,y_predT4,y_predT5)

Row3 = y_pred = avg(y_predT1,y_predT2,y_predT3,y_predT4,y_predT5)

....

```
NumOfTrees = To be decided by developer

RandomForest Tree Creation          (X_train, y_train)          38 samples

Step1:    => BootStrap Dataset Creation i.e. Random Row selection from the given training set
          => BootStrap dataset will be SAME SIZE as Training Dataset
                              e.g. X_train,y_train          38 samples
                                BootStrap Dataset           38 Samples

          => But Bootstrap Dataset      doesnt use all 38 values in the training set to create the BootStrap dataset
                    it just uses 2/3 of the total values in the training set to create the BootStrap dataset

                    2/3(training set)    ----------> 66% rows,  Bagging, remainder is called out of Bag Records
                    2/3(38 rows)         ----------> 25 rows random pick values from training set (66%)

          => BootStrap dataset after taking 2/3 unique rows the remainder rows are filledup to match the training set size by
                    duplicating rows from 2/3 volume

RandomForest (10 trees)

Tree1     BootStrap Dataset1
Tree2     BootStrap Dataset2
Tree3     BootStrap Dataset3
Tree4     BootStrap Dataset4
.............
Tree10    BootStrap Dataset10
```

Training Sample	
Row1	pattern1
Row2	pattern2
Row3	pattern1
Row4	pattern1
Row5	pattern2
Row6	pattern1
Row7	pattern3
Row8	pattern2
Row9	pattern2
Row10	pattern1
Row11	pattern2
BootStrap Dataset1	
Row1	pattern1
Row2	pattern2
Row3	pattern1
Row4	pattern1
Row5	pattern2
Row6	pattern1
Row7	pattern3
Row7	pattern3
Row7	pattern3
Row6	pattern1
Row6	pattern1

It’s a random pick. For understanding, done as systematic pick.

2/3 volume is the pickup volume.

Training Sample

Row1	pattern1
Row2	pattern2
Row3	pattern1
Row4	pattern1
Row5	pattern2
Row6	pattern1
Row7	pattern3
Row8	pattern2
Row9	pattern2
Row10	pattern1
Row11	pattern2

BootStrap Dataset1

Row1	pattern1
Row2	pattern2
Row3	pattern1
Row4	pattern1
Row5	pattern2
Row6	pattern1
Row7	pattern3
Row7	pattern3
Row7	pattern3
Row6	pattern1
Row6	pattern1

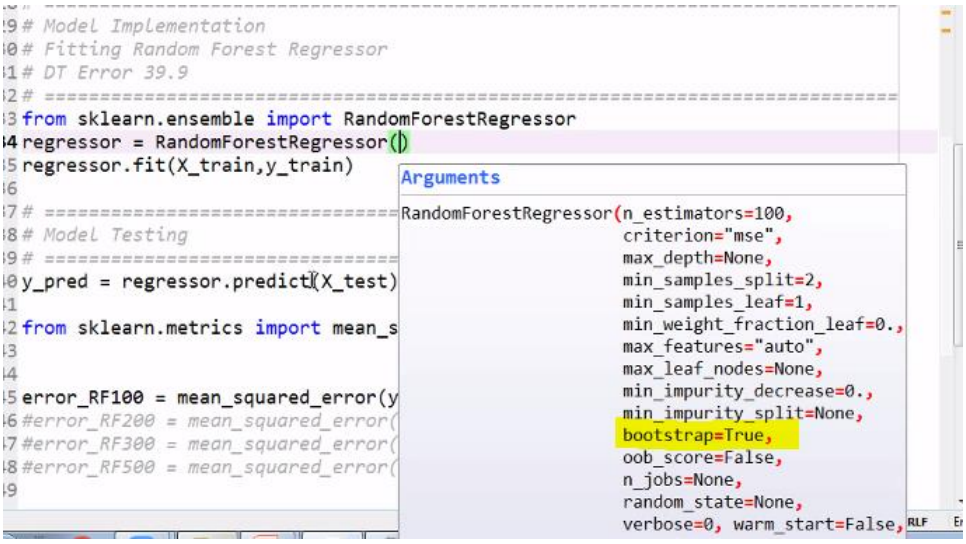
Randomly pick up from bootstrap dataset to fill the gap.

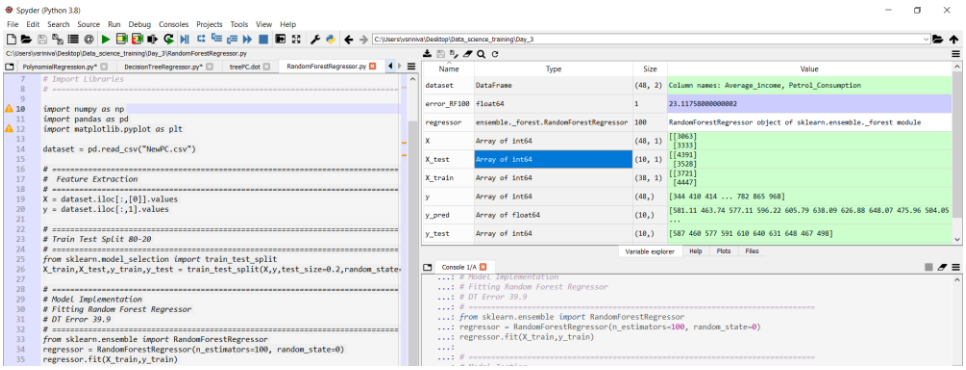
Bootstrap and training dataset size should be same.

BootStrap Dataset1

Row1	pattern1
Row2	pattern2
Row3	pattern1
Row4	pattern1
Row5	pattern2
Row6	pattern1
Row7	pattern3
Row7	pattern3
Row7	pattern3
Row6	pattern1
Row6	pattern1

Randomness won’t be effective after certain number of trees.





error_RF100	float64	1	23.11758000000002
error_RF200	float64	1	22.971057500000065
error_RF300	float64	1	23.582036666666742
error_RF500	float64	1	23.001594500000017

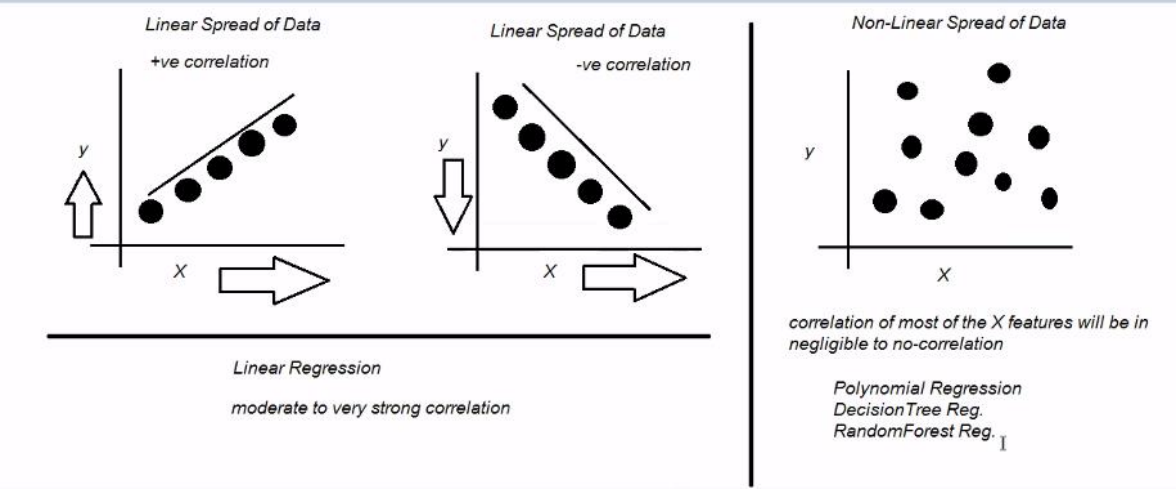
Observation:

Exhausted the error value after 100. You can see the same similarity.

Try it between 120 to 150:

error_RF100	float64	1	23.11758000000002
error_RF120	float64	1	21.793486111111214
error_RF150	float64	1	21.8920755555556
error_RF200	float64	1	22.971057500000065
error_RF300	float64	1	23.582036666666742
error_RF500	float64	1	23.001594500000017

Regression overview:



Which regression supports what kind of spread of data?

SUPERVISED LEARNING	Input Data & Output Data	Remember & Generalize
Regression	Value to Predict is Continuous	
Linear Regression		Linear spread of data
Polynomial Regression		non-Linear spread of data
DecisionTree Regressor		non-Linear spread of data
RandomForest Regressor		non-Linear spread of data
Classification	Value to Predict is Discrete	
Logistic Regression		
Support Vector Machine		
DecisionTree Classifier		
RandomForest Classifier		
Naive Bayes		

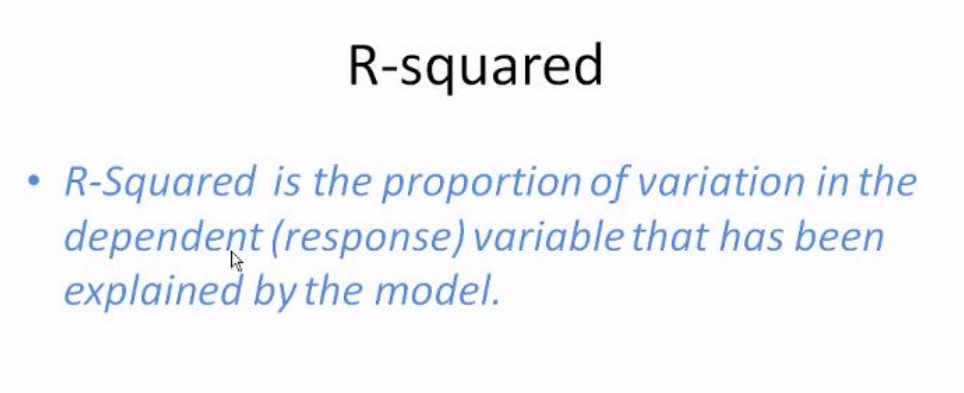
How to decide on which regression to pick up?

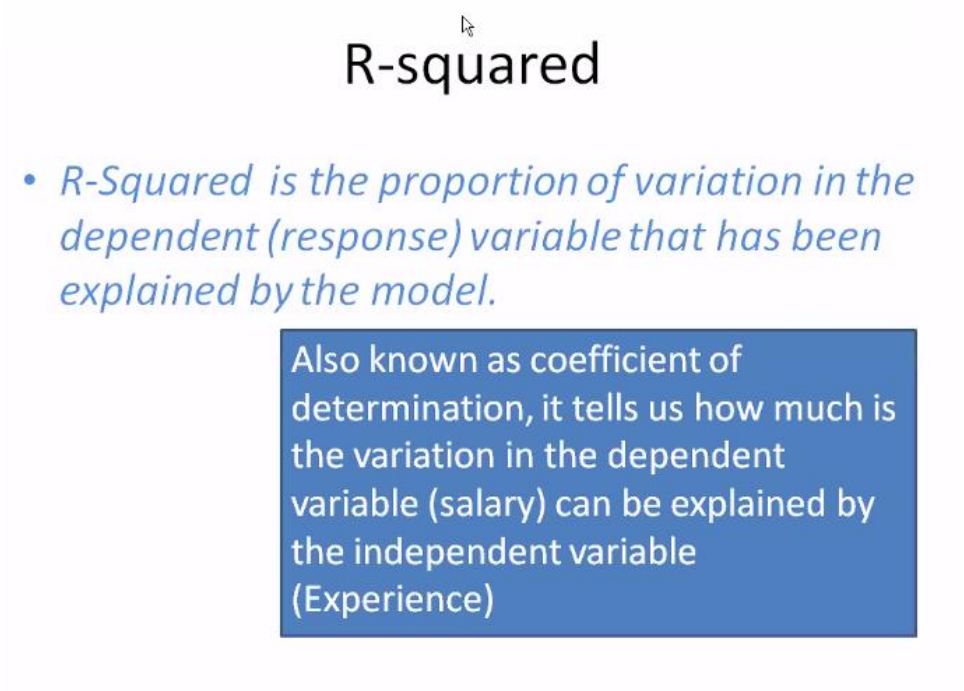
If most of the X features (80%) are moderate to very strongly correlated to Y feature
Then Choose LINEAR REGRESSION

If most of the X features (80%) are no correlation to negligible correlation to Y feature
Then Choose Poly Reg. , DT , RF

R2 score:

Tell us how much variation in y that can be explained by all x features.

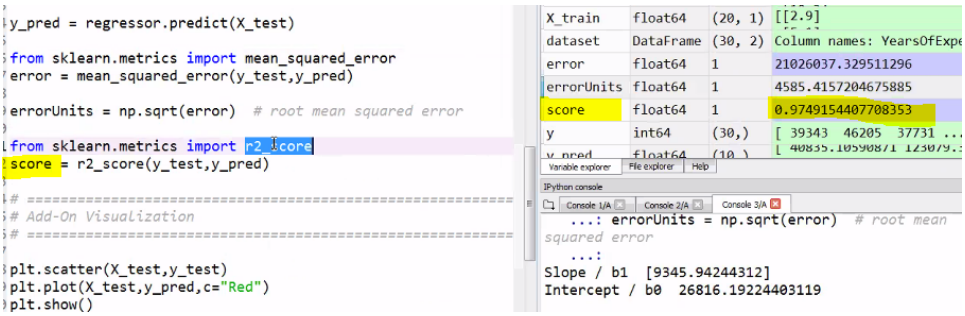




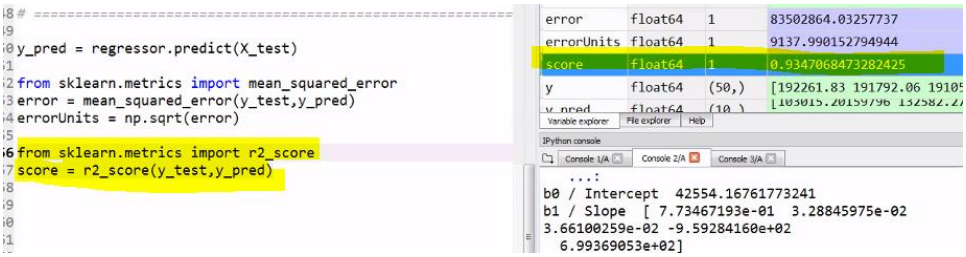
R2 score of 80 and above is healthly.

```
51 from sklearn.metrics import r2_score
52 score = r2_score(y_test,y_pred)
53
```

Simple regression:



Multiple regression:



```
y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error
error = mean_squared_error(y_test,y_pred)
errorUnits = np.sqrt(error)

from sklearn.metrics import r2_score
score = r2_score(y_test,y_pred)
```

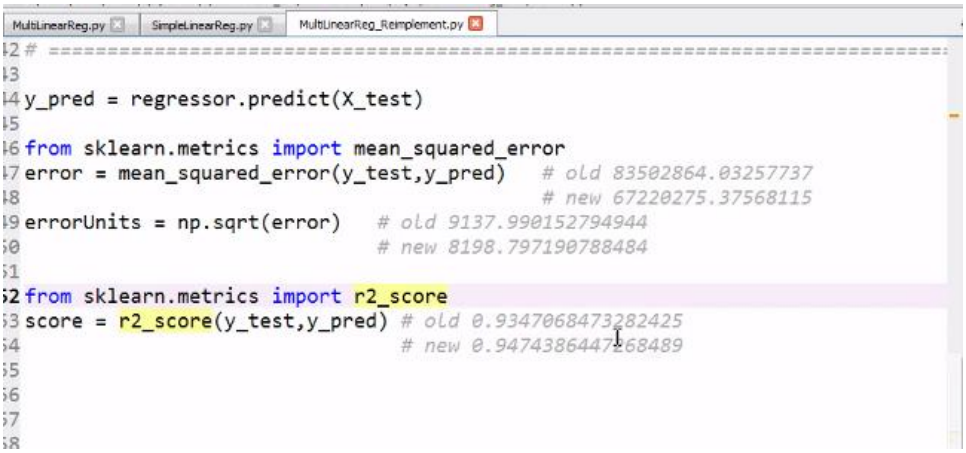
```
error       float64   1   83502864.03257737
errorUnits  float64   1   9137.990152794944
score       float64   1   0.9347068473282425
y           float64   (50,)   [192261.83 191792.06 191
y_pred      float64   (10,)   [183015.20159796 132582.27
```

```
...:
b0 / Intercept  42554.16761773241
b1 / Slope  [ 7.73467193e-01  3.28845975e-02
3.66100259e-02 -9.59284160e+02
  6.99369053e+02]
```

Pearsonr score:

```
datasetEval = pd.read_csv("02Companies.csv")

datasetEval = pd.get_dummies(data=datasetEval, columns=["State"], drop_first=True)

from scipy.stats import pearsonr

pearsonr(datasetEval['RNDSpend'], datasetEval['Profit'])
pearsonr(datasetEval['Administration'], datasetEval['Profit'])
pearsonr(datasetEval['MarketingSpend'], datasetEval['Profit'])
pearsonr(datasetEval['State_Florida'], datasetEval['Profit'])
pearsonr(datasetEval['State_New_York'], datasetEval['Profit'])
```

Taking all features vs taking specific features:



MultiLinearReg.py | SimpleLinearReg.py | MultiLinearReg_Reimplement.py

```
y_pred = regressor.predict(X_test)

from sklearn.metrics import mean_squared_error
error = mean_squared_error(y_test,y_pred)   # old 83502864.03257737
                                            # new 67220275.37568115
errorUnits = np.sqrt(error)   # old 9137.990152794944
                              # new 8198.797190788484

from sklearn.metrics import r2_score
score = r2_score(y_test,y_pred) # old 0.9347068473282425
                                # new 0.9474386447268489
```

Comparison of Linear Regression and Multiple regression:

error_LR	float64	1	556.0375904395161
error_RF	float64	1	21.8920755555556

Classification:

Use cases:

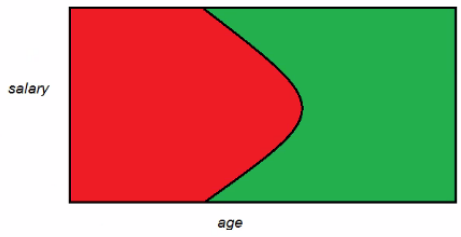
User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0
15570769	Female	26	80000	0
15606274	Female	26	52000	0
15746139	Male	20	86000	0
15704987	Male	32	18000	0
15628972	Male	18	82000	0

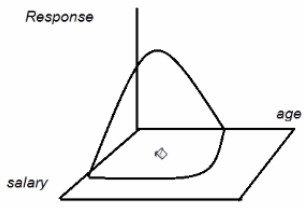
Discrete Value Prediction, Measure of model performance is Accuracy percentage

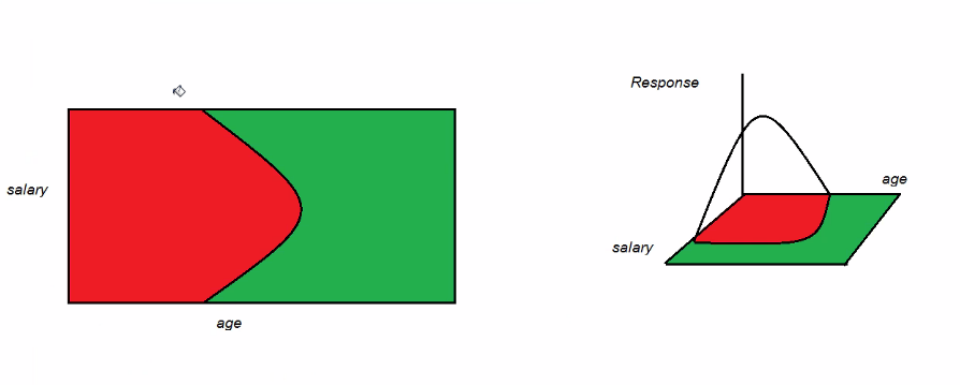
Age	ActualResponse	PredictedResponse
20	0	0
30	1	1
21	0	0
35	1	0
40	1	1
45	1	1
50	0	1
18	0	0
19	1	0

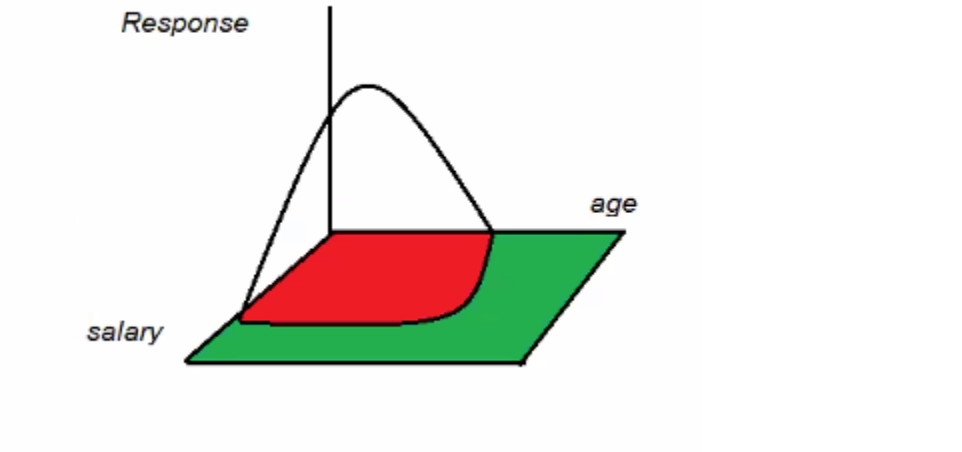
6/9*100=66% accuracy score

How am I going to generalize here?









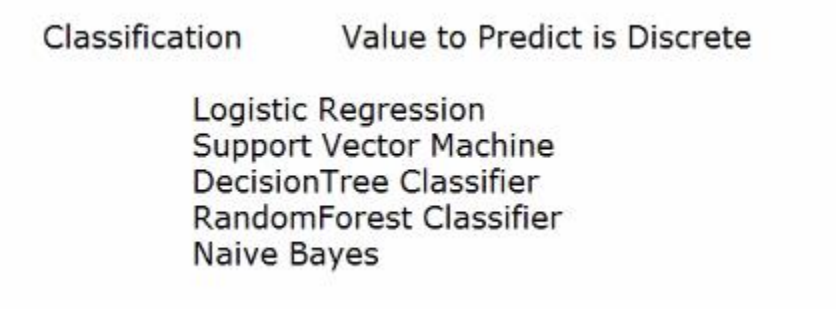
For Classification, accuracy is in the range 0 to 1. We can't take MSE here.

Consider multiple test sets and get accuracy.

Do we have a good training set?

K-fold validation – It is used for Evaluation.

Statistical calculation.



SVM: Support Vector Machine

Dataset:

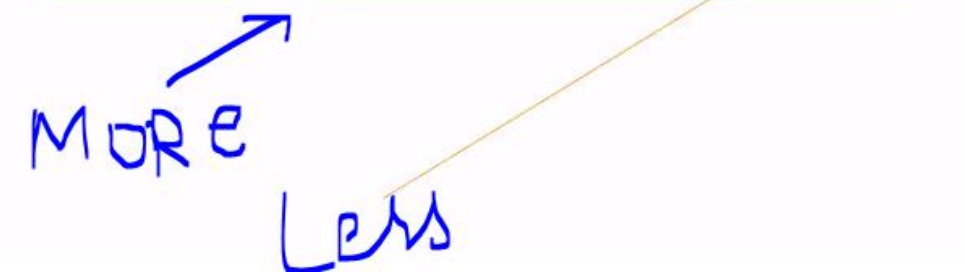
dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0

Scaling:

Age	Salary	BankBalance	CreditScore	OwnHouse
30	24000	50000	650	1

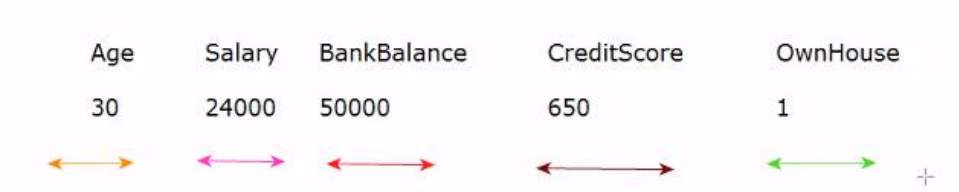
Age	Salary	BankBalance	CreditScore	OwnHouse
30	24000	50000	650	1



Bank balance: Contributes more in calculation

Own House: Contributes less in calculation

Each has its own scale



Machine doesn't understand its scale.

So, get everything under one scale

Take example: Common scale – range –2 to 2

To take balanced data

To avoid the imbalance

There are scalar available to perform this operations.

Standard scaler – good if your dataset doesn't have many outlier

Min and max scalar – good to use when your dataset has more outlier

Max absolute scalar

Robust scalar

Most popular standard, and min and max scalar

Standardization StandardScaler *(if the dataset doesnot have too many outliers)*

$$X_{stand} = \frac{X - mean(X)}{standard\ deviation\ (x)}$$

MinMaxScaler *(if the dataset has too many outliers)*

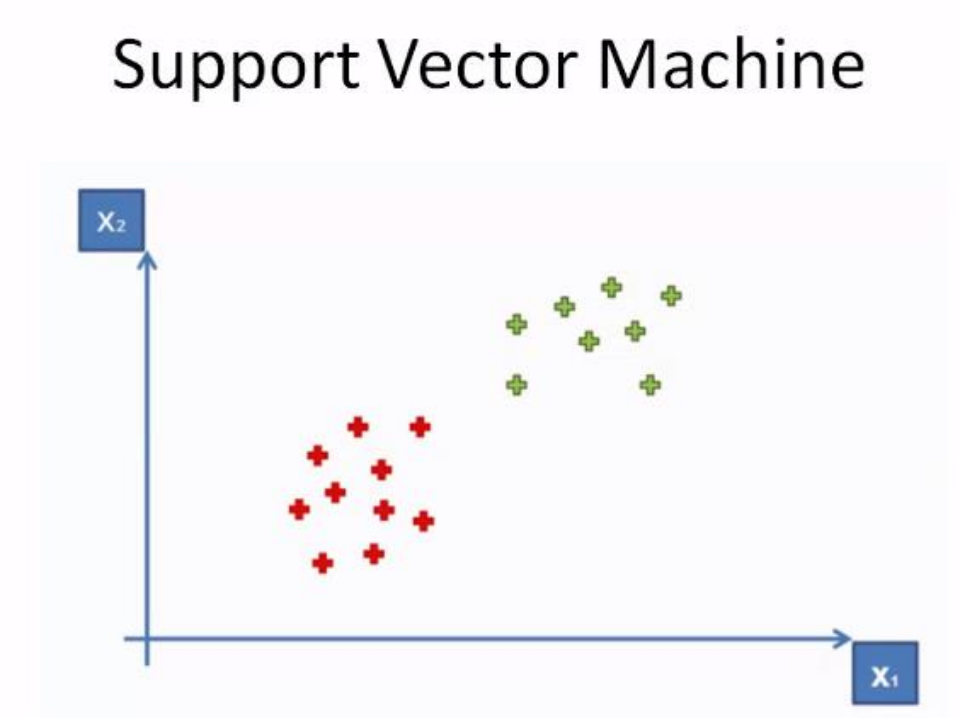
$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}.$$

We won't having scaling in regression as we didn't face imbalance situation there.

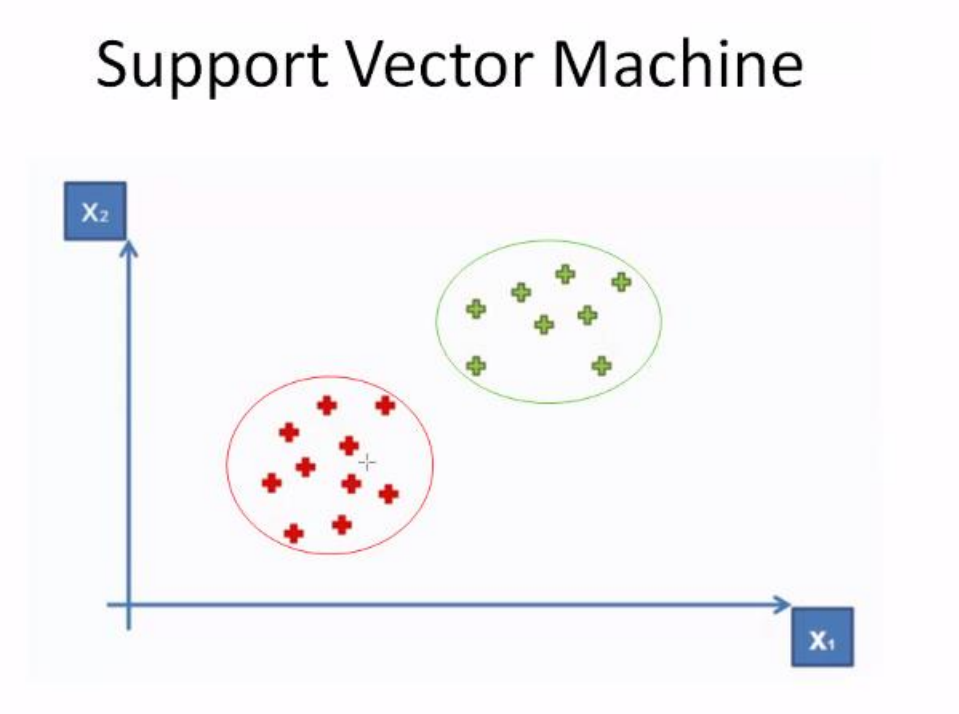
```
In [6]: np.min(X_train)
Out[6]: -1.9931891594584856

In [7]: np.max(X_train)
Out[7]: 2.3315320031817324

In [8]:
```

Pretty balanced! Not too low or high.

SVM:

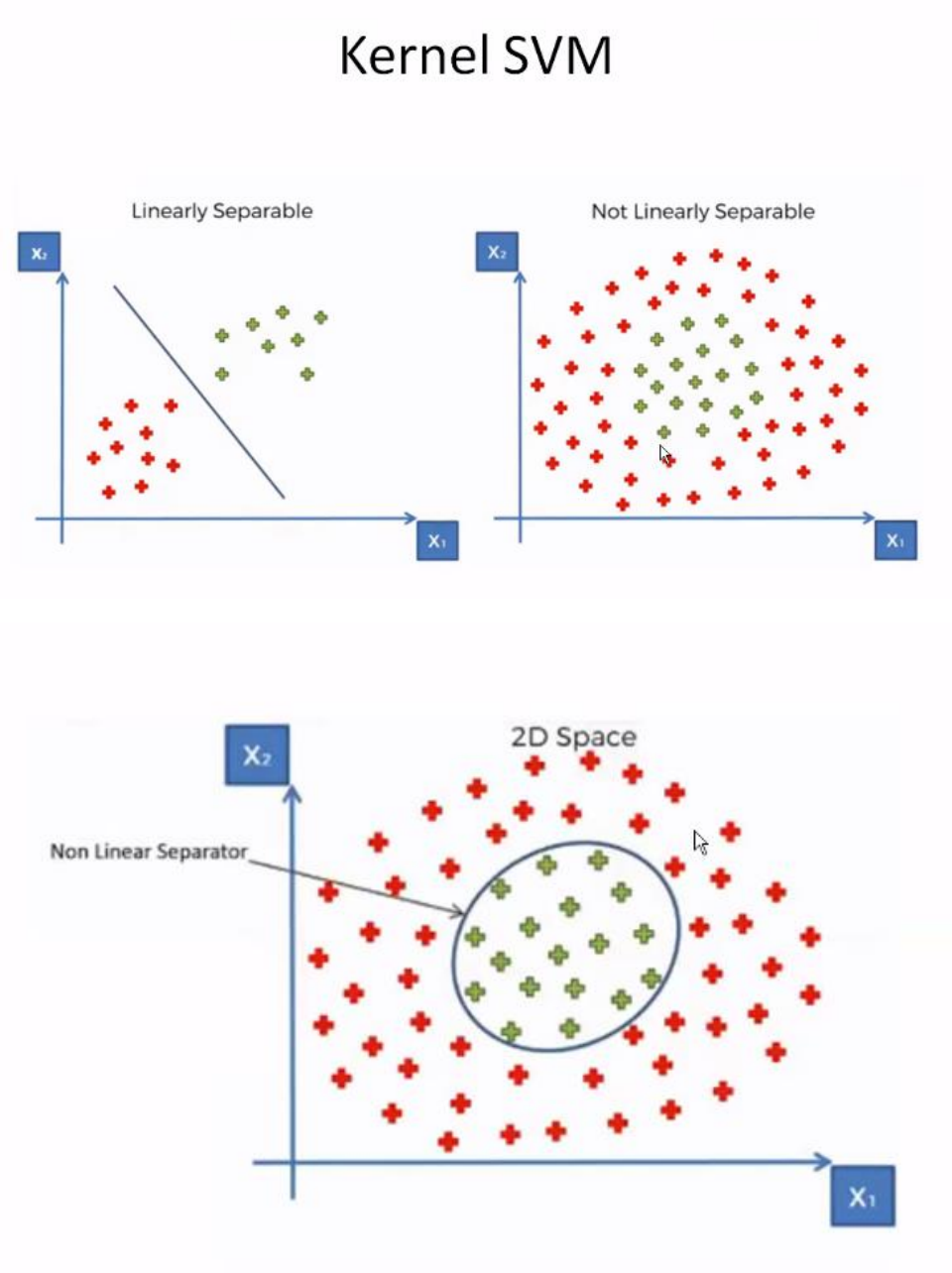


Learn by looking into the similarity and dissimilarity of the data.

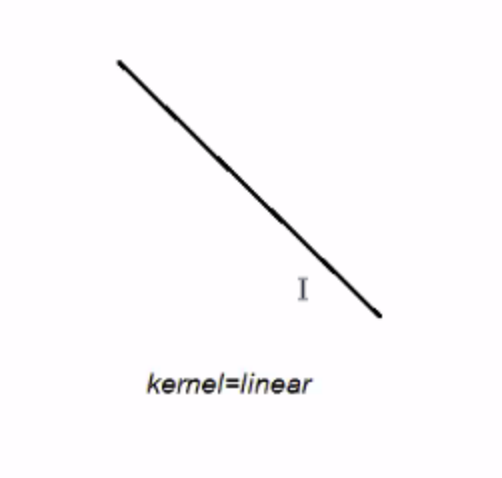


At boundaries, you can see data will not be similar.

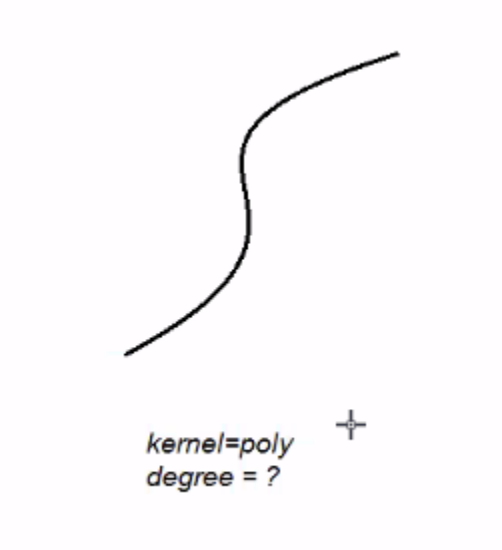
We will draw the hyperplane



Simple problem:

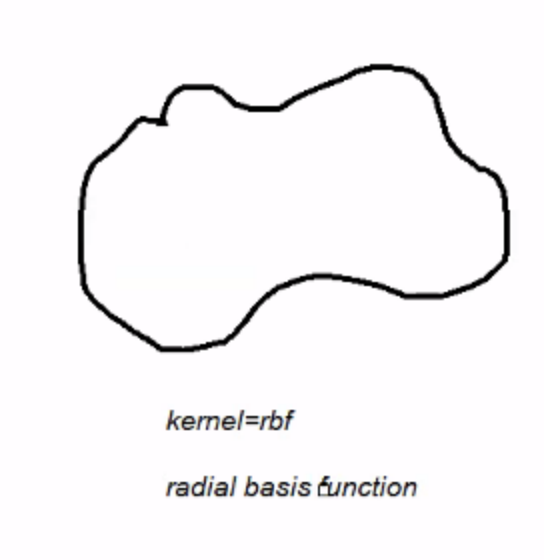


For complex problem: poly kernel



Didn’t serve long as data is more complex in real world

Radial basis function



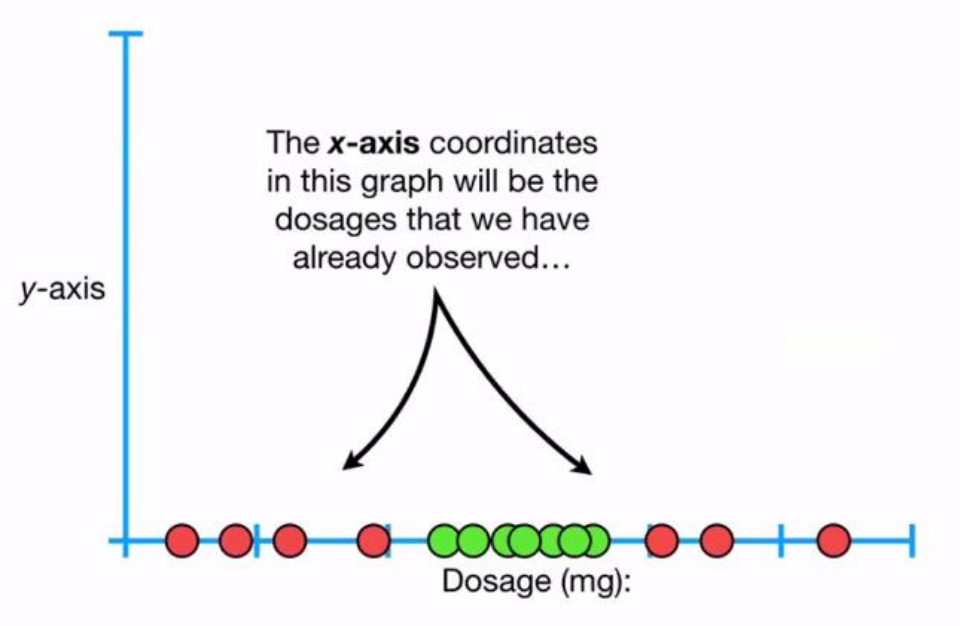
What is the math behind them?

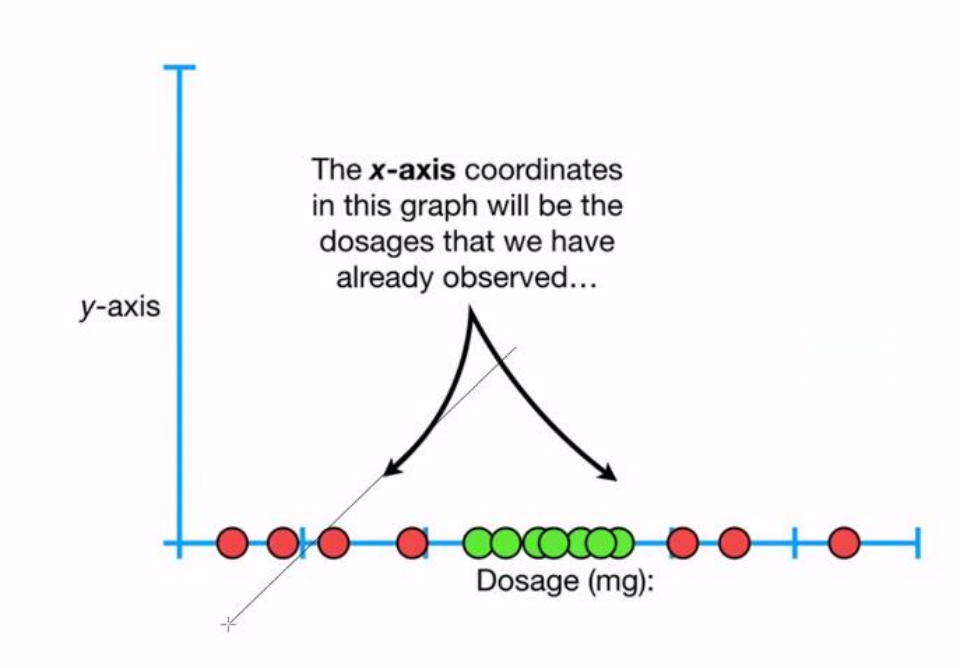
Example: Doctor prescribed the medicines with different dosages

Green – cured

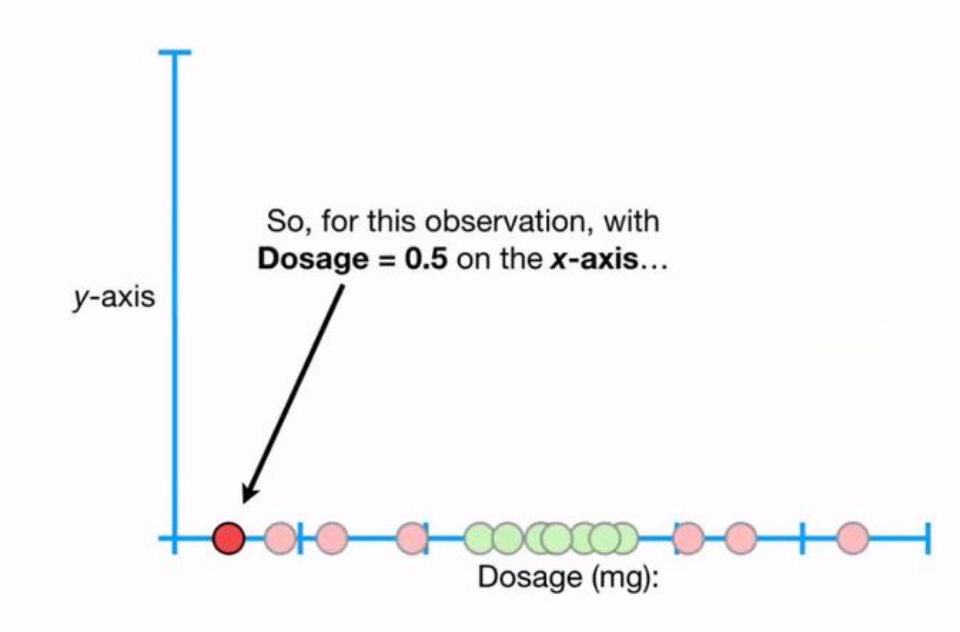
Red – not cured.

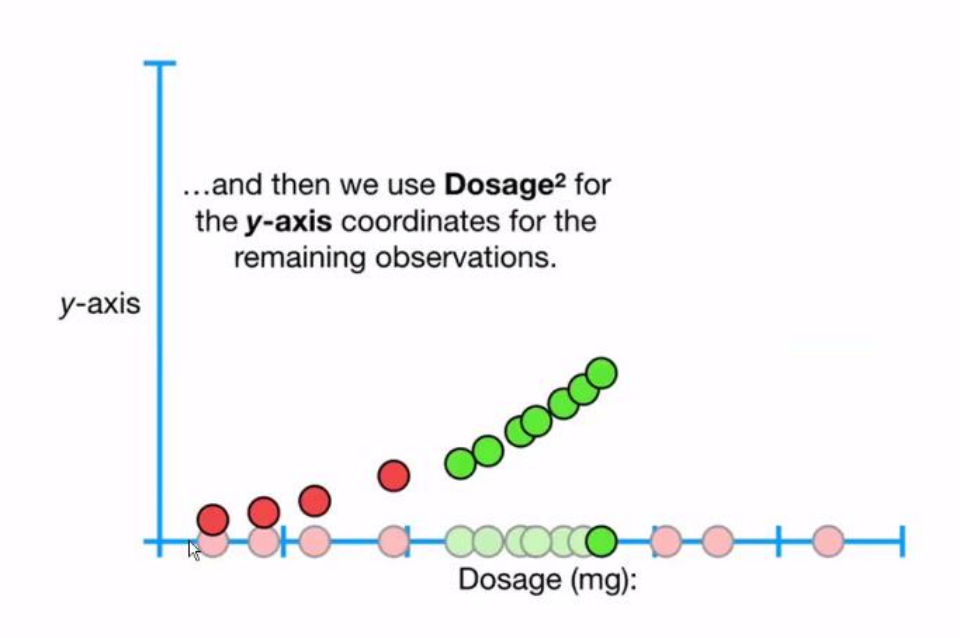
We need to draw the best fit line

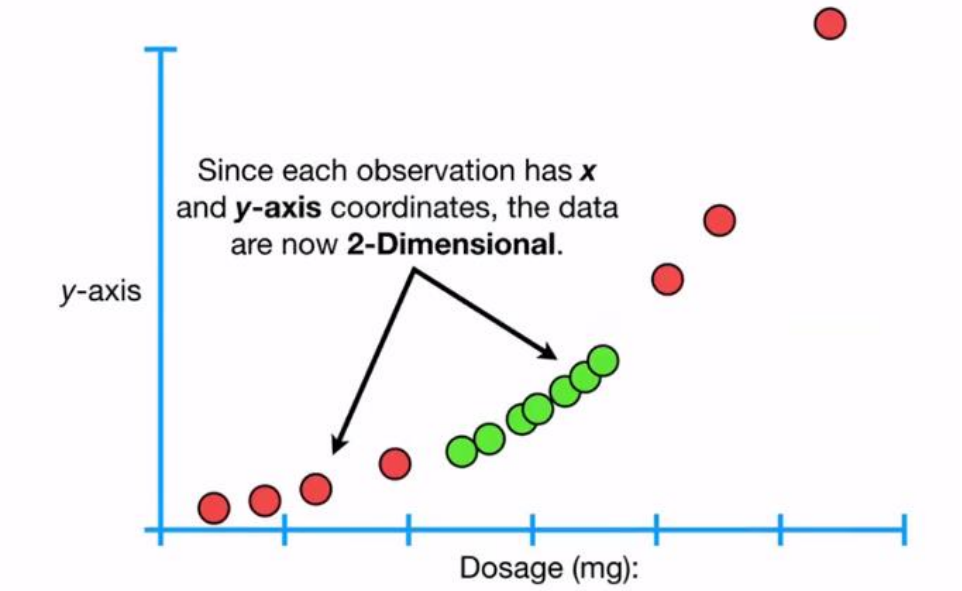


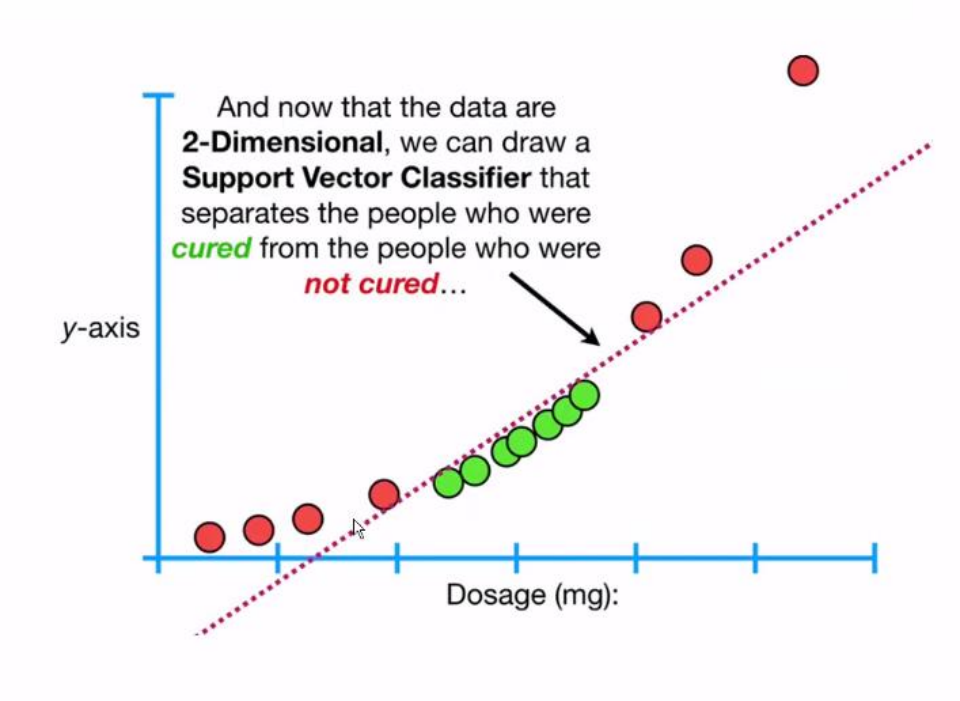


Pull into higher dimension

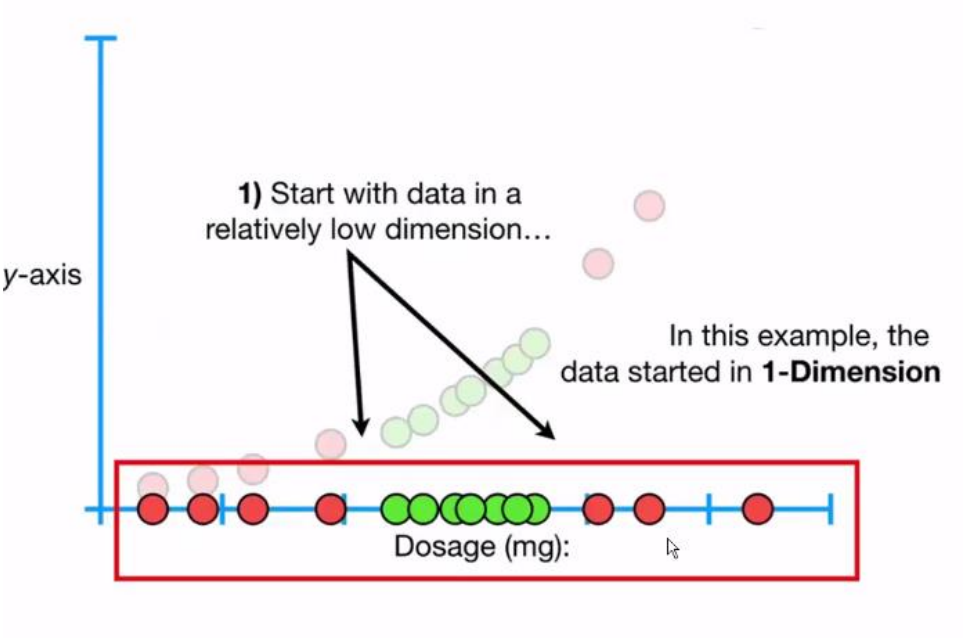


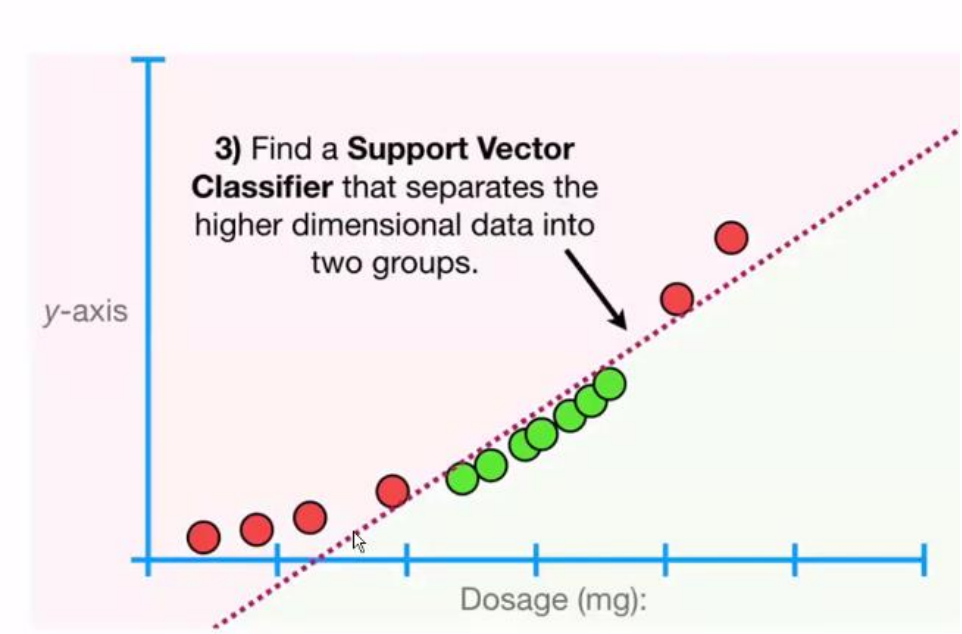

…and then we use **Dosage²** for the ***y*-axis** coordinates for the remaining observations.
y-axis
Dosage (mg):

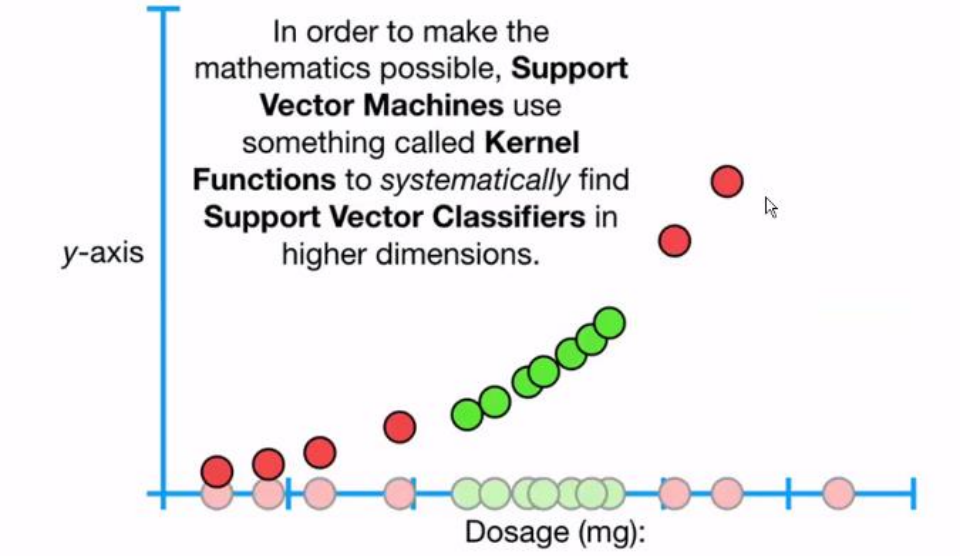

Since each observation has ***x*** and ***y*-axis** coordinates, the data are now **2-Dimensional**.
y-axis
Dosage (mg):

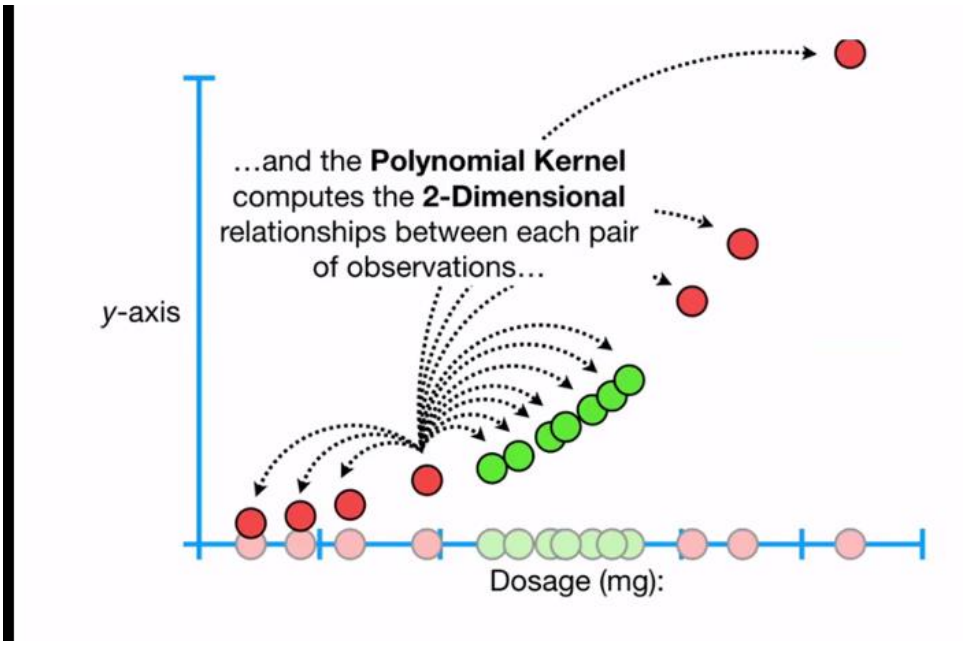


Taking the points which are very boarder of it. We will try to draw the lines (hyperplane)




3) Find a **Support Vector Classifier** that separates the higher dimensional data into two groups.
y-axis
Dosage (mg):


In order to make the mathematics possible, **Support Vector Machines** use something called **Kernel Functions** to *systematically* find **Support Vector Classifiers** in higher dimensions.
y-axis
Dosage (mg):



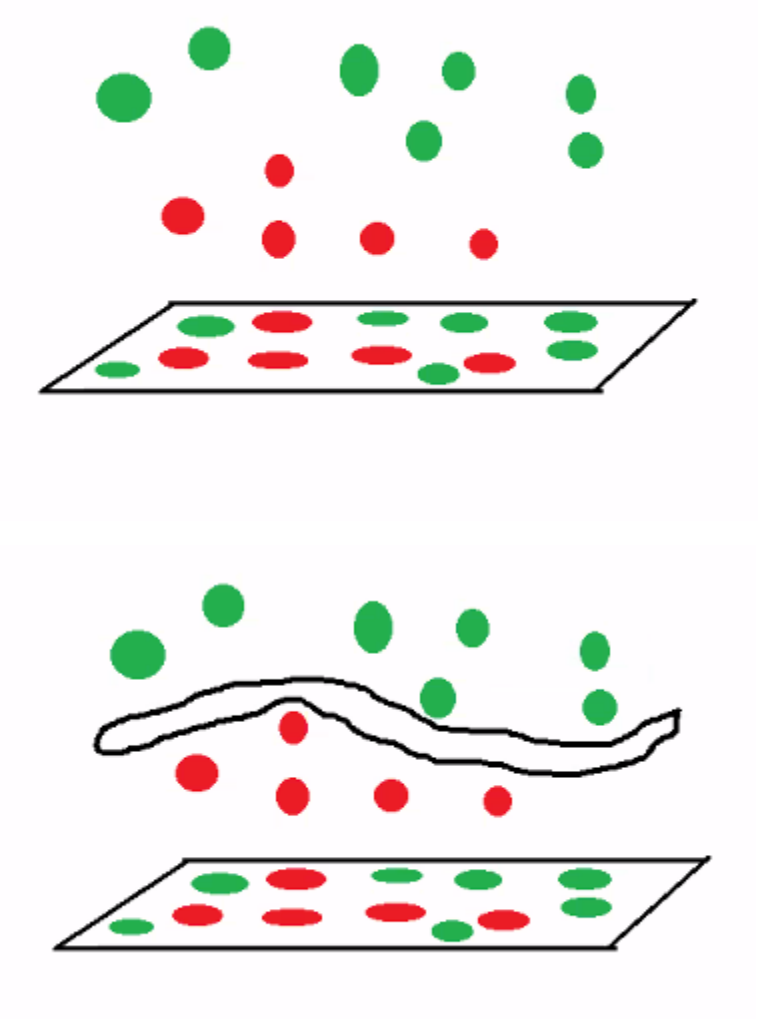
Another very commonly used **Kernel** is the **Radial Kernel**, also known as the **Radial Basis Function (RBF) Kernel**.

Radial Kernel finds **Support Vector Classifiers** in *infinite dimensions*

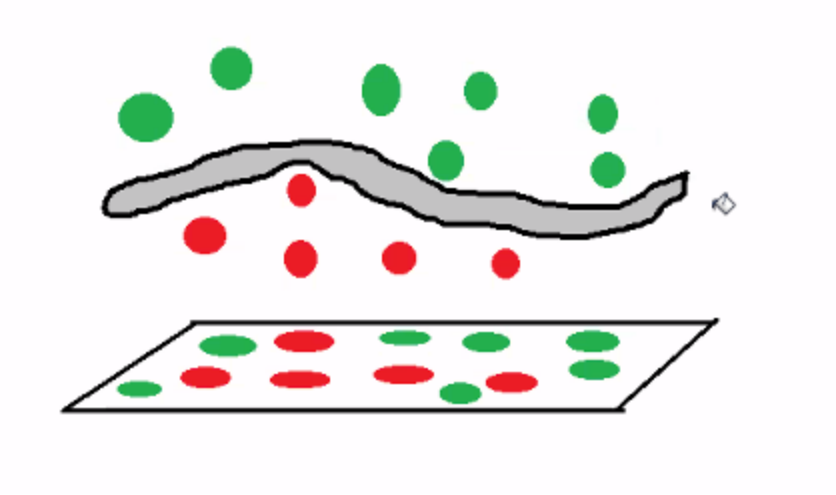
Datapoints:

Pull them into higher dimension space:

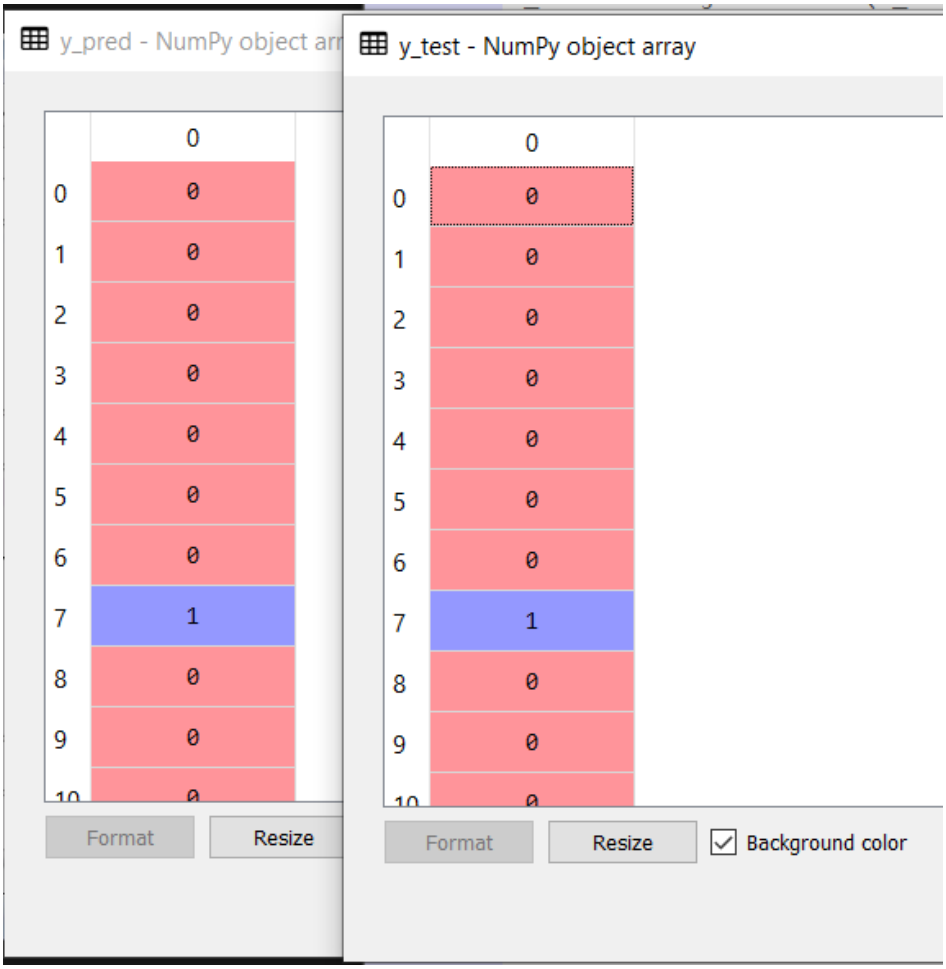
Floating in the air



Radial basis function: (RDF)



Default kernel is RBF.

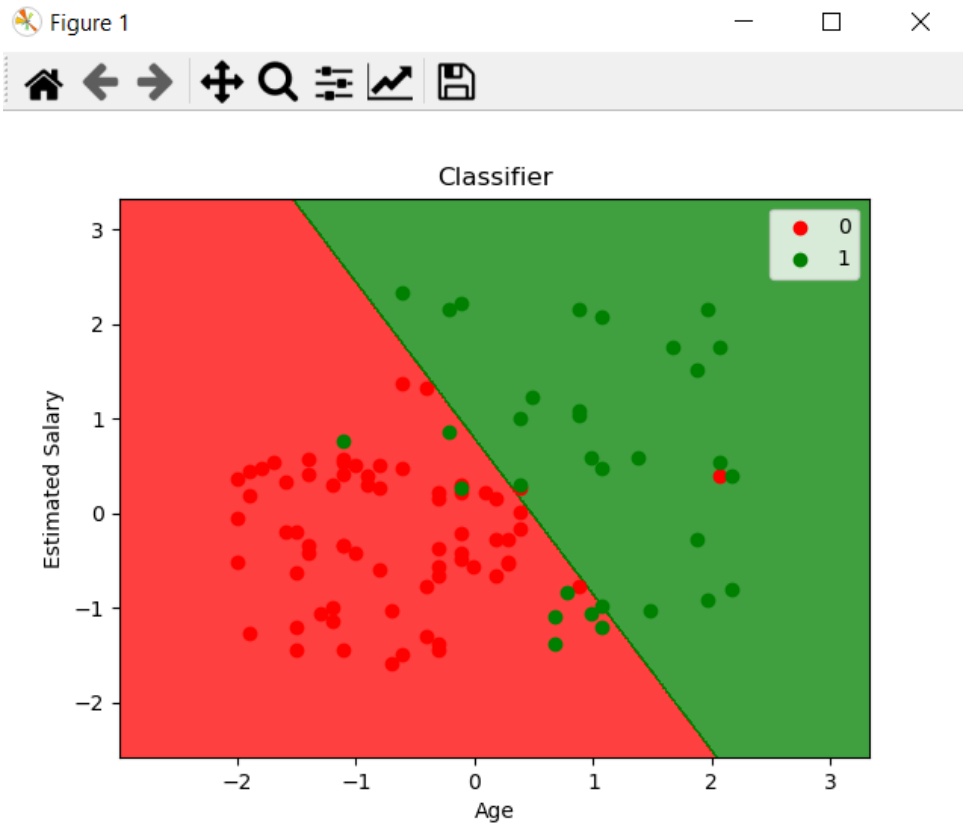


Accuracy:

acc	float64	1	0.9
classifier	svm._classes.SVC	1	SVC object of sklearn.svm._classes module

Visualization:

Linear:



Polynomial with degree 5:

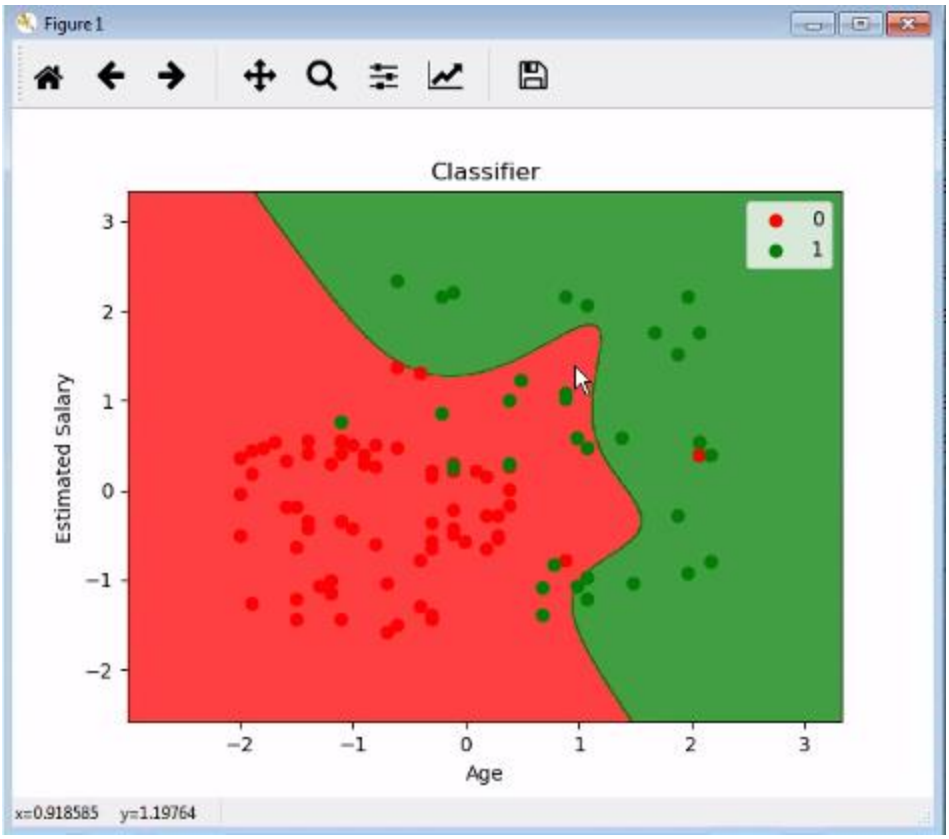


Figure 1
Classifier
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−2
−1
0
1
2
3
x=0.918585 y=1.19764

RBF:

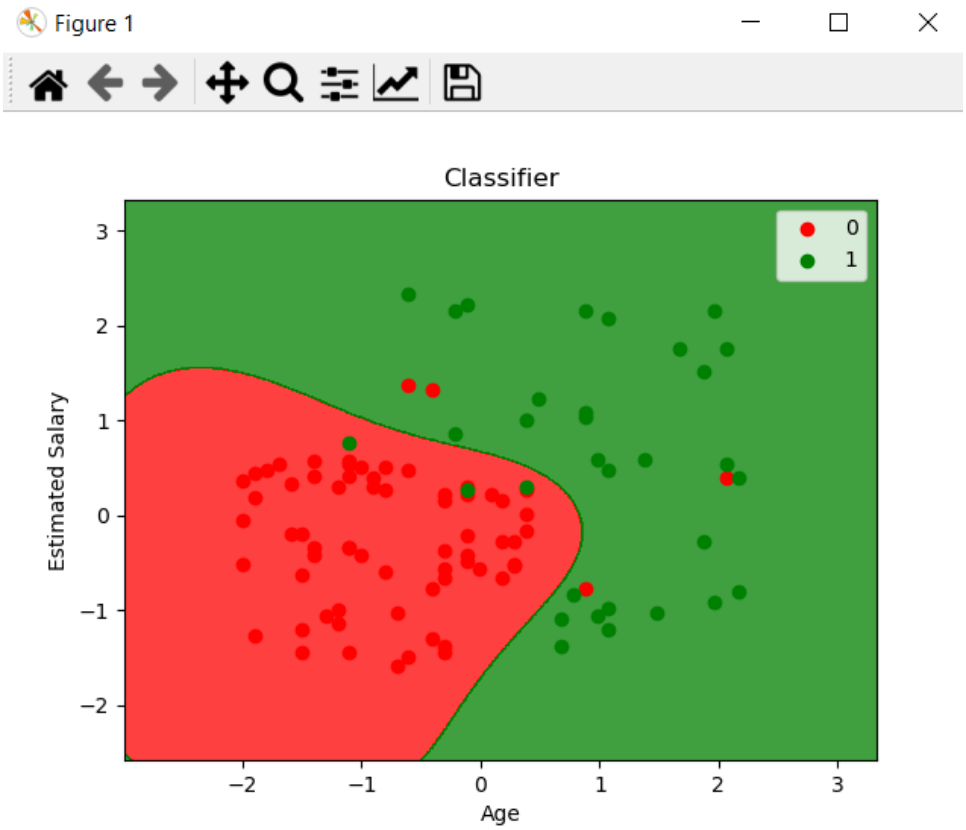


Figure 1
Classifier
0
1
Estimated Salary
Age
3
2
1
0
−1
−2
−2
−1
0
1
2
3

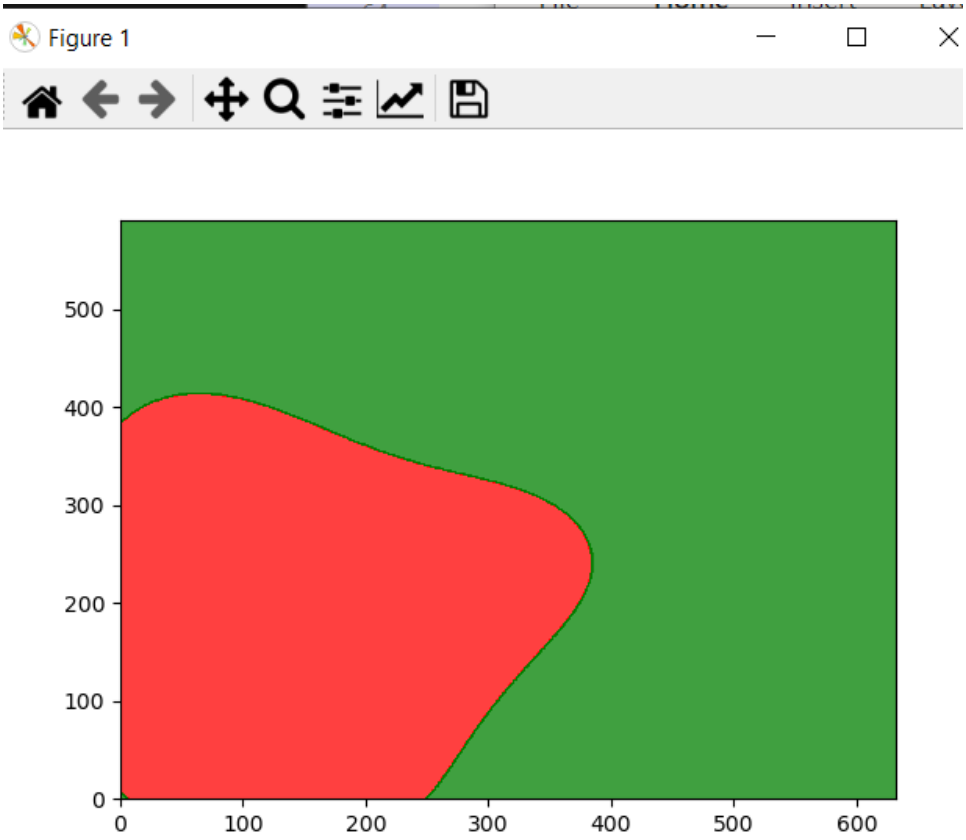
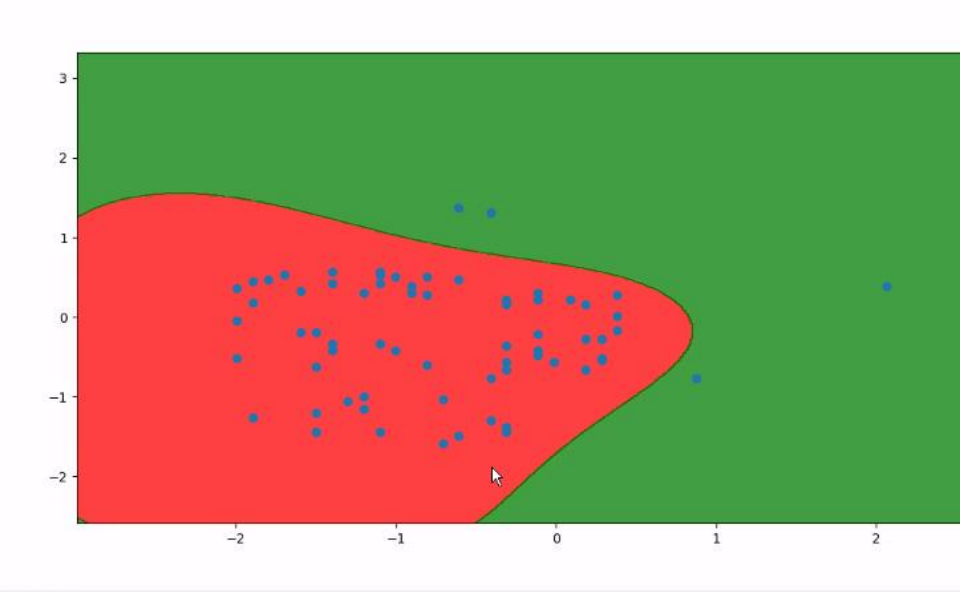


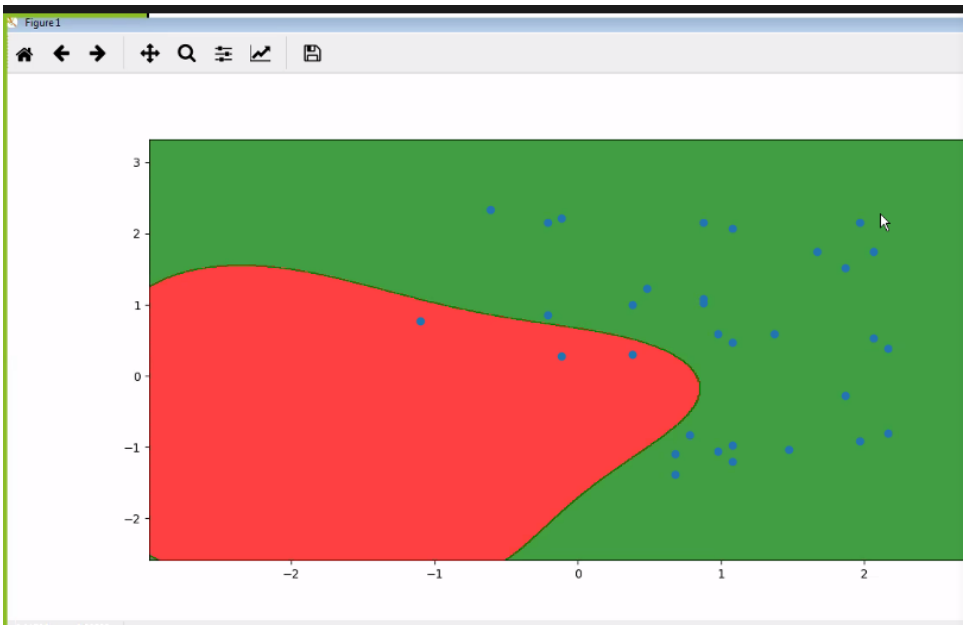
Figure 1
500
400
300
200
100
0
0
100
200
300
400
500
600

False cases:

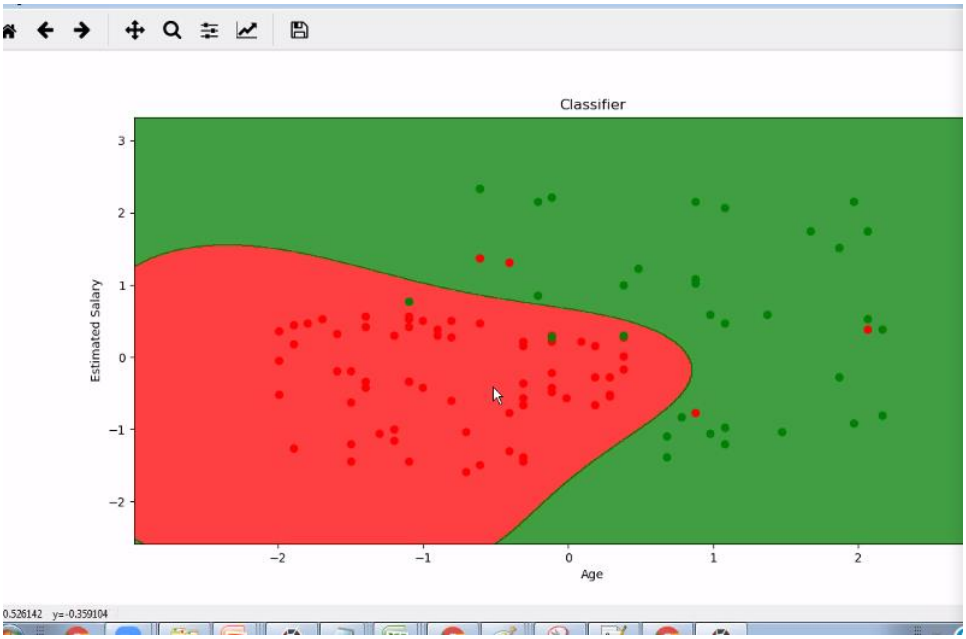



```
X1, X2 = np.meshgrid(np.ara ge(start=X_set[:,0].min()-1,stop=X_set[:,1].max()+1,step=0 01),
                     np.ara ge(start=X_set[:,1].min()-1,stop=X_set[:,1].max()+1,step=0 01))

plt.contourf(X1,X2,
             classifier.pre ict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape)
             alpha = 0.75 , cmap = ListedColormap(('red','green')))

plt.scatter(X_test[y_test== ,0],X_test[y_test==1,1])
plt.show()

plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())

for i, j in enumerate(np.un que(y_set)):
    plt.scatter(X_set[y_set == j , 0], X_set[y_set == j, 1],
                c=ListedCol rmap(('red','green'))(i), label = j)
plt.title('Classifier')
plt.xlabel('Age')
plt.ylabel('Estimated Salar ')
plt.legend()
plt.show()
```

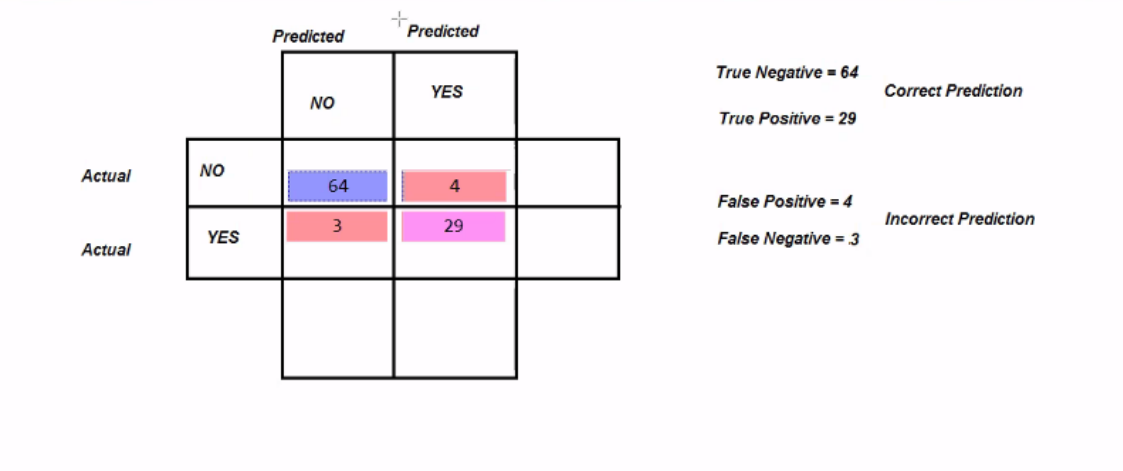
True cases:



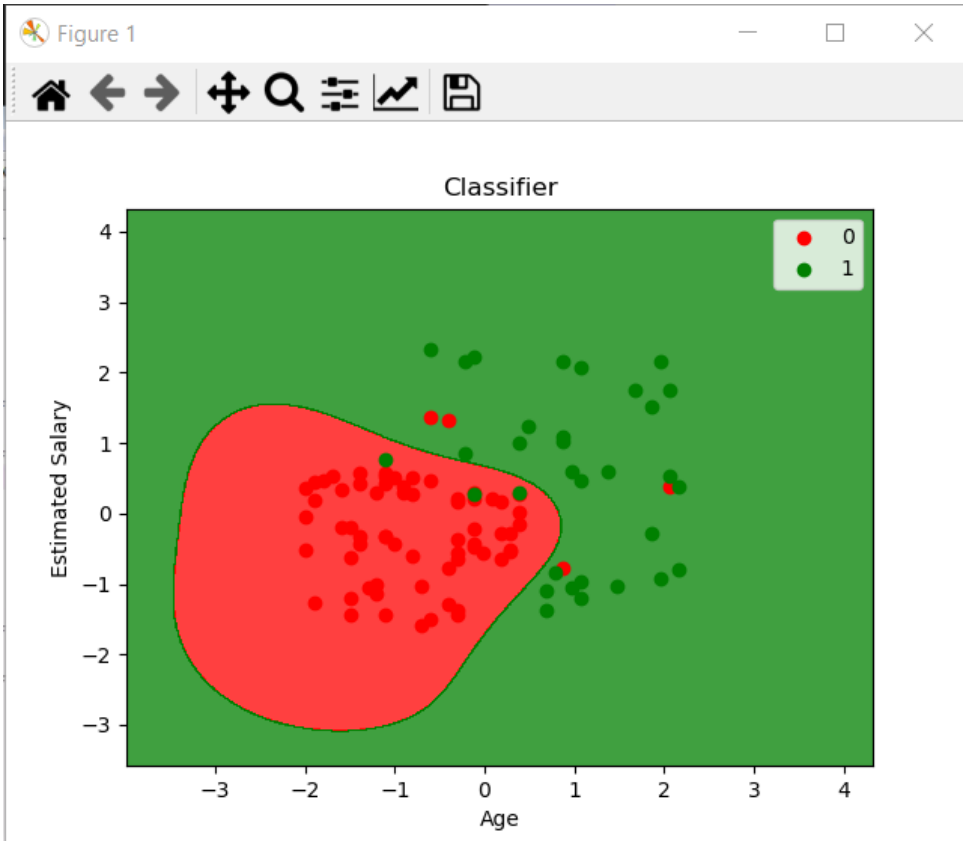
All cases:

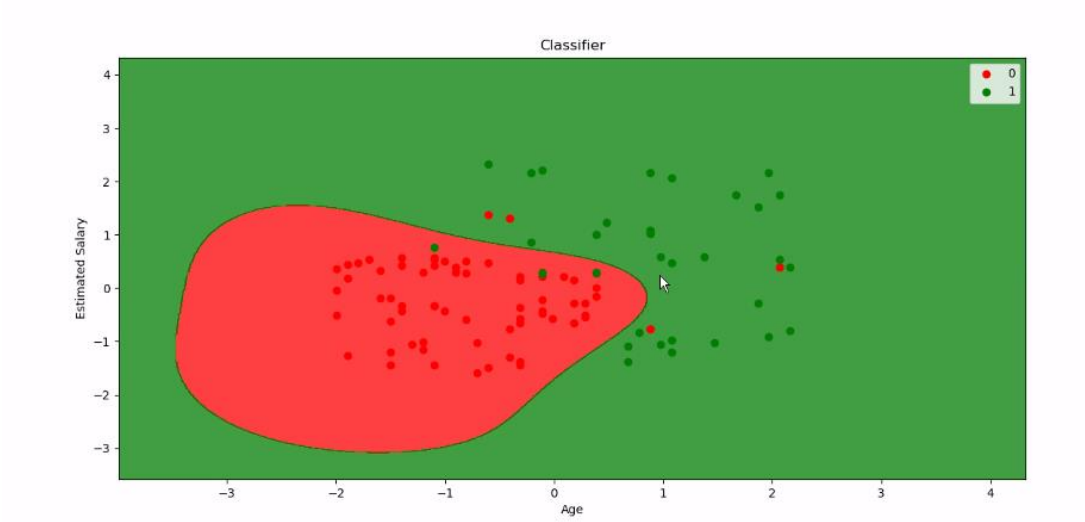


Confusion matrix:



Increased the scale:

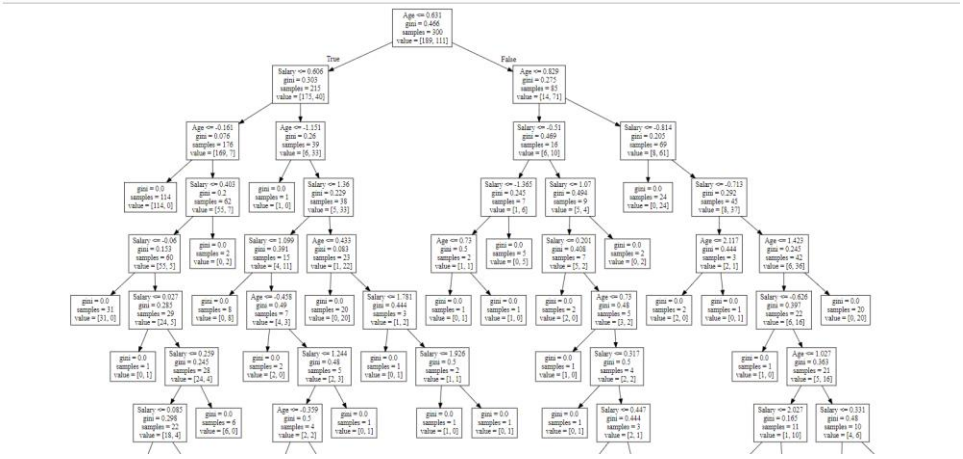




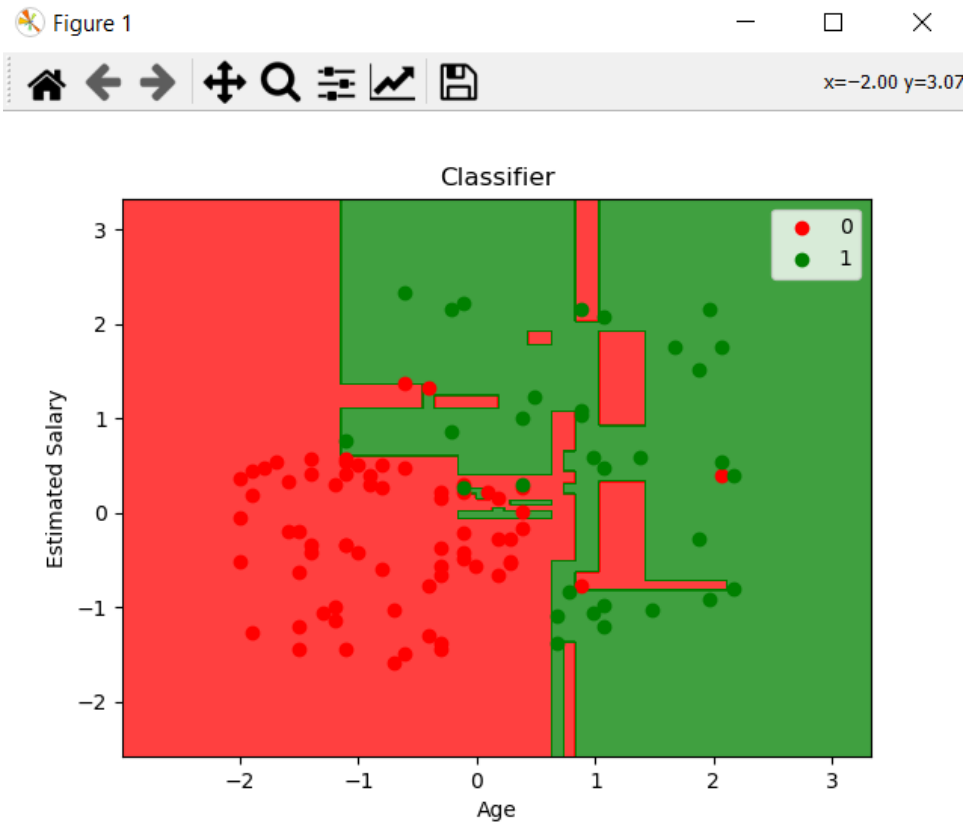

```
from matplotlib.colors import ListedColormap

X_set, y_set = X_test,y_test

X1, X2 = np.meshgrid(np.arange(start=X_set[:,0].min()-1,stop=X_set[:,1].max()+1,step=0.01),
                     np.arange(start=X_set[:,1].min()-1,stop=X_set[:,1].max()+2,step=0.01))

plt.contourf(X1,X2,
             classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75 , cmap = ListedColormap(('red','green')))

plt.xlim(X1.min(),X1.max())
plt.ylim(X2.min(),X2.max())

for i, j in enumerate(np.unique(y_set)):
```

Decision Tree Classifier:

Impurity calculation tells how well it perform no and yes response – Gini index

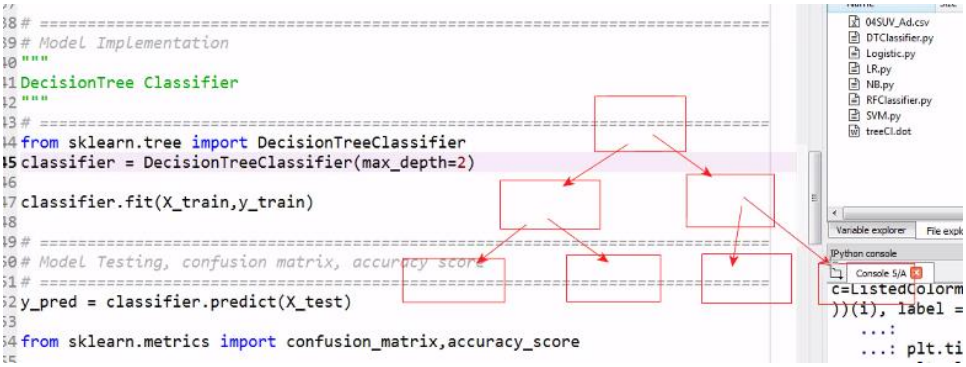


Visualization:

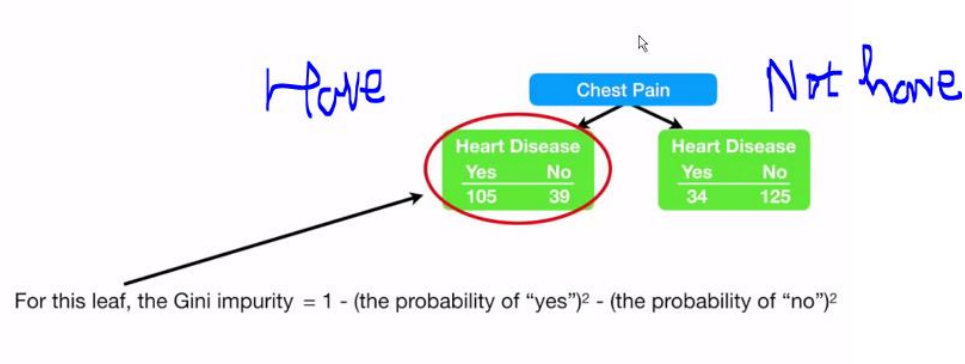


Boundary is not generalized. To control this, we will have minimum split => max depth

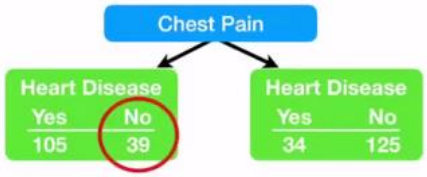
Control the depth here



Gini impurity calculation:

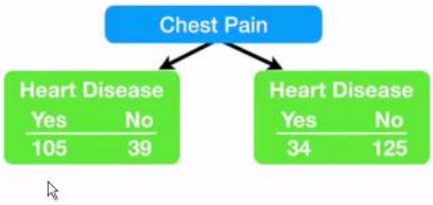


For this leaf, the Gini impurity = 1 - (the probability of "yes")^2 - (the probability of "no")^2



For this leaf, the Gini impurity = 1 - (the probability of "yes")^2 - (the probability of "no")^2

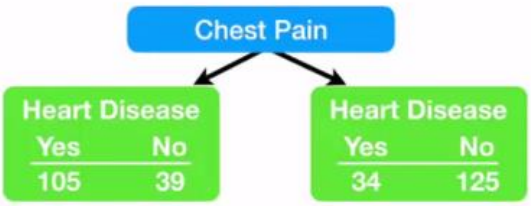
$$= 1 - \left(\frac{105}{105 + 39}\right)^2 - \left(\frac{39}{105 + 39}\right)^2$$



For this leaf, the Gini impurity = 1 - (the probability of "yes")^2 - (the probability of "no")^2

$$= 1 - \left(\frac{105}{105 + 39}\right)^2 - \left(\frac{39}{105 + 39}\right)^2$$

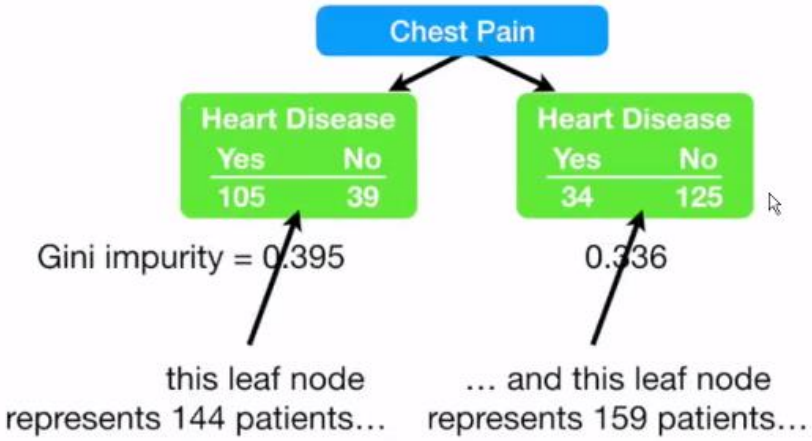
$$= 0.395$$



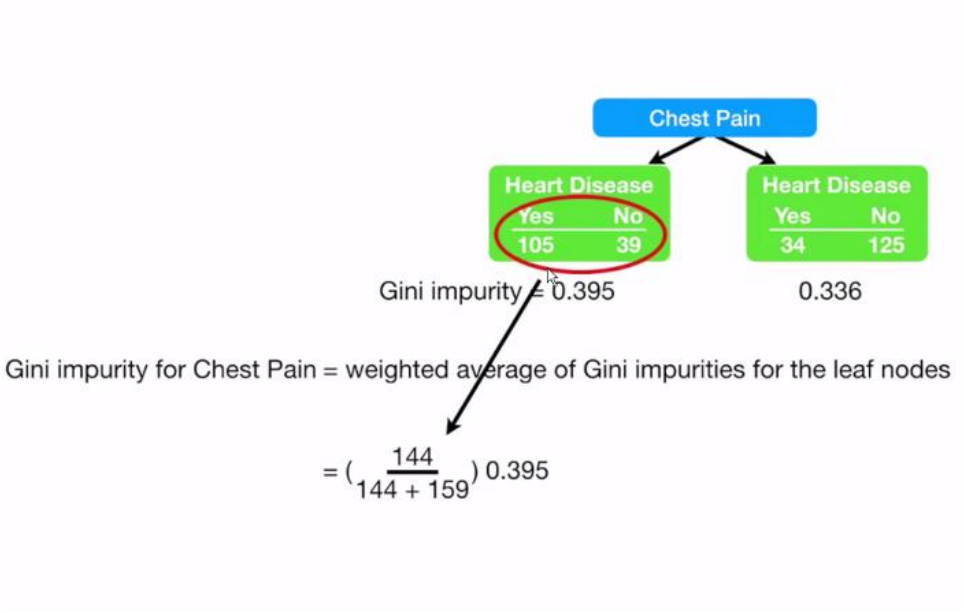
= 1 - (the probability of "yes")^2 - (the probability of "no")^2

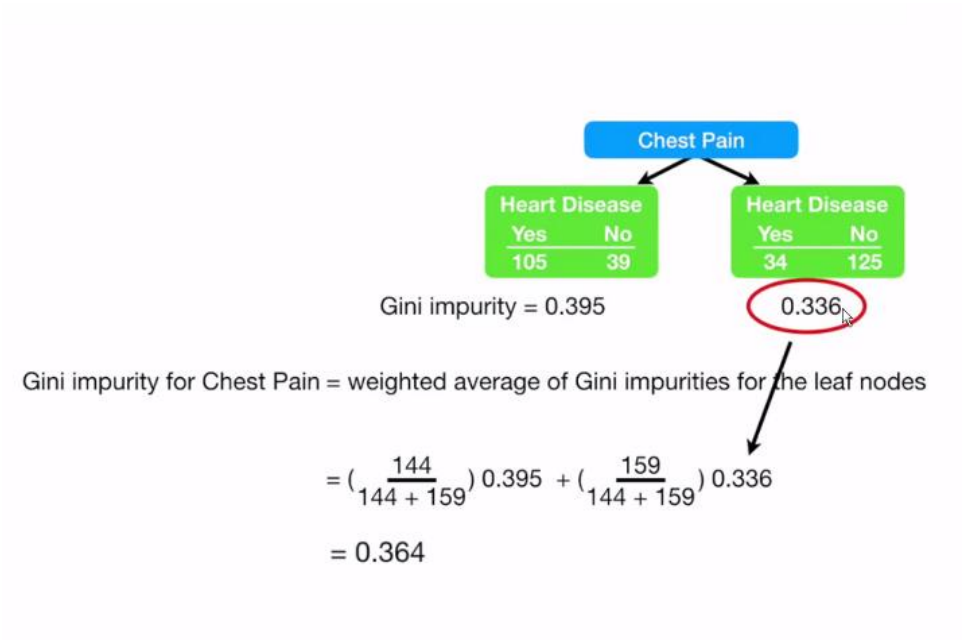
$$= 1 - \left(\frac{34}{34 + 125}\right)^2 - \left(\frac{125}{34 + 125}\right)^2$$

= 0.336

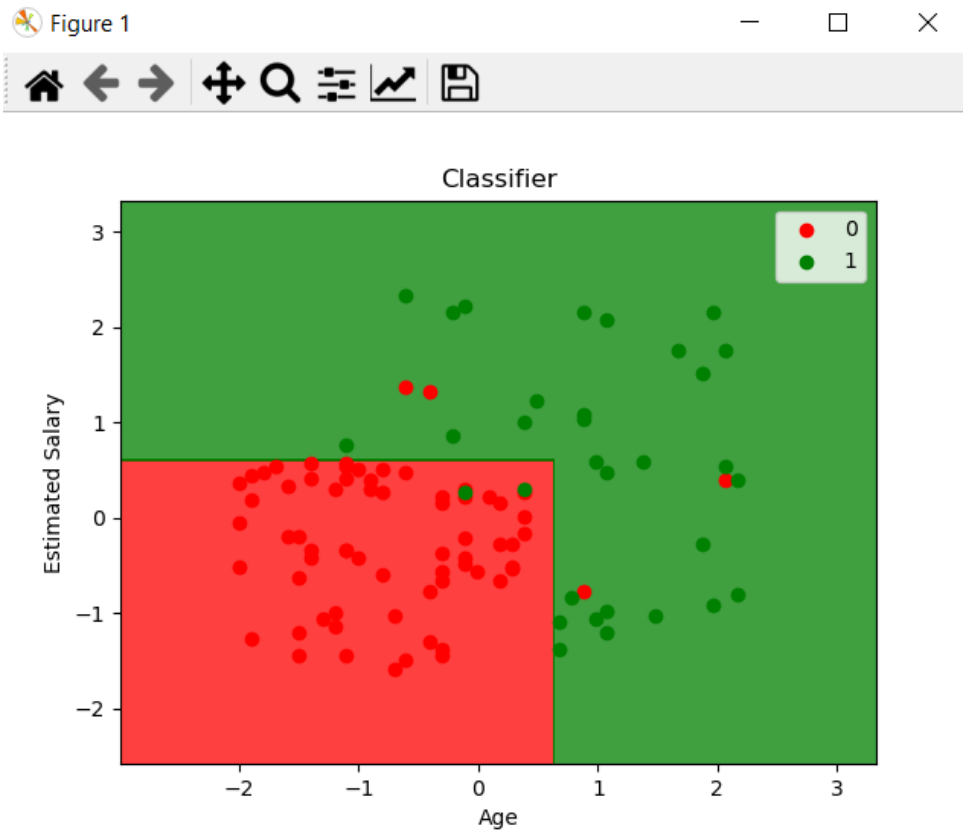


…the leaf nodes do not represent the same number of patients.



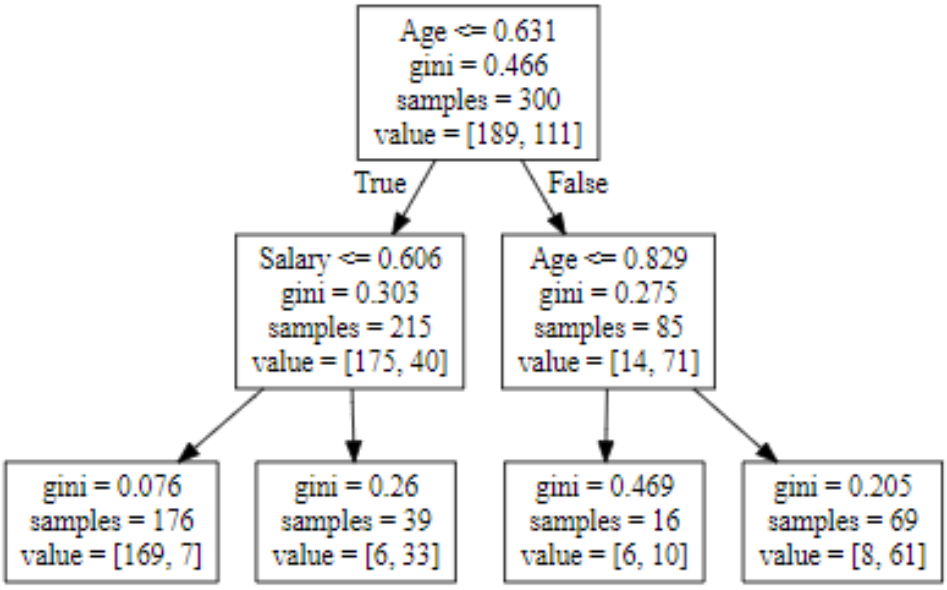


Increase the max depth = 2:



Boundary seems to be generalized.

Good decision tree:



Classification Value to Predict is Discrete

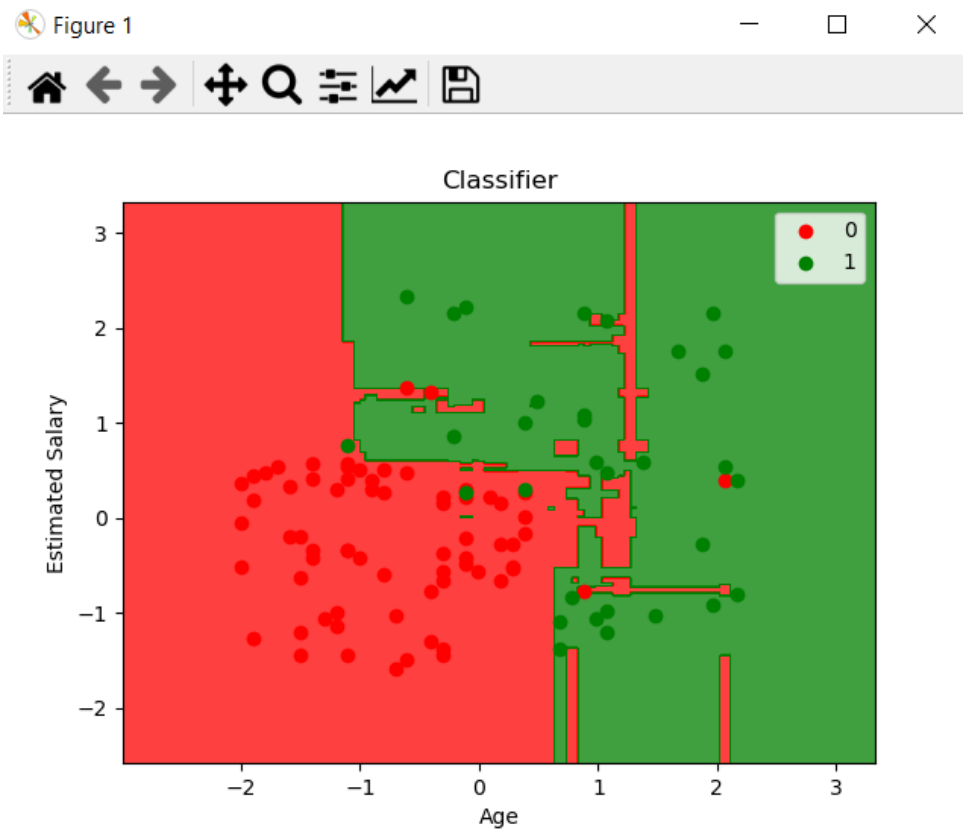
Logistic Regression		
Support Vector Machine	93%	
DecisionTree Classifier	94%	max_depth=2
RandomForest Classifier		
Naive Bayes		

Random Forest Classifier:

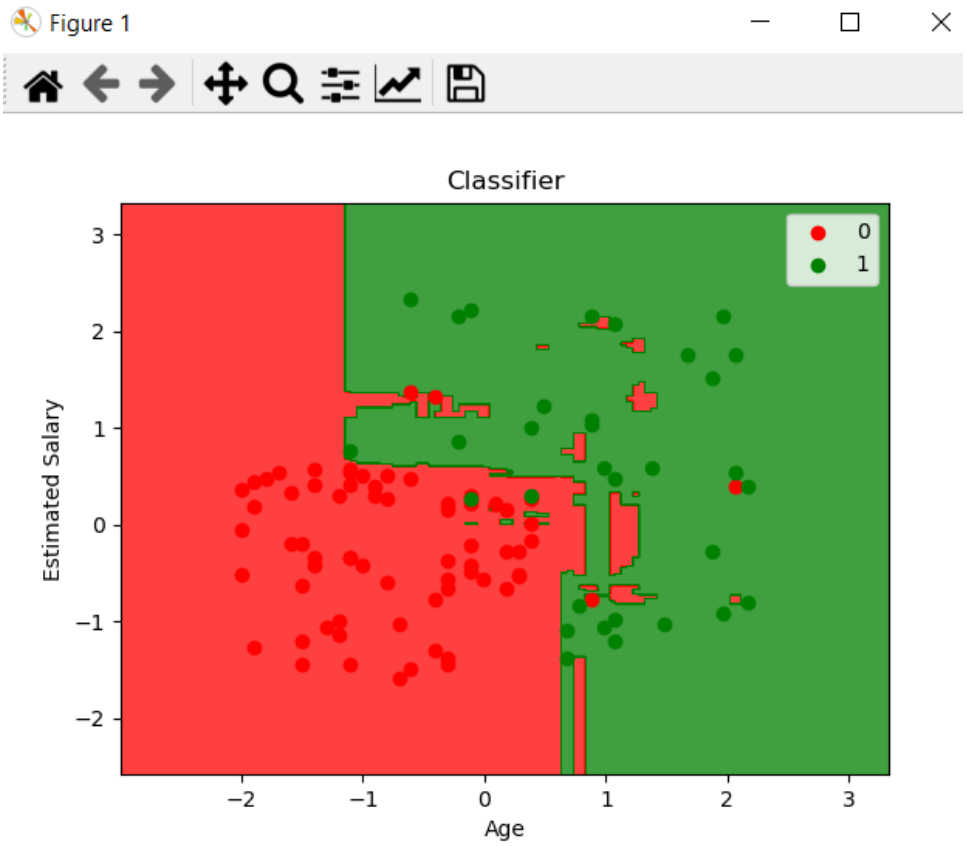
Name	Type	Size	Value
acc	float64	1	0.92
classifier	ensemble._forest.RandomForestClassifier	10	RandomForestClassif…

Visualization:

Overfitting issue!



Change it to 100 trees:

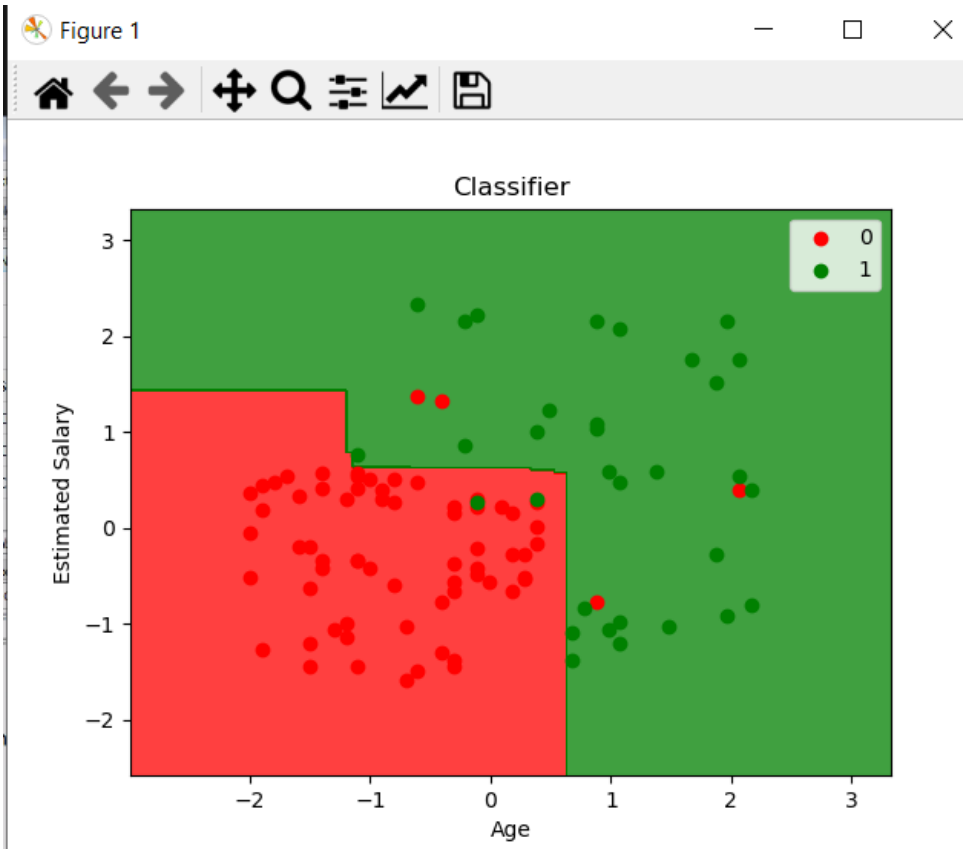


Name	Type	Size	Value
acc	float64	1	0.92
classifier	ensemble._forest.RandomForestClassifier	100	RandomForestClass

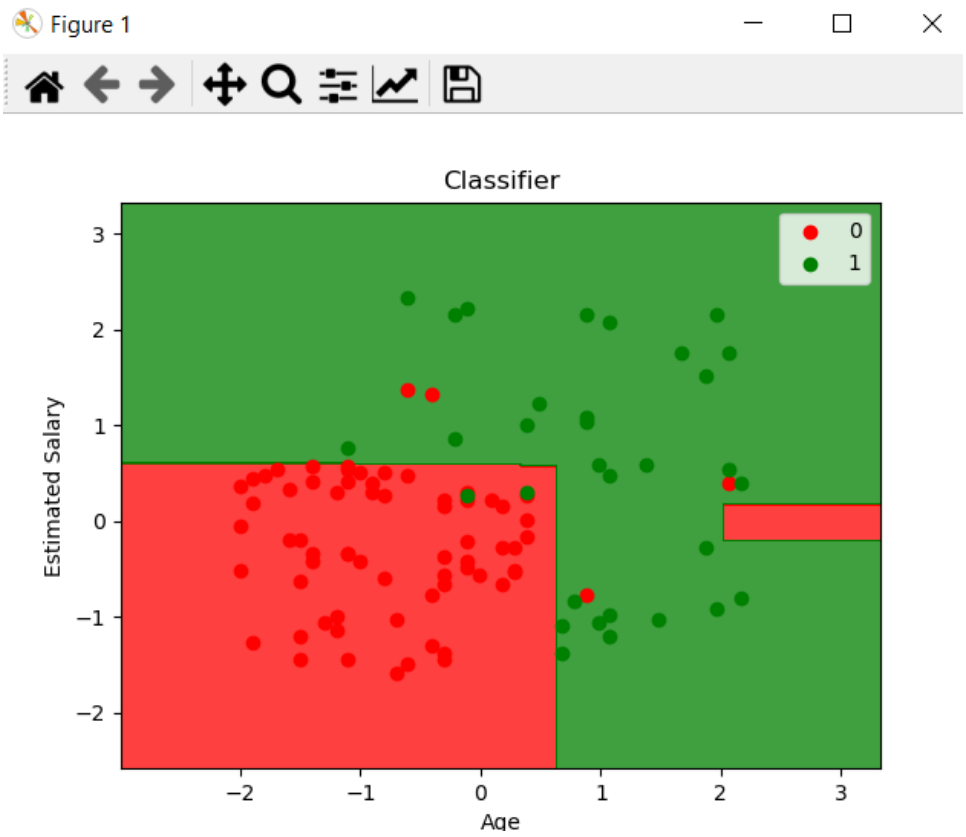
Not helping! Same accuracy

Try to include max_depth=2:

With 100 trees:



With 10 trees:



Name	Type	Size	Value
acc	float64	1	0.94
classifier	ensemble._forest.RandomForestClassifier	100	RandomForestClass…

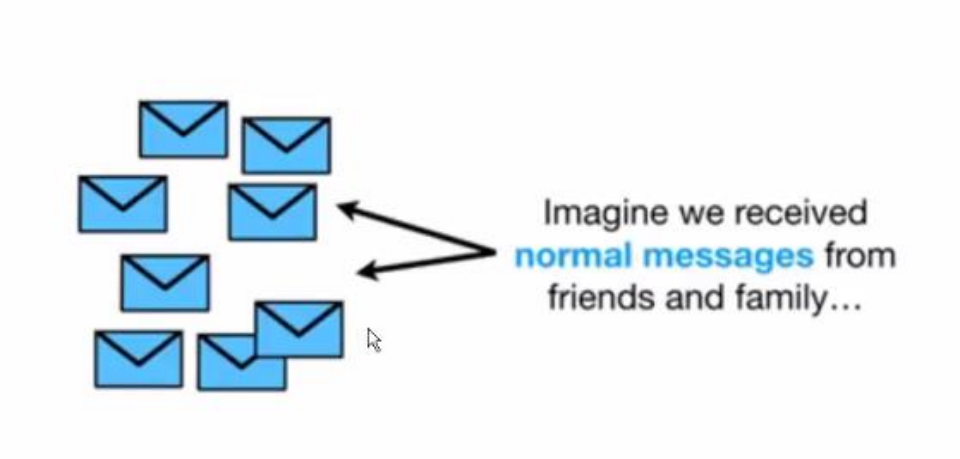
Name	Type	Size	Value
acc	float64	1	0.94
classifier	ensemble._forest.RandomForestClassifier	10	RandomForestClass…

Classification Value to Predict is Discrete

Logistic Regression		
Support Vector Machine	93%	
DecisionTree Classifier	94%	max_depth=2
RandomForest Classifier	94%	max_depth=2 ,Trees=10
Naive Bayes		

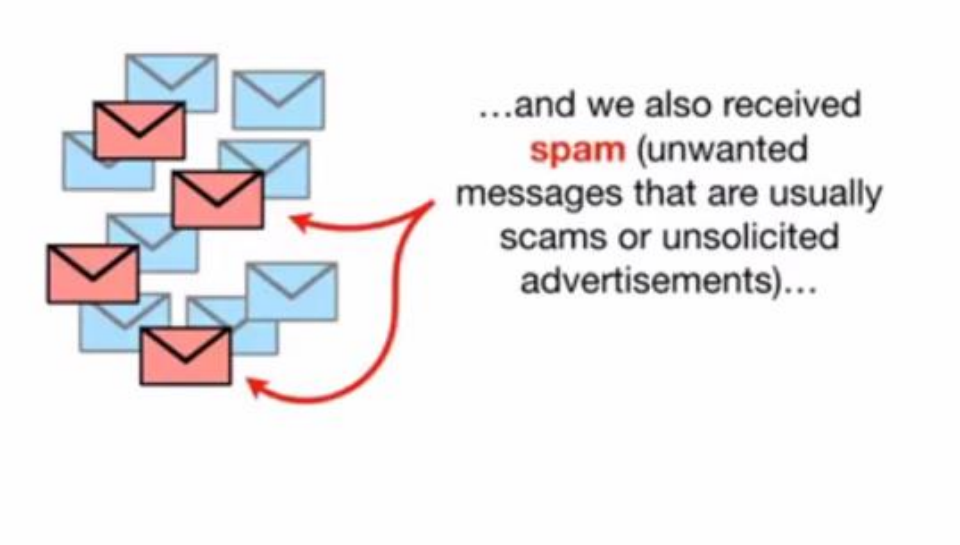
Naïve Bayes:

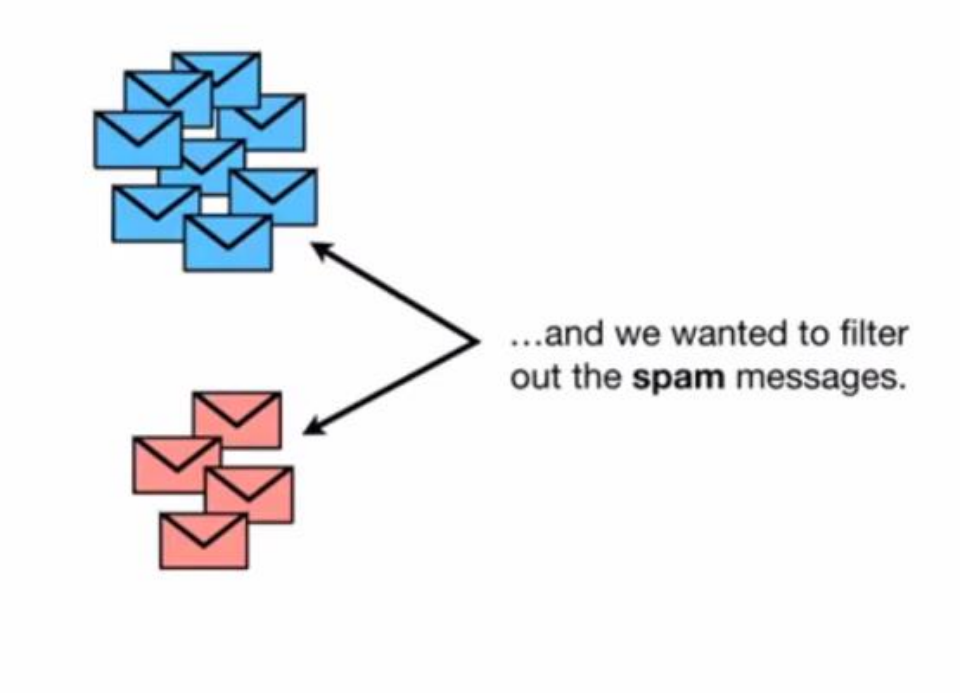
Learned based on probabilities.



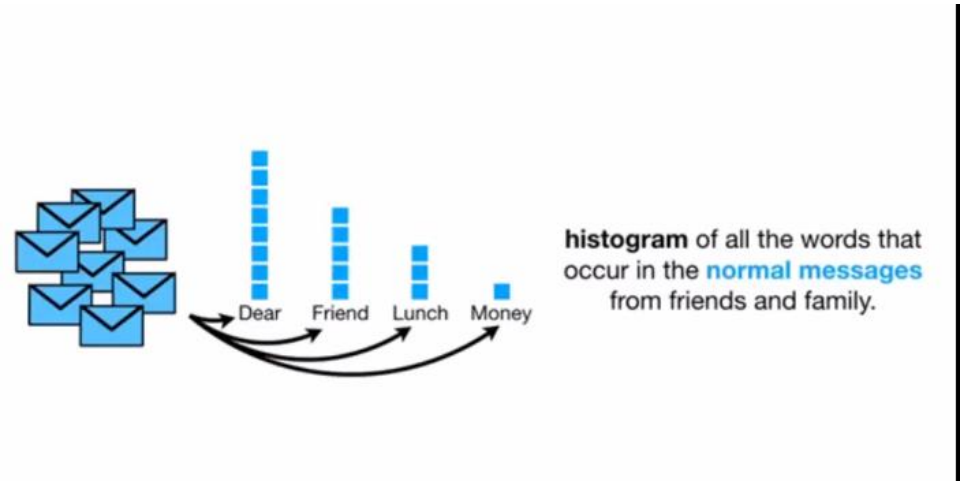
Create spam filter

Segreate normal and spam messages.

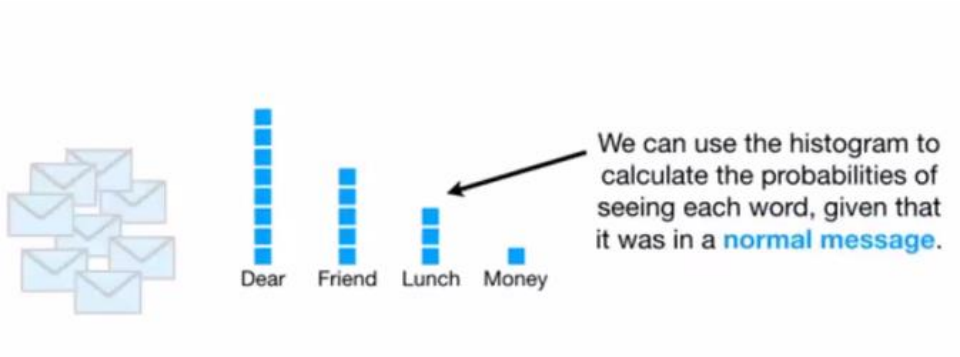


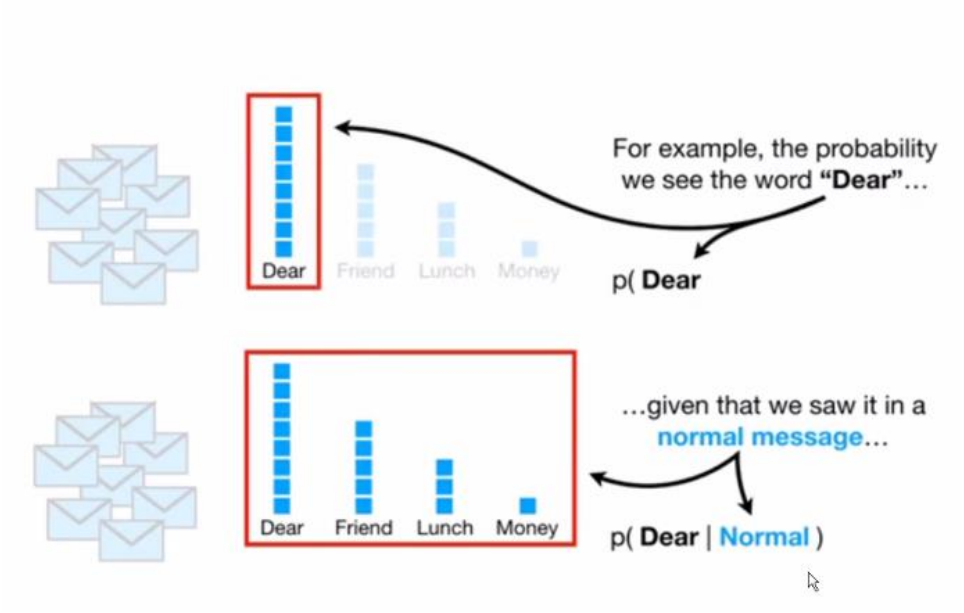


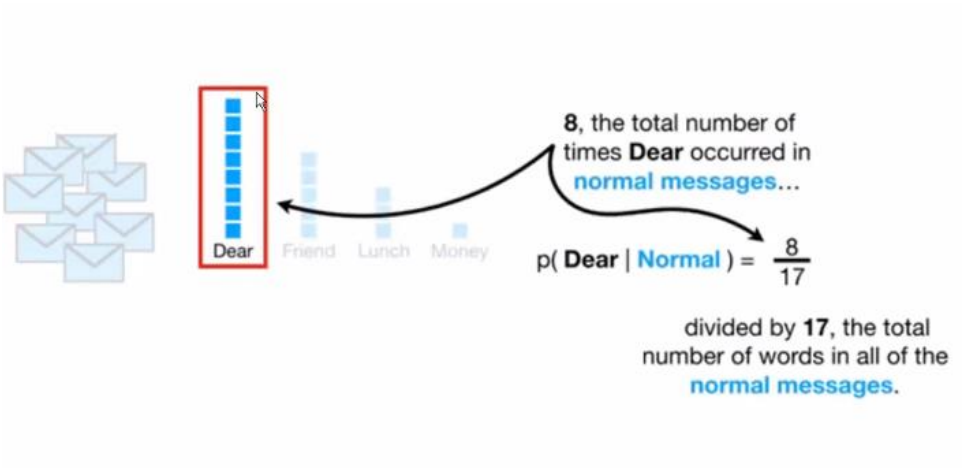
Find probabilities for each words:

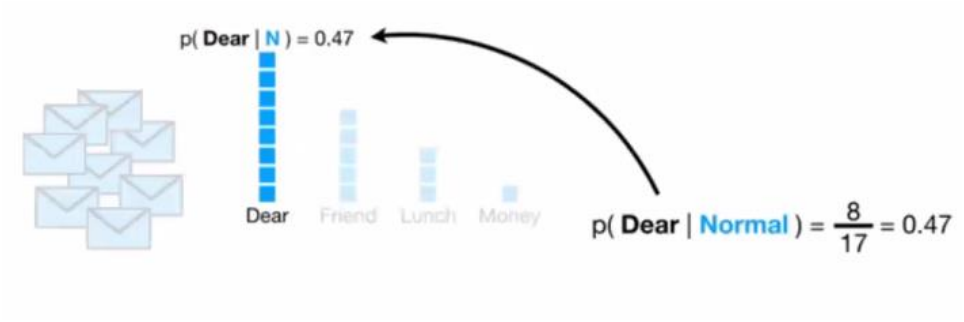


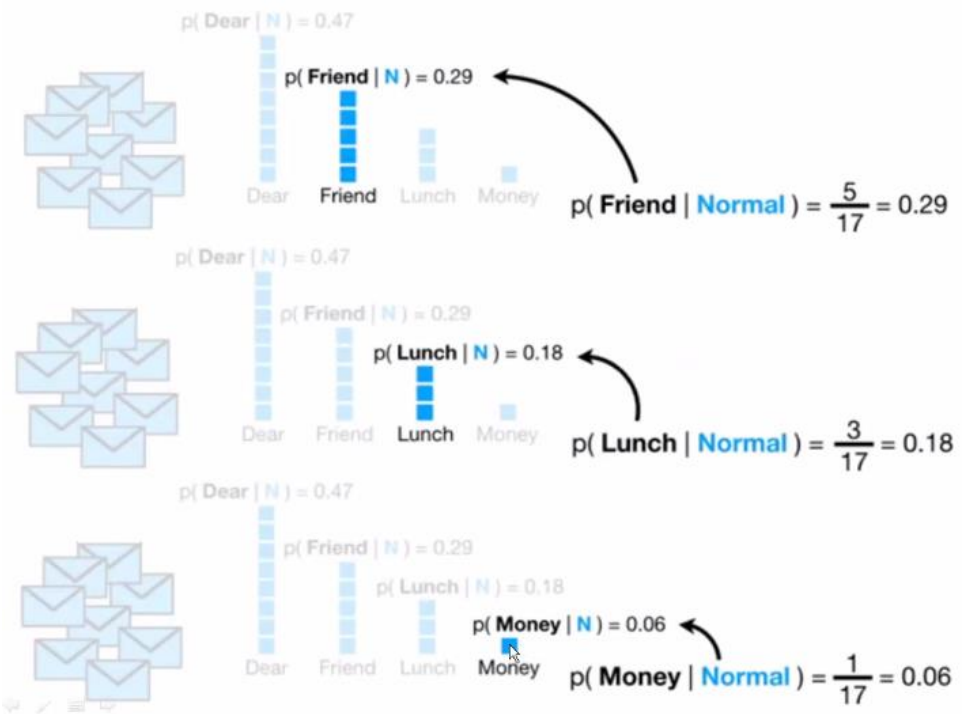
Find the probabilities of words given the visibilities of normal messages



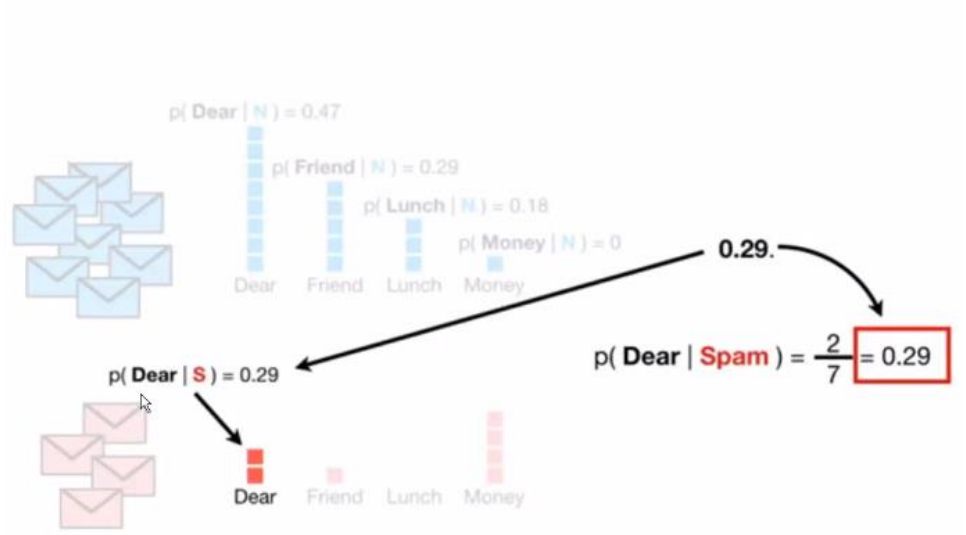

Dear
Friend
Lunch
Money
For example, the probability we see the word "Dear"...
p(Dear
Dear
Friend
Lunch
Money
...given that we saw it in a normal message...
p(Dear | Normal)

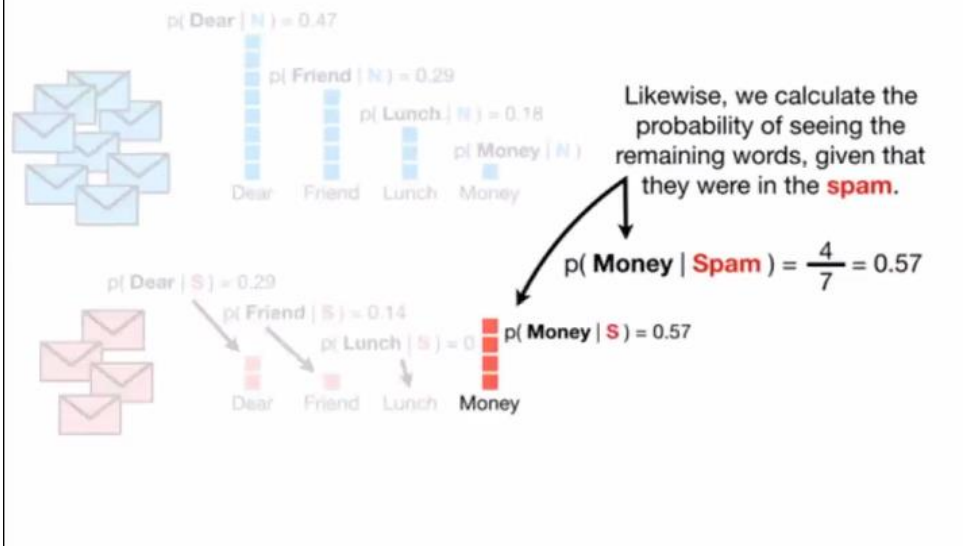

Dear
Friend
Lunch
Money
8, the total number of times Dear occurred in normal messages...
p(Dear | Normal) = 8/17
divided by 17, the total number of words in all of the normal messages.


p(Dear | N) = 0.47
Dear
Friend
Lunch
Money
p(Dear | Normal) = 8/17 = 0.47


p(Dear | N) = 0.47
p(Friend | N) = 0.29
Dear
Friend
Lunch
Money
p(Friend | Normal) = 5/17 = 0.29
p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
Dear
Friend
Lunch
Money
p(Lunch | Normal) = 3/17 = 0.18
p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06
Dear
Friend
Lunch
Money
p(Money | Normal) = 1/17 = 0.06

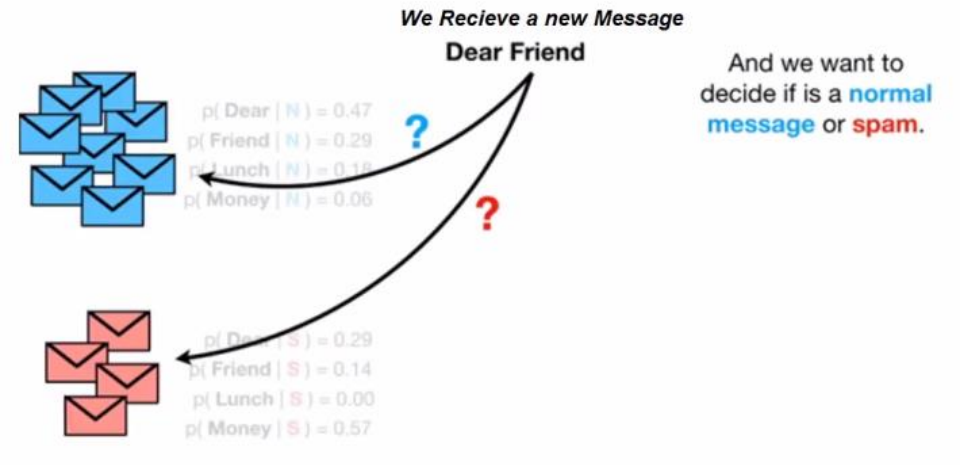

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N)
Dear
Friend
Lunch
Money
Now we make a histogram of all the words that occur in the spam...
Dear
Friend
Lunch
Money

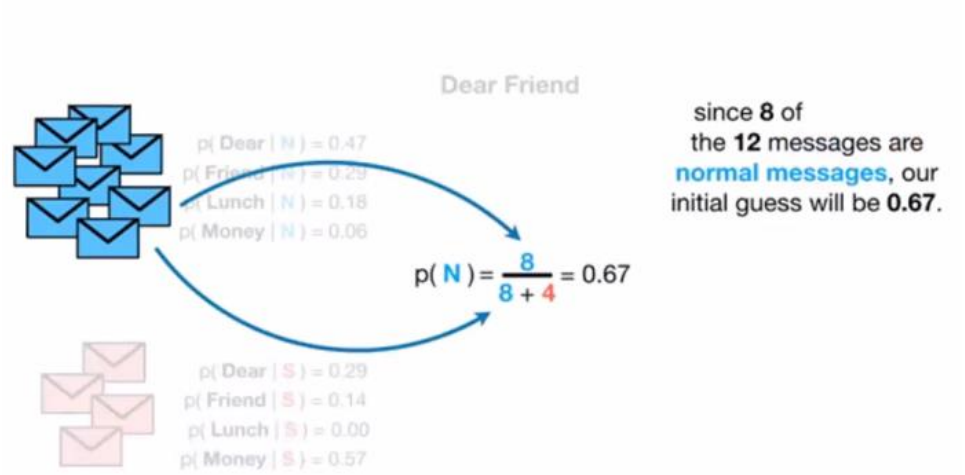

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0
Dear
Friend
Lunch
Money
0.29.
p(Dear | Spam) = 2/7 = 0.29
p(Dear | S) = 0.29
Dear
Friend
Lunch
Money


p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N)
Dear
Friend
Lunch
Money
Likewise, we calculate the probability of seeing the remaining words, given that they were in the spam.
p(Money | Spam) = 4/7 = 0.57
p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0
p(Money | S) = 0.57
Dear
Friend
Lunch
Money

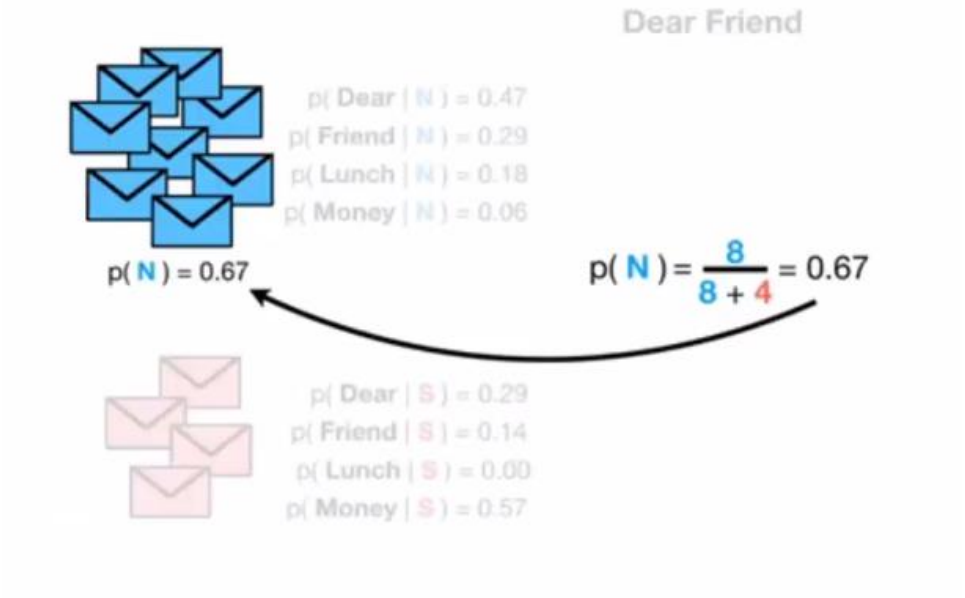
p(**Dear** | **N**) = 0.47
p(**Friend** | **N**) = 0.29
p(**Lunch** | **N**) = 0.18
p(**Money** | **N**) = 0.06

p(**Dear** | **S**) = 0.29
p(**Friend** | **S**) = 0.14
p(**Lunch** | **S**) = 0.00
p(**Money** | **S**) = 0.57





since **8** of the **12** messages are **normal messages**, our initial guess will be **0.67**.



p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

Now we multiply that initial guess by probability that the word **Dear** occurs in a **normal message**...

...and the probability that the word **Friend** occurs in a **normal message**.

p(N) × p(**Dear** | N) × p(**Friend** | N)

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

Now we multiply that initial guess by probability that the word **Dear** occurs in a **normal message**...

...and the probability that the word **Friend** occurs in a **normal message**.

p(N) × p(**Dear** | N) × p(**Friend** | N)

0.67 × 0.47 × 0.29 = 0.09

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

Now we multiply that initial guess by probability that the word **Dear** occurs in a **normal message**...

...and the probability that the word **Friend** occurs in a **normal message**.

p(N) × p(**Dear** | N) × p(**Friend** | N)

0.67 × 0.47 × 0.29 = 0.09

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

In a simple way, we can think of **0.09** as the score that **Dear Friend** gets if it is a **Normal Message**.

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

Now we multiply that initial guess by probability that the word **Dear** occurs in a **normal message**...

...and the probability that the word **Friend** occurs in a **normal message**.

p(N) x p(**Dear** | N) x p(**Friend** | N)

$0.67 \times 0.47 \times 0.29 = 0.09$ p(N | **Dear Friend**)

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

In a simple way, we can think of **0.09** as the score that **Dear Friend** gets if it is a **Normal Message**.

p(N | Dear Friend) ∝ 0.09 Dear Friend

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

$p(S) = \frac{4}{4+8} = 0.33$

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

p(N | Dear Friend) ∝ 0.09

Dear Friend

p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

p(S) × p(**Dear** | S) × p(**Friend** | S)

p(Dear | S) = 0.29
p(Friend | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

p(S) = 0.33

p(N | Dear Friend) ∝ 0.09

Dear Friend

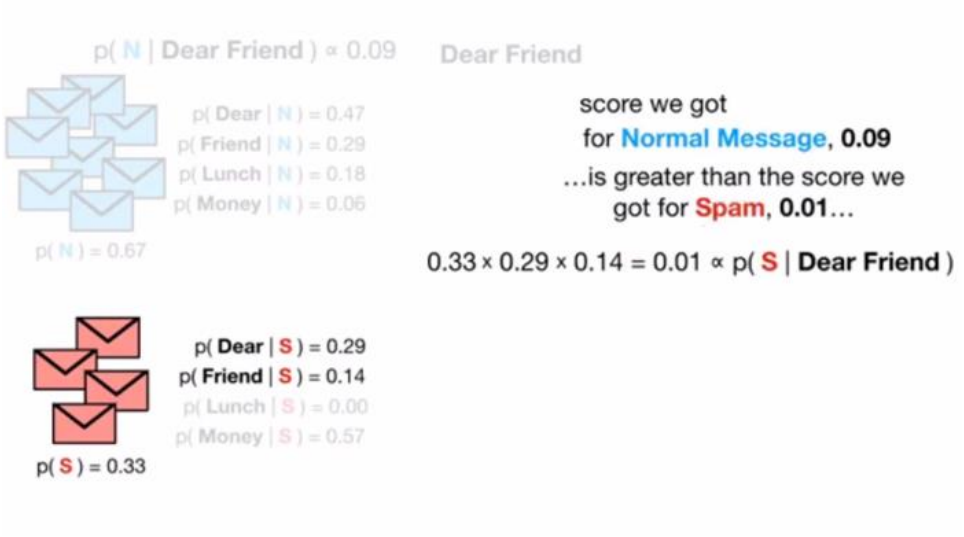
p(Dear | N) = 0.47
p(Friend | N) = 0.29
p(Lunch | N) = 0.18
p(Money | N) = 0.06

p(N) = 0.67

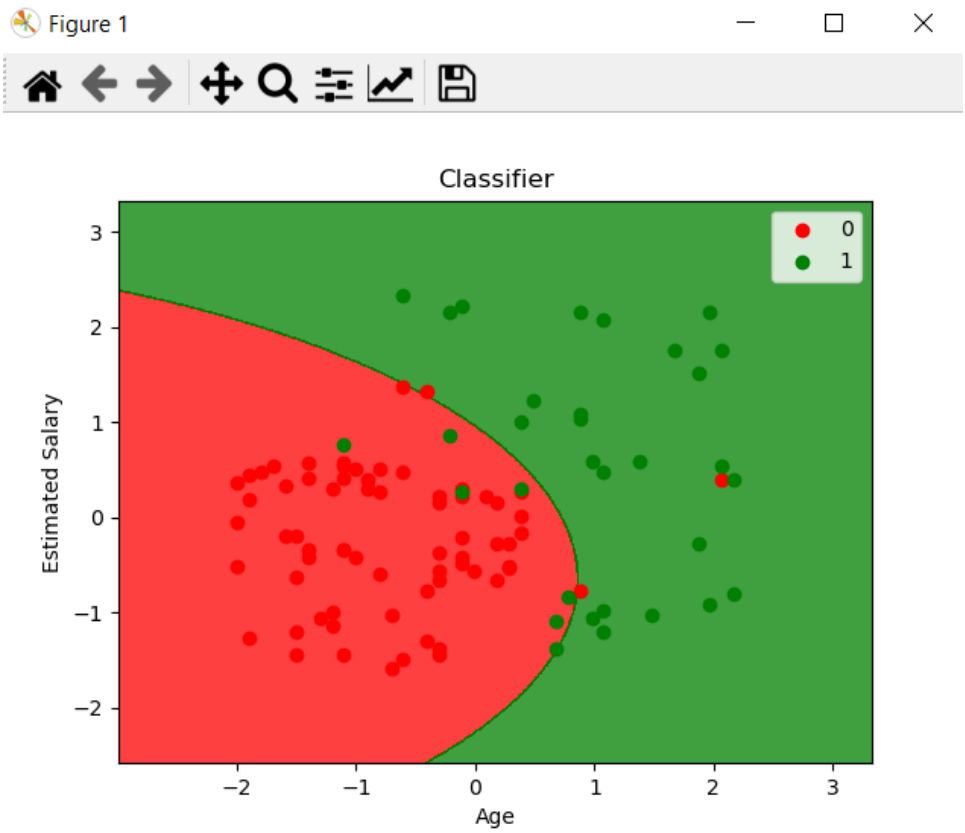
0.33 × 0.29 × 0.14 = 0.01 ∝ p(S | **Dear Friend**)

p(**Dear** | S) = 0.29
p(**Friend** | S) = 0.14
p(Lunch | S) = 0.00
p(Money | S) = 0.57

p(S) = 0.33



Visualization:



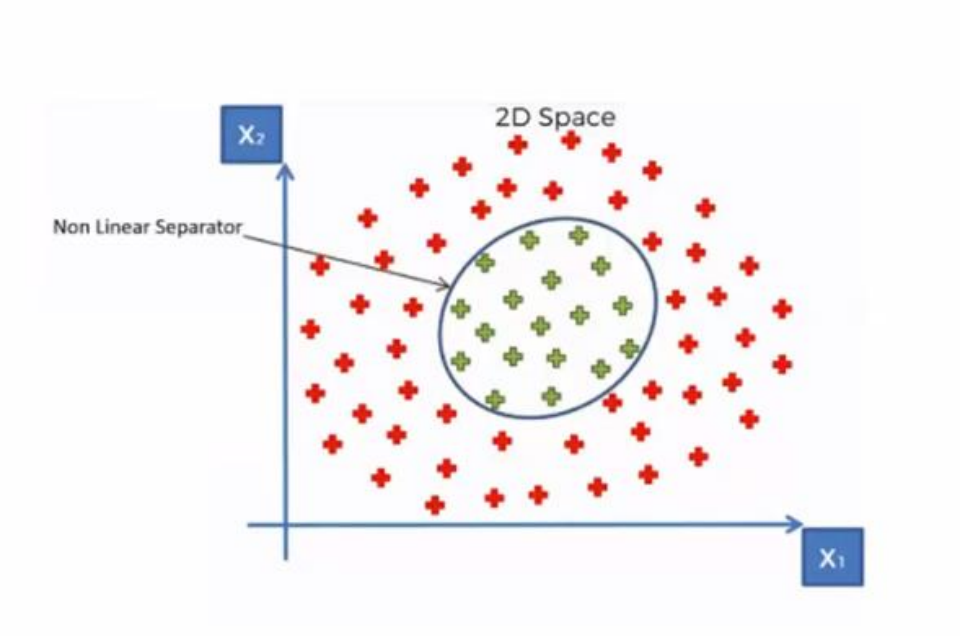
Accuracy:

Name	Type	Size	Value
acc	float64	1	0.9
classifier	naive_bayes.GaussianNB	1	GaussianNB object c

Classification	Value to Predict is Discrete		
	Logistic Regression		
	Support Vector Machine	93%	
	DecisionTree Classifier	94%	max_depth=2
	RandomForest Classifier	94%	max_depth=2 ,Trees=10
	Naive Bayes	90%	

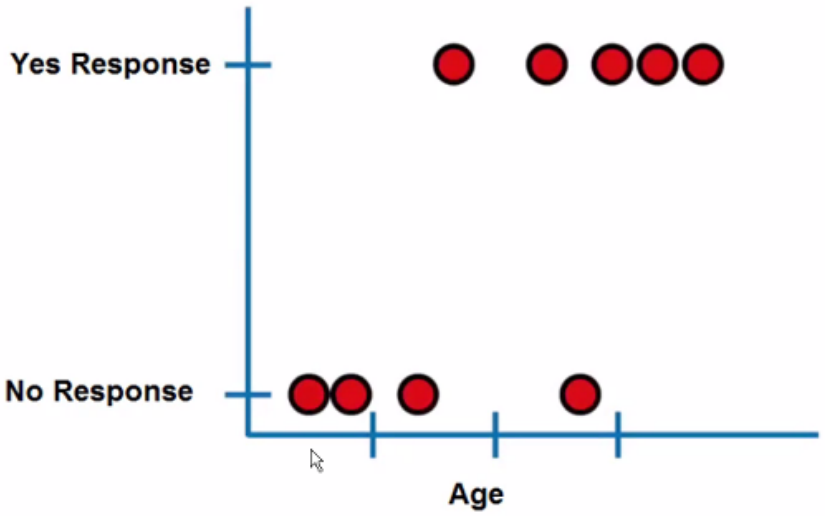
Logistic Regression:

How the formula is going to give to results to classify the problem

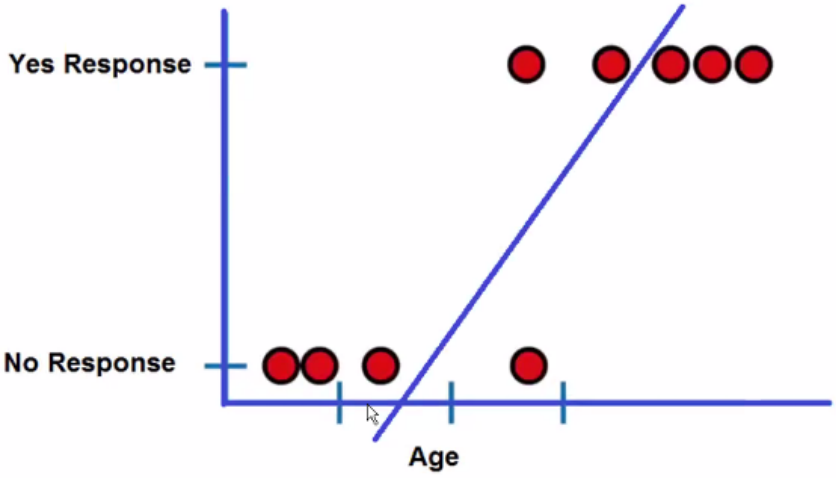


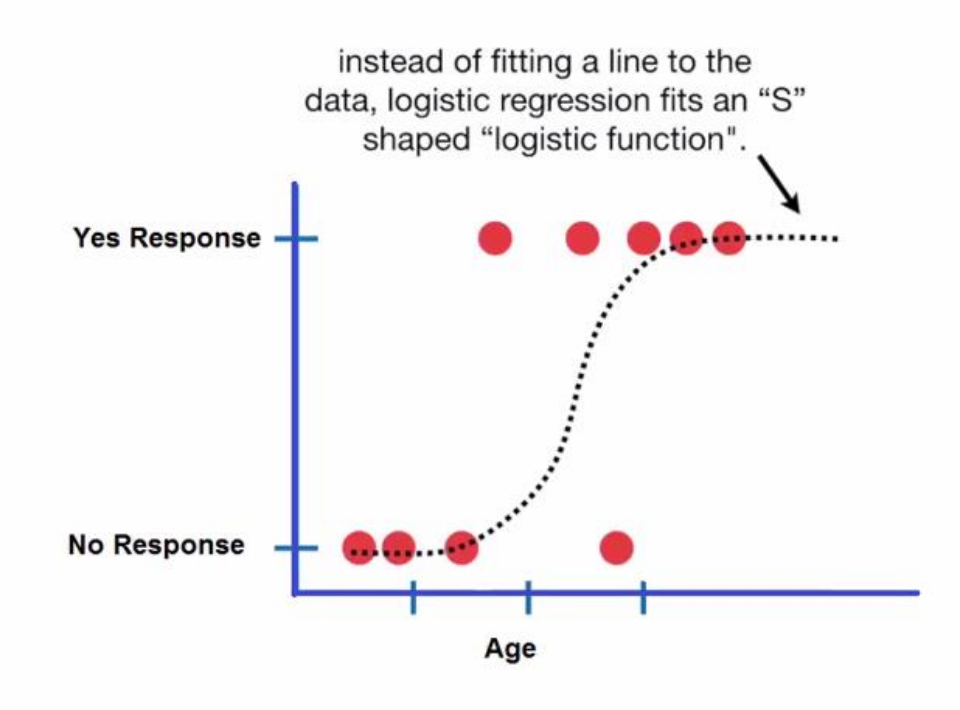
How the straight line fit => Regression

Logistic Regression

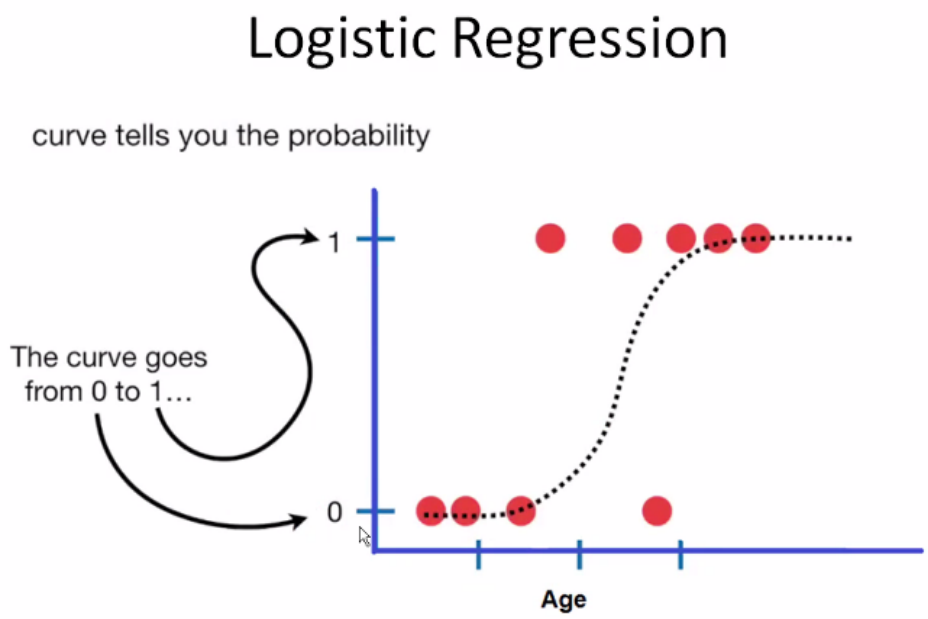


Logistic Regression

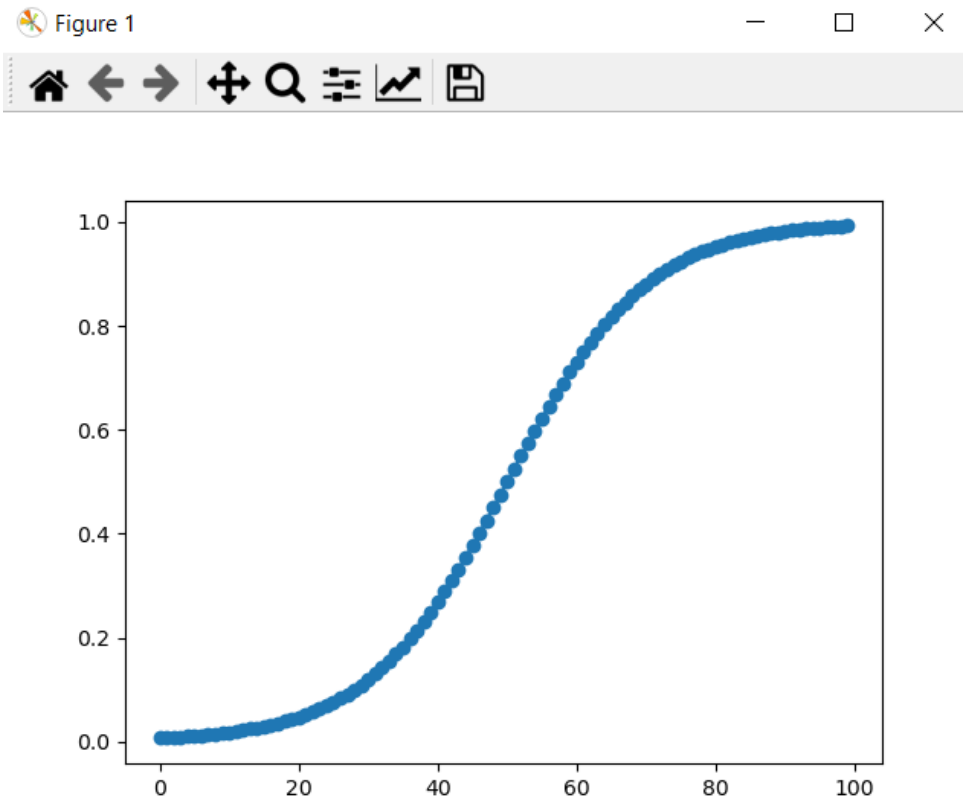




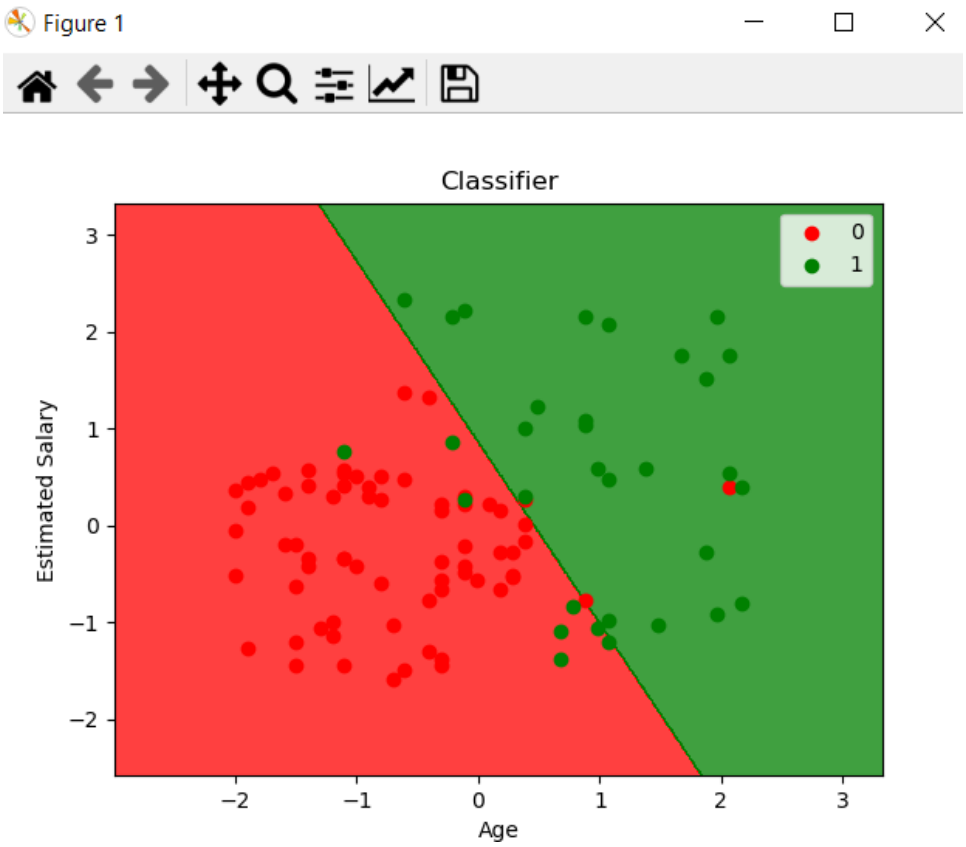
Bring Logistic function



Logistic function is also called as sigmoid functions



Visualization:

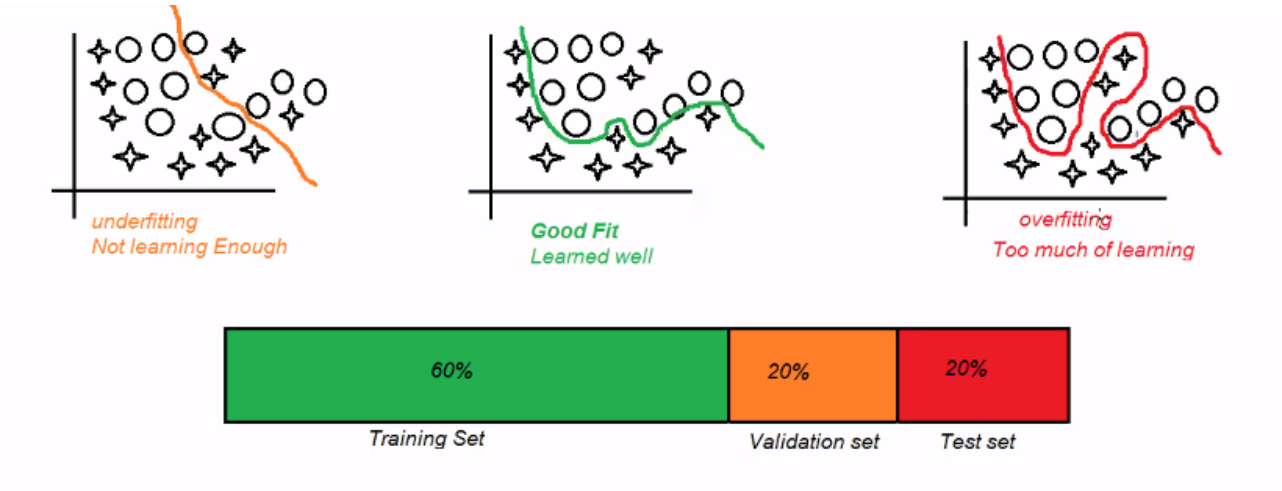


Accuracy:

Name	Type	Size	Value
acc	float64	1	0.89
classifier	linear_model._logistic.LogisticRegression	1	LogisticRegress…

Classification Value to Predict is Discrete

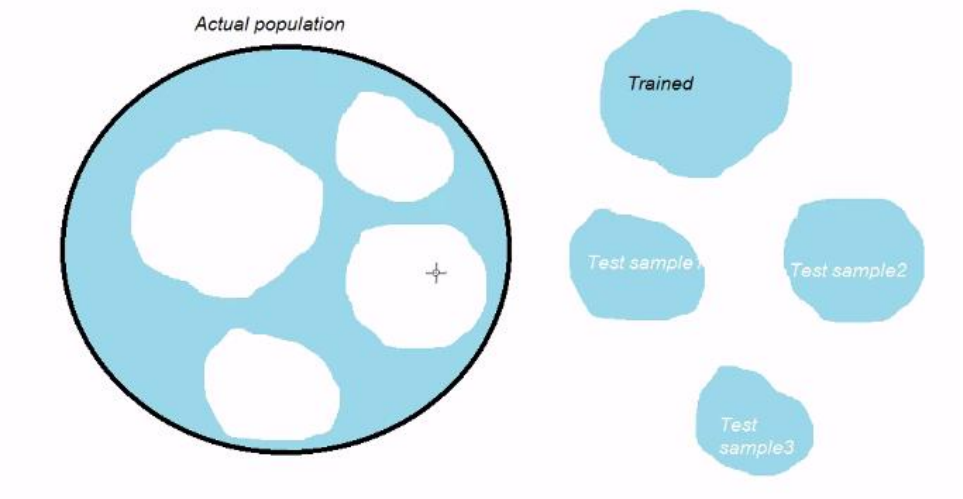
Logistic Regression	89%	
Support Vector Machine	93%	
DecisionTree Classifier	94%	max_depth=2
RandomForest Classifier	94%	max_depth=2 ,Trees=10
Naive Bayes	90%	



Training Set	*we train the model on training data*
Validation Set	*After training the model, we check how well it performs on the validation set*
Test Set	*When we have final Model (model that has performed well on both training and validation). We evaluate it on test set to go and get unbiaded estimation of its performance*

Stability check:

Test the model with various test dataset



K-Fold Cross Validation:

X_train, y_train – divide it into K-Folds

K = 5 to 10 folds

100 rows – dataset

Fold1	20 rows
Fold2	20 rows
Fold3	20 rows
Fold4	20 rows
Fold5	20 rows

Same as bootstrapping – to have equal folds.

Cross validation – 90%-10% Train – Test

K-Fold Cross Validation
X_train,y_train divide it into K-Folds k=5
100 rows

Fold1	20 rows
Fold2	20 rows
Fold3	20 rows
Fold4	20 rows
Fold5	20 rows

Cross Validation - 90%-10% Train-Test

Training	Testing	Accuracy
Fold1-2-3-4	Fold-5	93%
Fold-1-2-3-5	Fold-4	80%
Fold-1-2-4-5	Fold-3	75%
Fold-1-3-4-5	Fold-2	98%
Fold-2-3-4-5	Fold-1	90%

High variance
Accuracy variation > 1% Overfiiting issues
Accuracy variation == 1% Good Fit