



ML Training and Algorithms

by IIT Roorkee





- How algorithms work



- How algorithms work
- Explore



- How algorithms work
- Explore
 - Internal working



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples
- Concepts



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples
- Concepts
 - Linear Regression



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples
- Concepts
 - Linear Regression
 - Decision Trees



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples
- Concepts
 - Linear Regression
 - Decision Trees
 - Ensemble



- How algorithms work
- Explore
 - Internal working
 - Using real-life examples
- A few key Algorithms
 - Linear Regression
 - Decision Trees
 - SVM
 - Ensemble
 - Unsupervised



- Concepts Associated to Classification Algorithm Implementation
 - Gradient Descent
 - Batch Gradient Descent
 - Stochastic Gradient Descent
 - Mini Batch
 - Regularization



Regression Vs Classification

(a) Regression – We try to predict value of a continuous output variable or we try to obtain a function that maps input variables to some continuous valued output.

$$f(x) = 0.8 \text{ //continuous value}$$

Given experience of an employee in years, predict salary

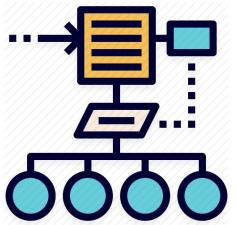
(b) Classification – We try to predict value of a discrete output variable or we try to obtain a function that maps input variables to some discrete categories.

$$f(x) = 0.8, 0.8 > \text{threshold} \Rightarrow \text{discrete class} \text{ //also read as probability of the class } y \text{ given data } x$$

Given a patient with a tumor, we have to predict whether the tumor is malignant or benign.



Data



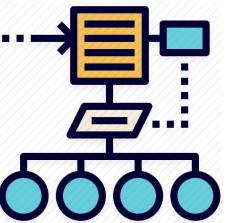
Algorithm



Model



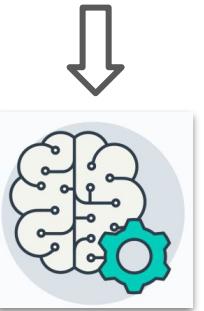
Data



Algorithm



Live Data



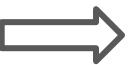
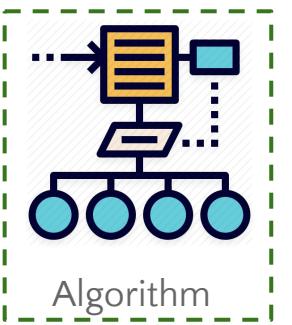
Model



Prediction



Data



Model



Prediction



Predicting salary based on the number of years of experience



2000



2000, 4000



2000, 4000, 6000



2000, 4000, 6000, _____



2000, 4000, 6000, 8000

Other scenario: 3000, 5000, 7000,.....



Let's formulate the machine learning problem..

Historical Data

Years of Experience	Salary
1	2000
2	4000
3	6000



Let's formulate the machine learning problem..





Let's formulate the machine learning problem..

Historical Data

Years of Experience	Salary
1	2000
2	4000
3	6000

Unknown Salary

Years of Experience	Salary
4	???

Training

Model



Let's formulate the machine learning problem..

Historical Data

Years of Experience	Salary
1	2000
2	4000
3	6000

Unknown Salary

Years of Experience	Salary
4	???

Training

Model

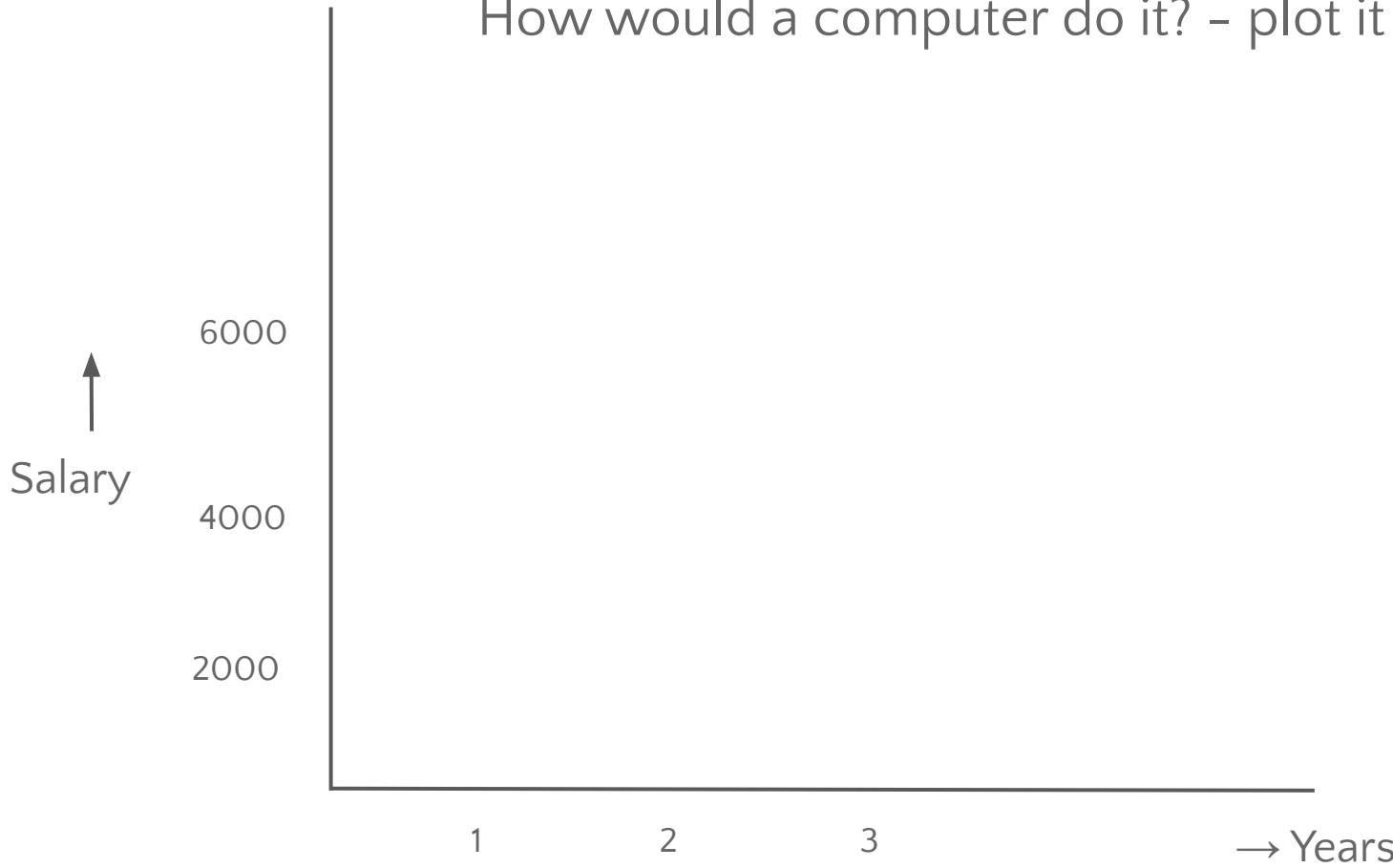
Salary

Years of Experience	Salary
4	8000

Prediction

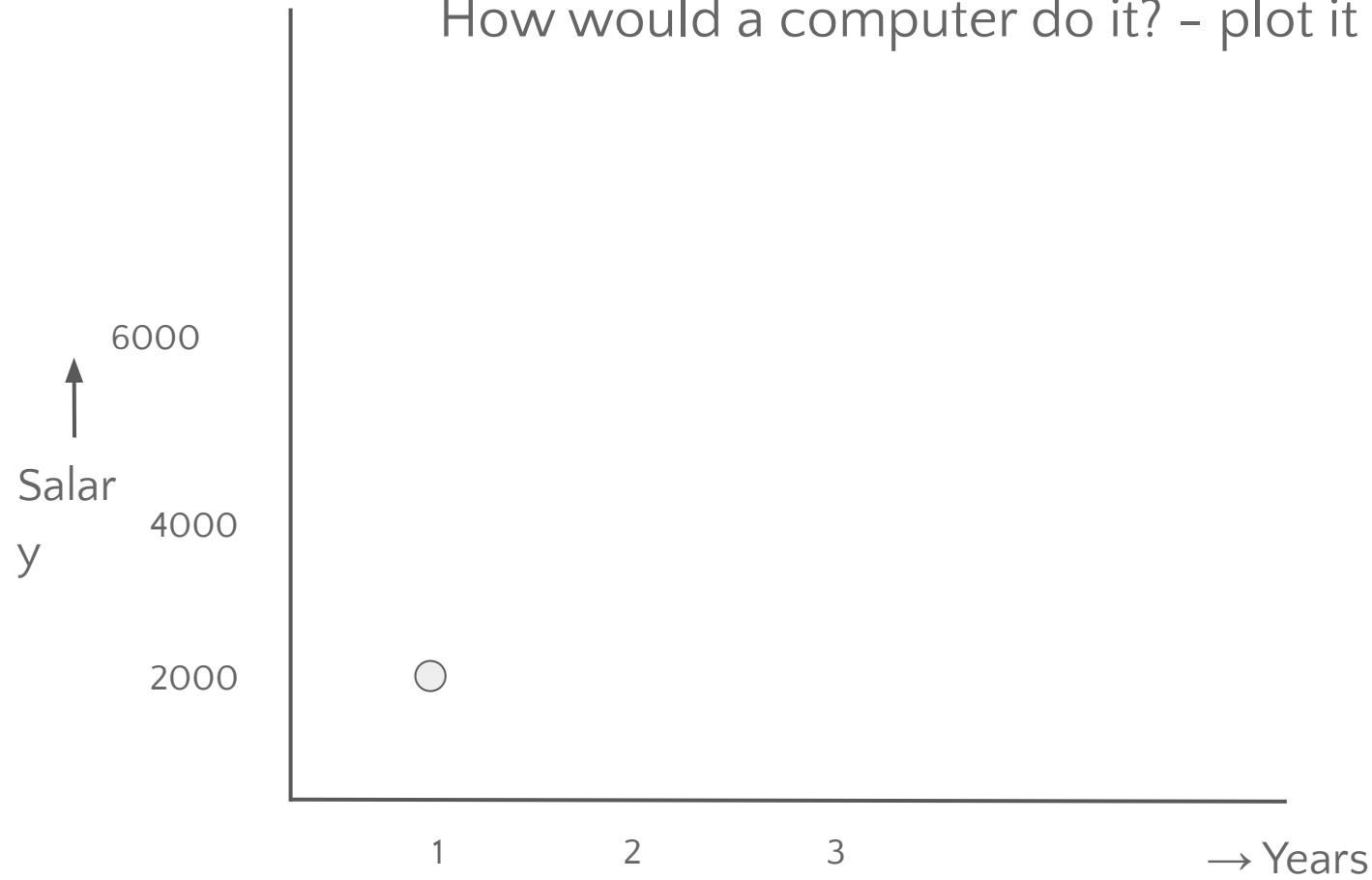


How would a computer do it? - plot it



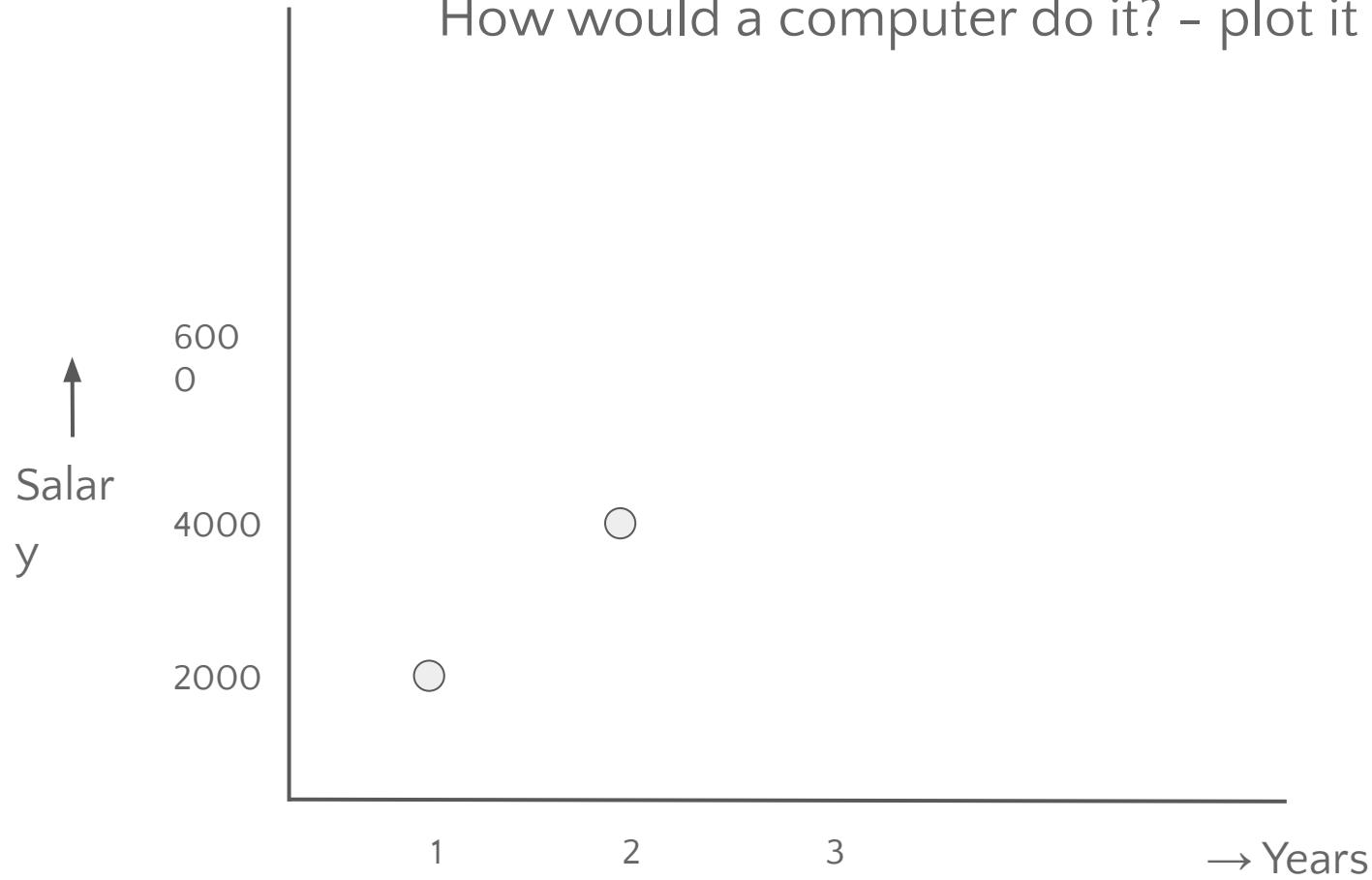


How would a computer do it? - plot it



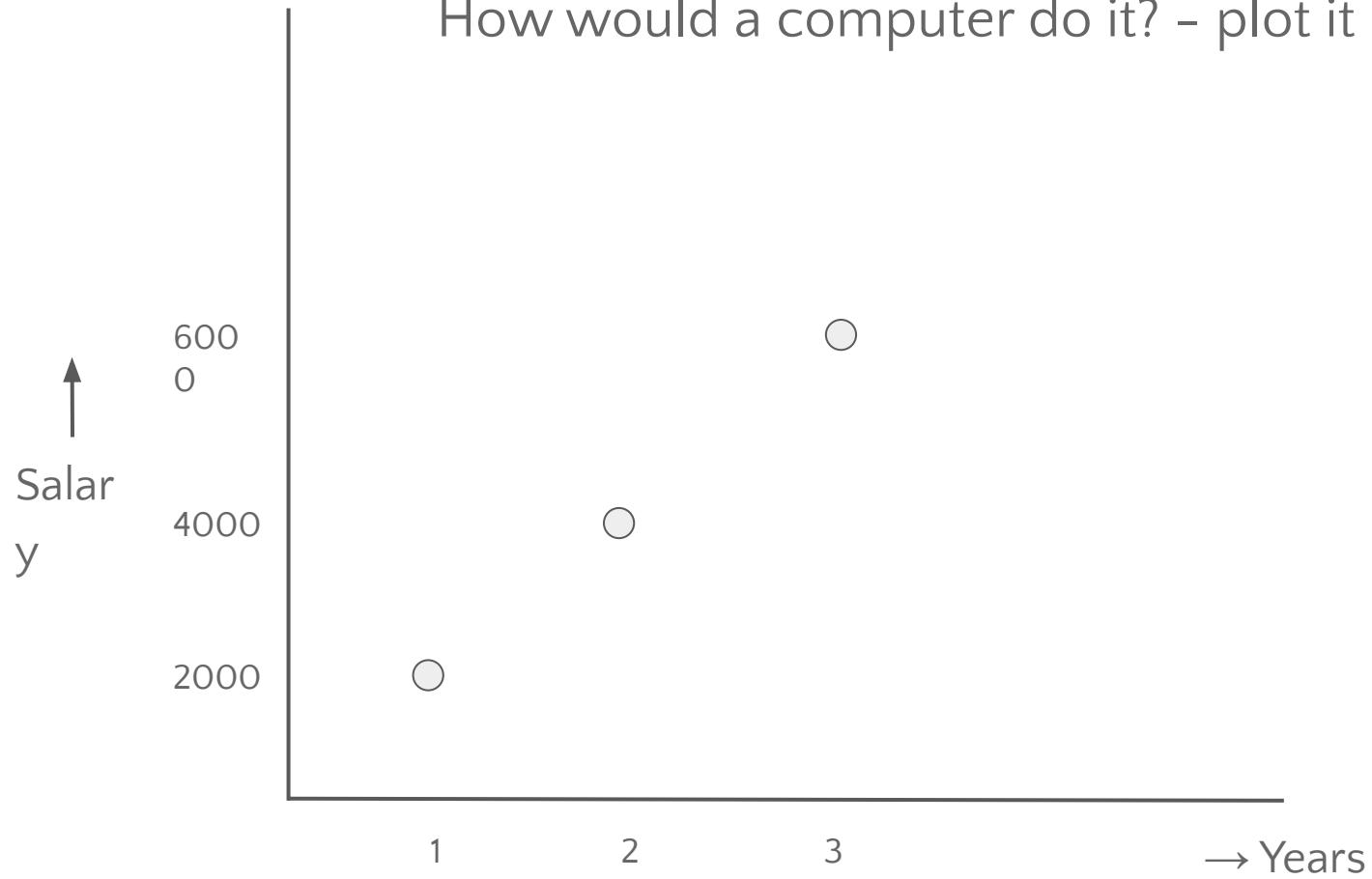


How would a computer do it? - plot it

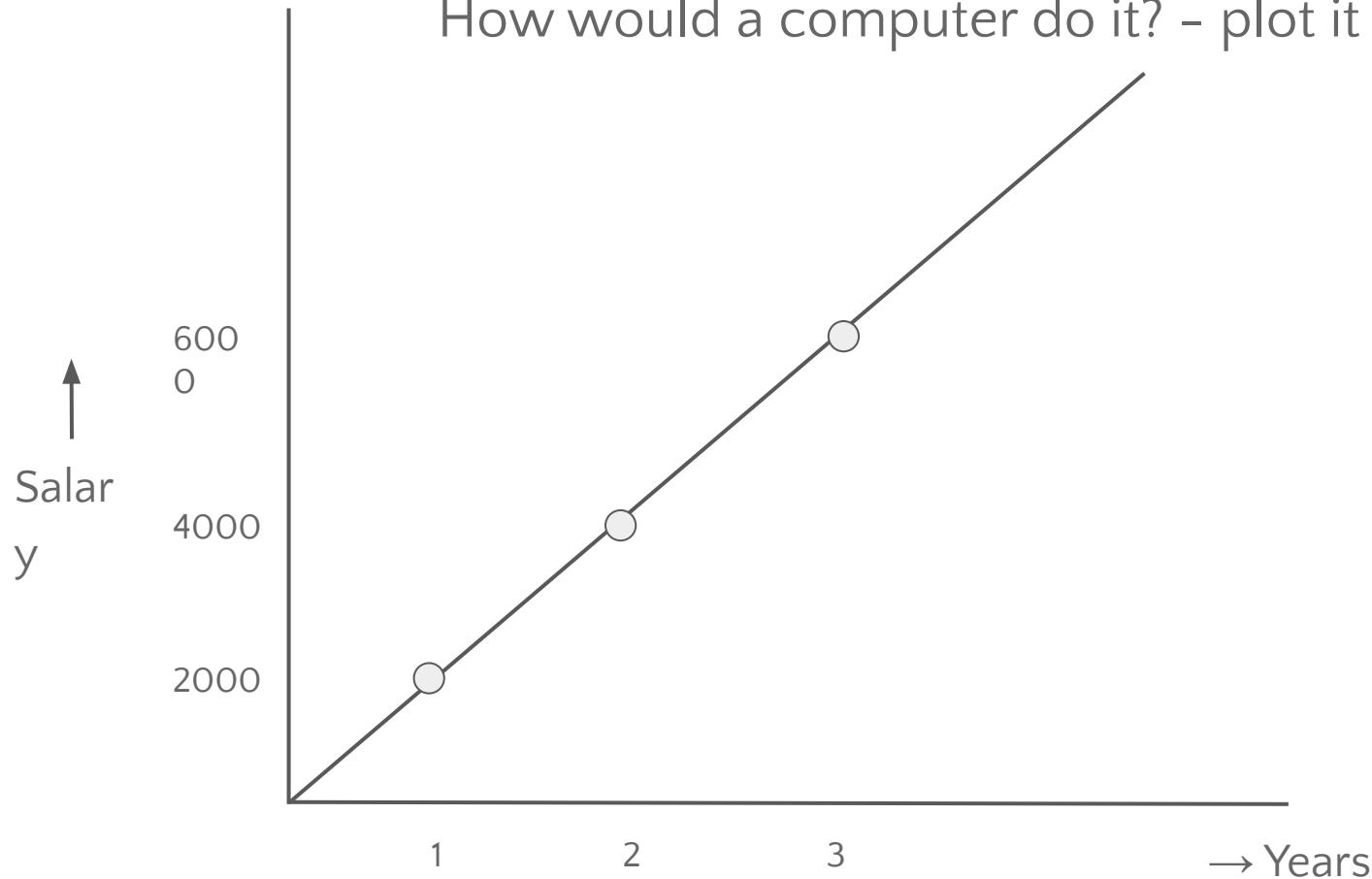




How would a computer do it? - plot it

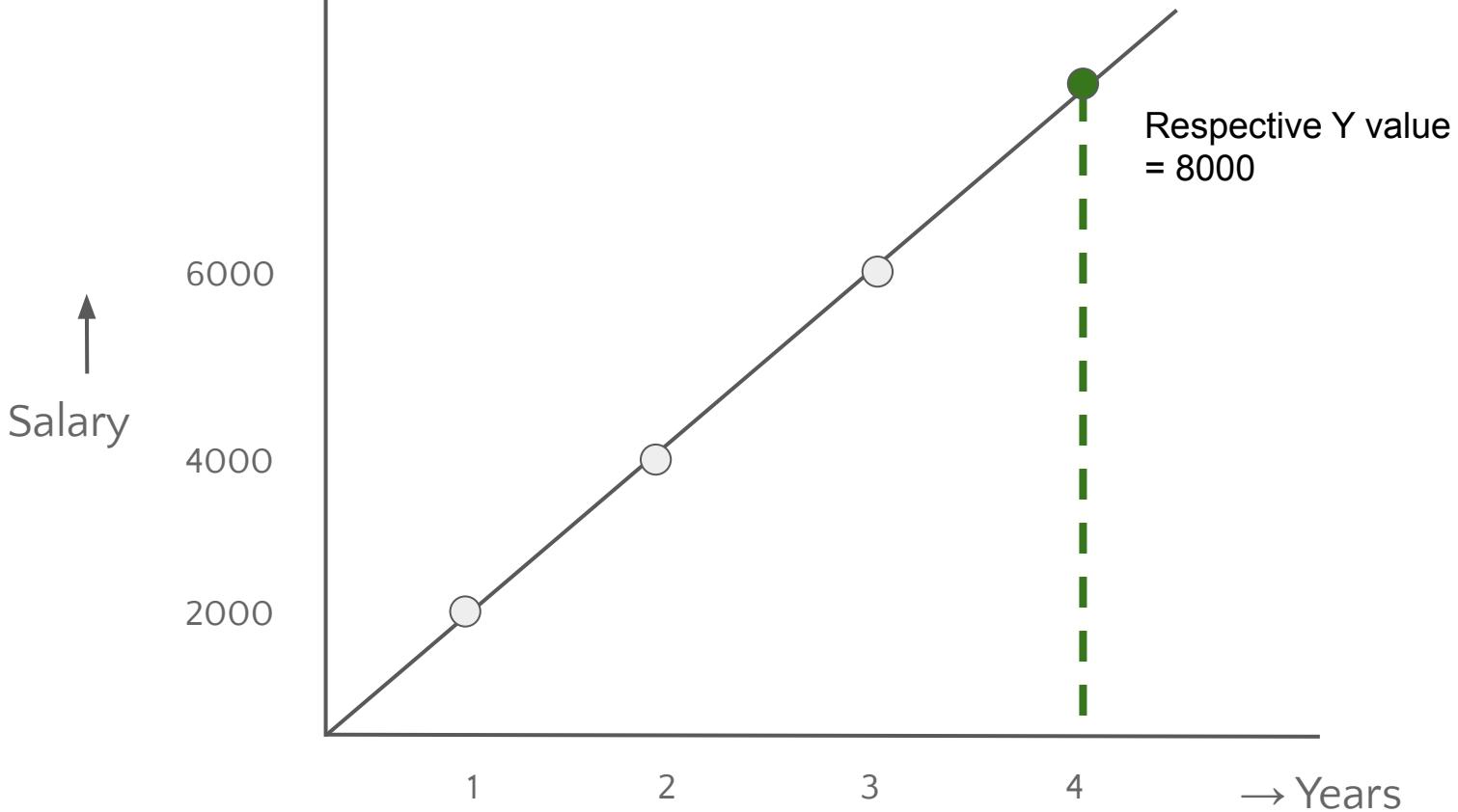


How would a computer do it? - plot it

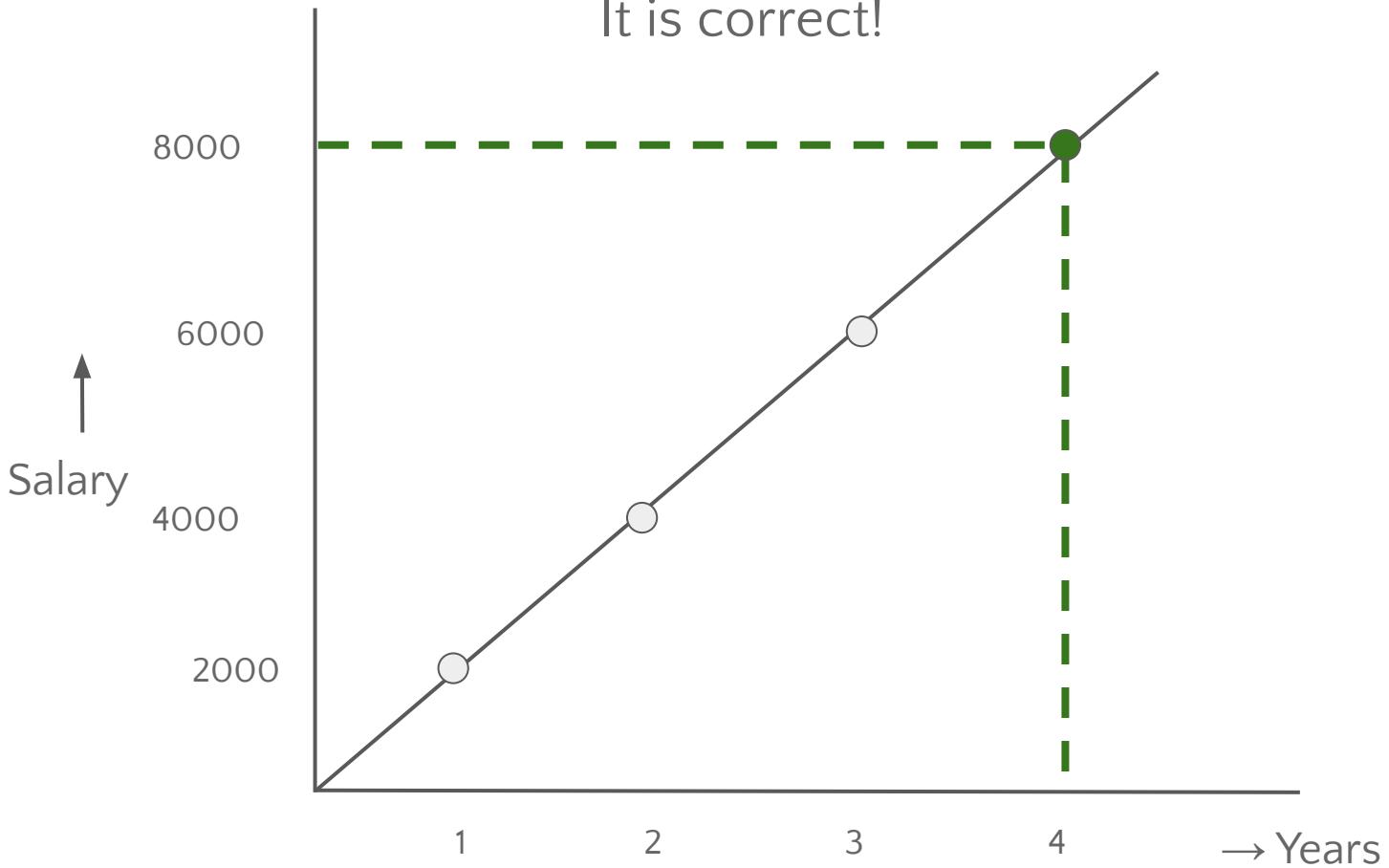




Ask the plot for the salary in 4th yr!

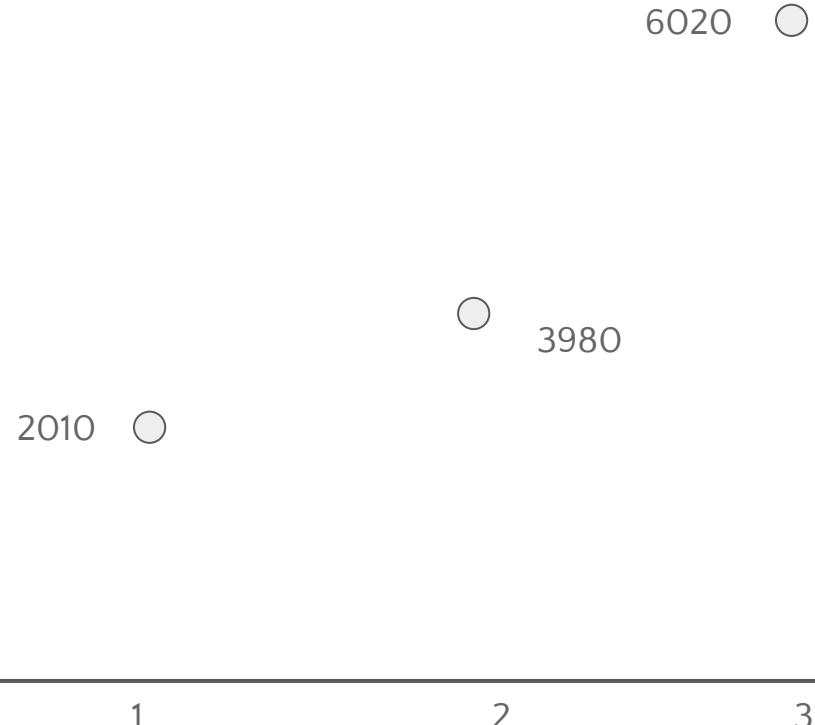


It is correct!



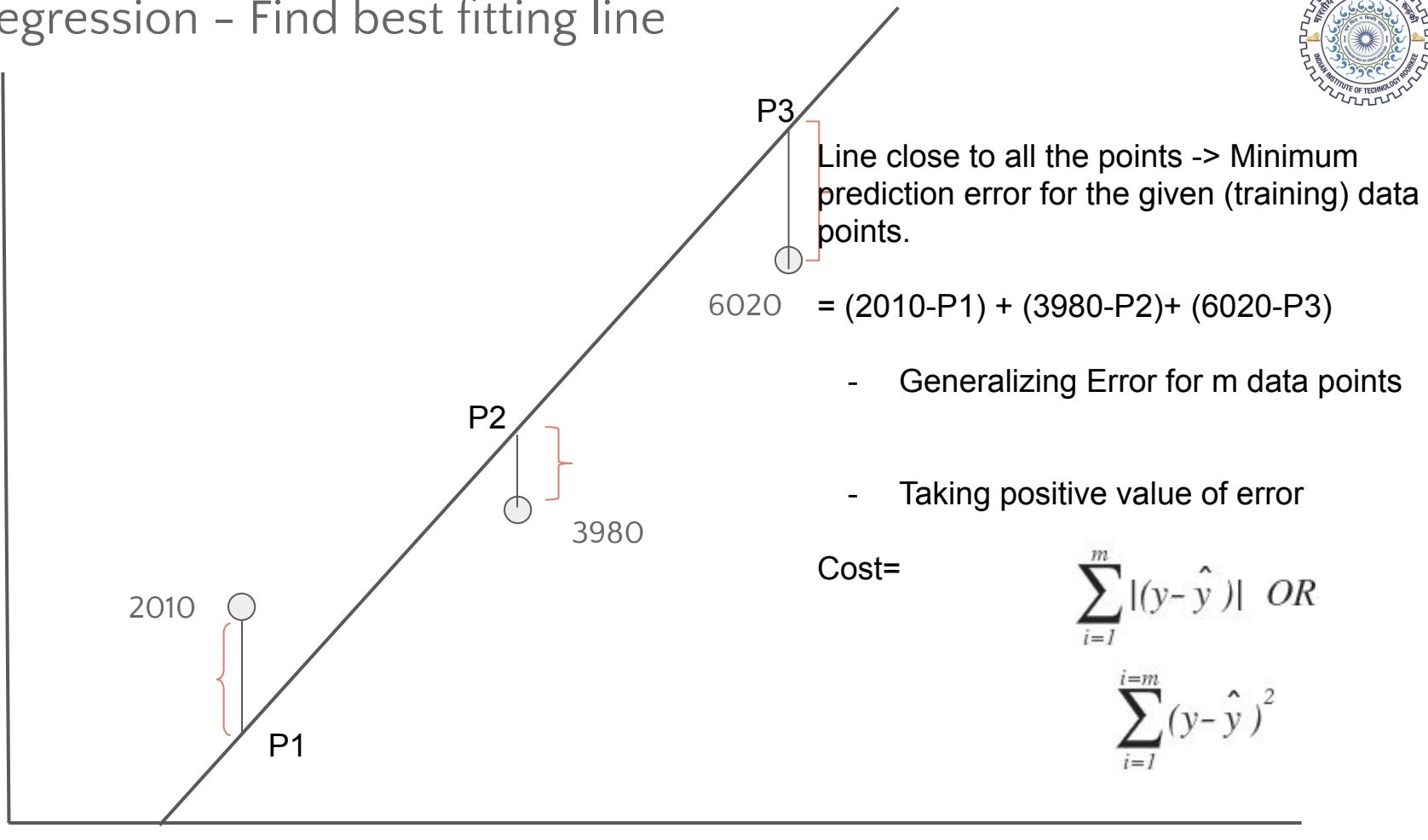


Life isn't perfect! You never get straight line!

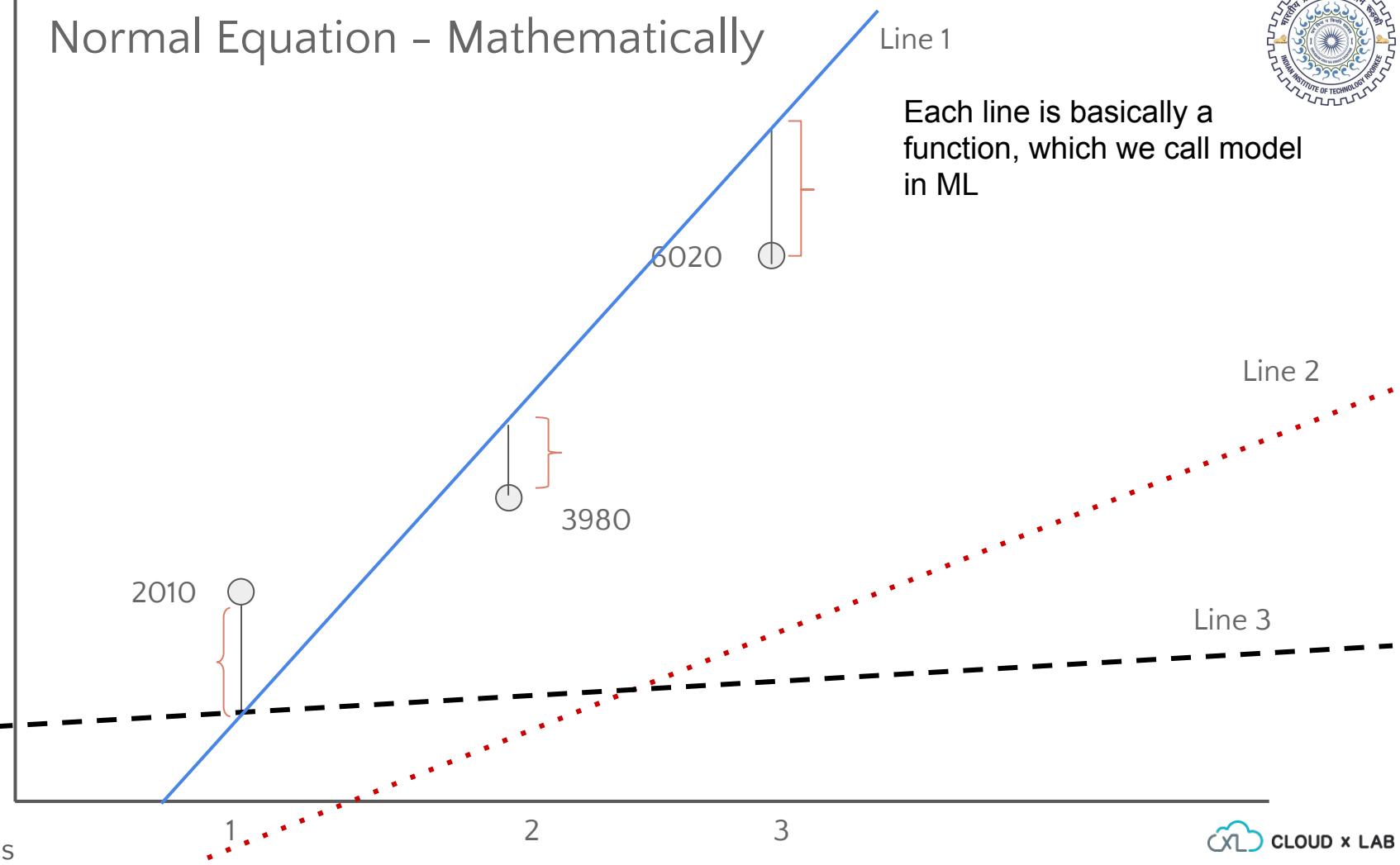




Linear Regression – Find best fitting line



Normal Equation - Mathematically





Try multiple lines! Whichever is closer.

Line 1

6020



Line 2

3980



Line 3

2010



1

2

3

Linear Regression Model Representation

- We need a model to fit the data.
- It is a straight line that best fit the data
-

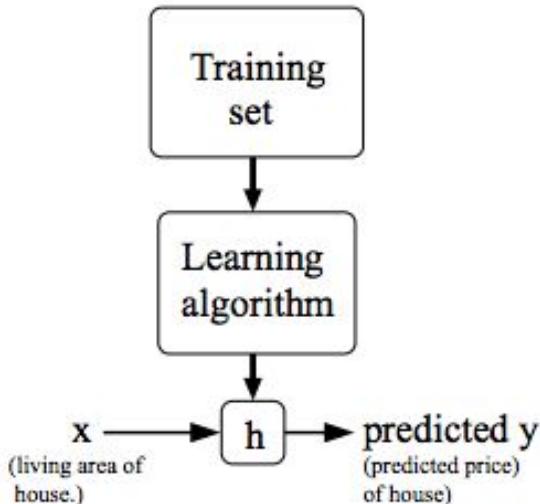
Notations:

m = Number of training examples,

X = input variable/ features,

y = output variable

(x^i, y^i) = i^{th} training example



$h : X \rightarrow y$ so that $h(x)$ is a good predictor for the corresponding value of y

// h is the decision function, the value given by h is decision score.

$$h_{\theta}(x) = \theta_0 + \theta_1 x, // w_0 \text{ and } w_1 \text{ decides the position of the line}$$

(seperating line between two classes in case of classification) and (predictive line crossing the data points in case of regression)

Here θ_0 is intercept, which is value of y at $x=0$ and θ_1 is slope of the line.



More Generic: Linear Regression Model

Representation

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \quad // \text{Two features } x_1 \text{ and } x_2$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \dots \dots \dots \quad // n \text{ number of features } x_1 \text{ and } x_2 \dots \dots \dots x_n$$

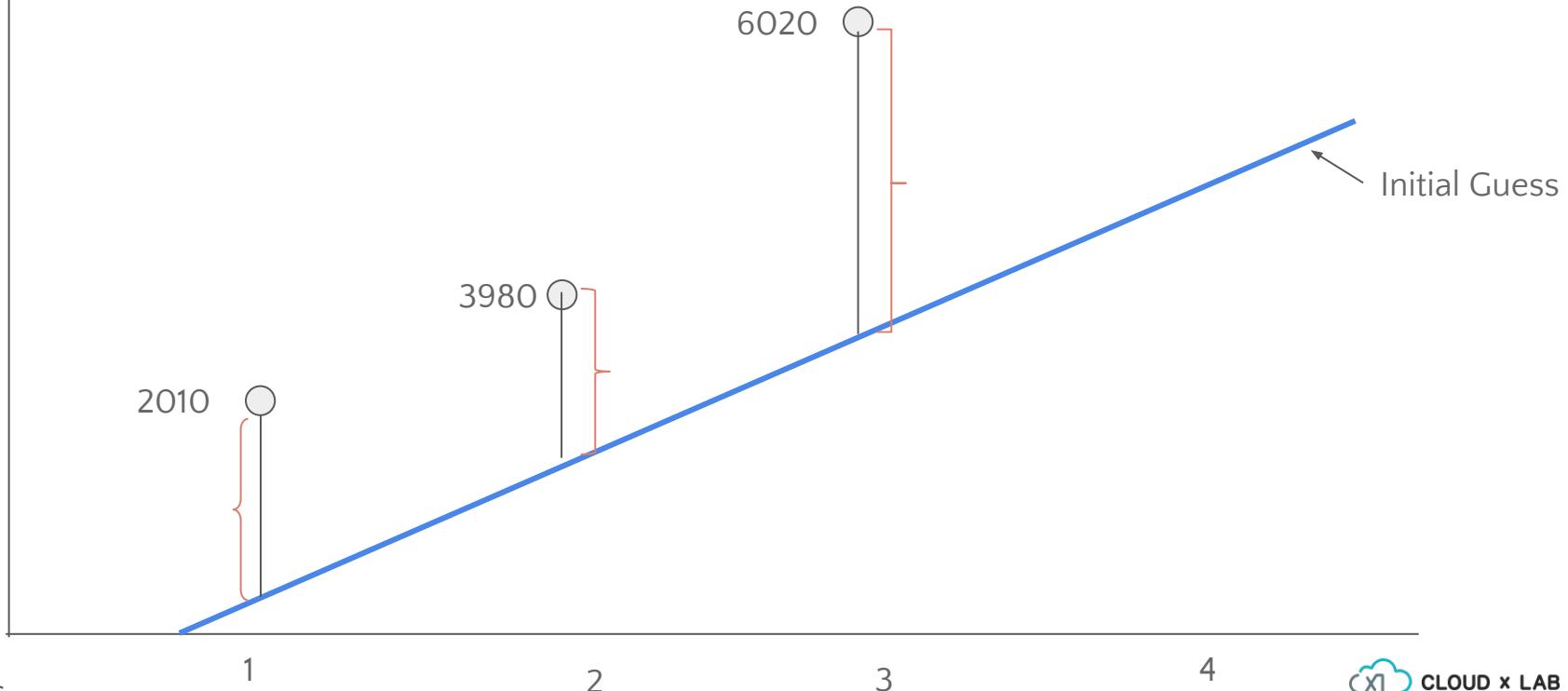
$$h(x) \equiv \sum_{i=0}^n \theta_i x_i$$

In vector form:

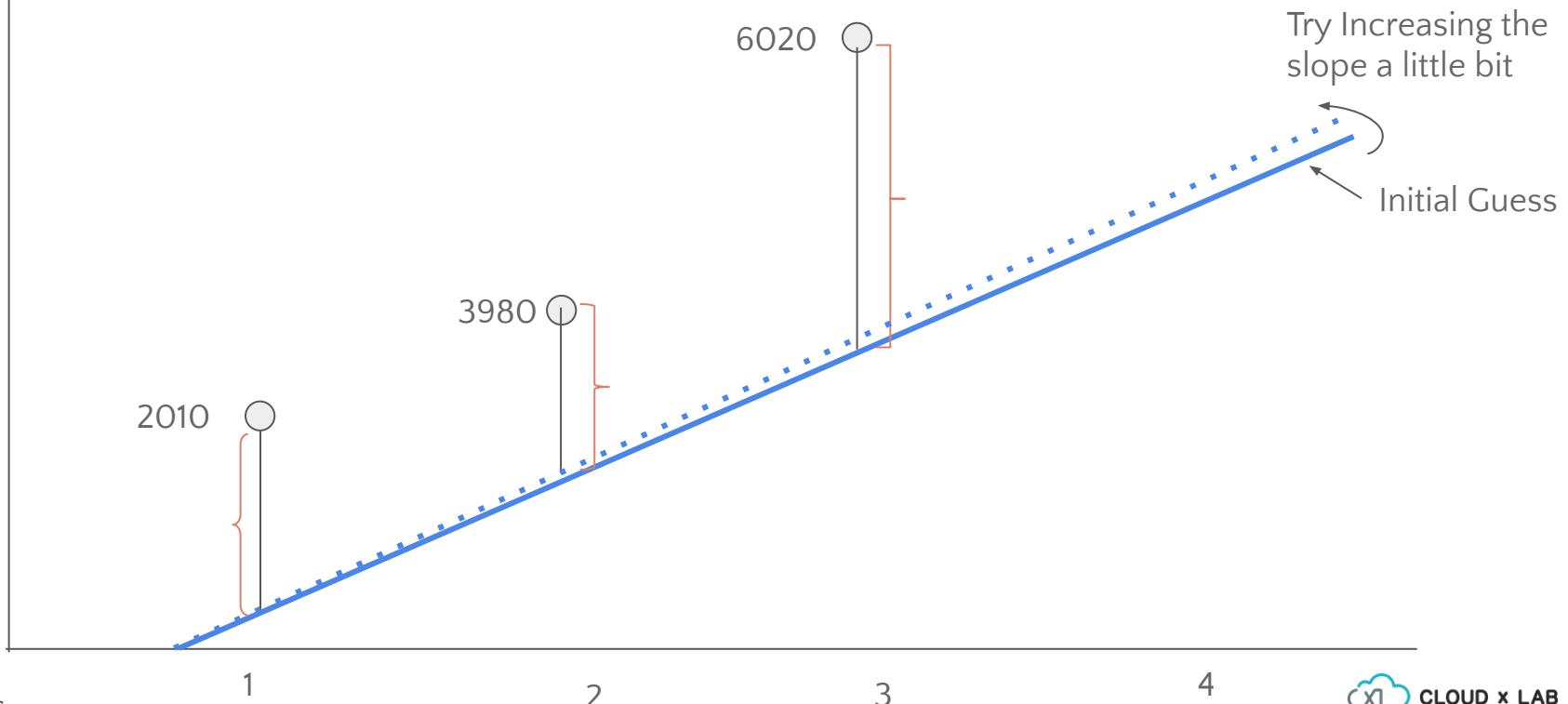
$$h(x) = \theta^T x \quad // \theta \text{ is the weight vector to be learned, and } x \text{ is the vector of one training instance}$$

$h(x) = \theta^T X \quad // X \text{ is the feature matrix for the complete dataset (computing function for all instances), in the classification.ipynb file we are dealing with matrix/vector operations.}$

Gradient Descent - Get the optimal slope of the line

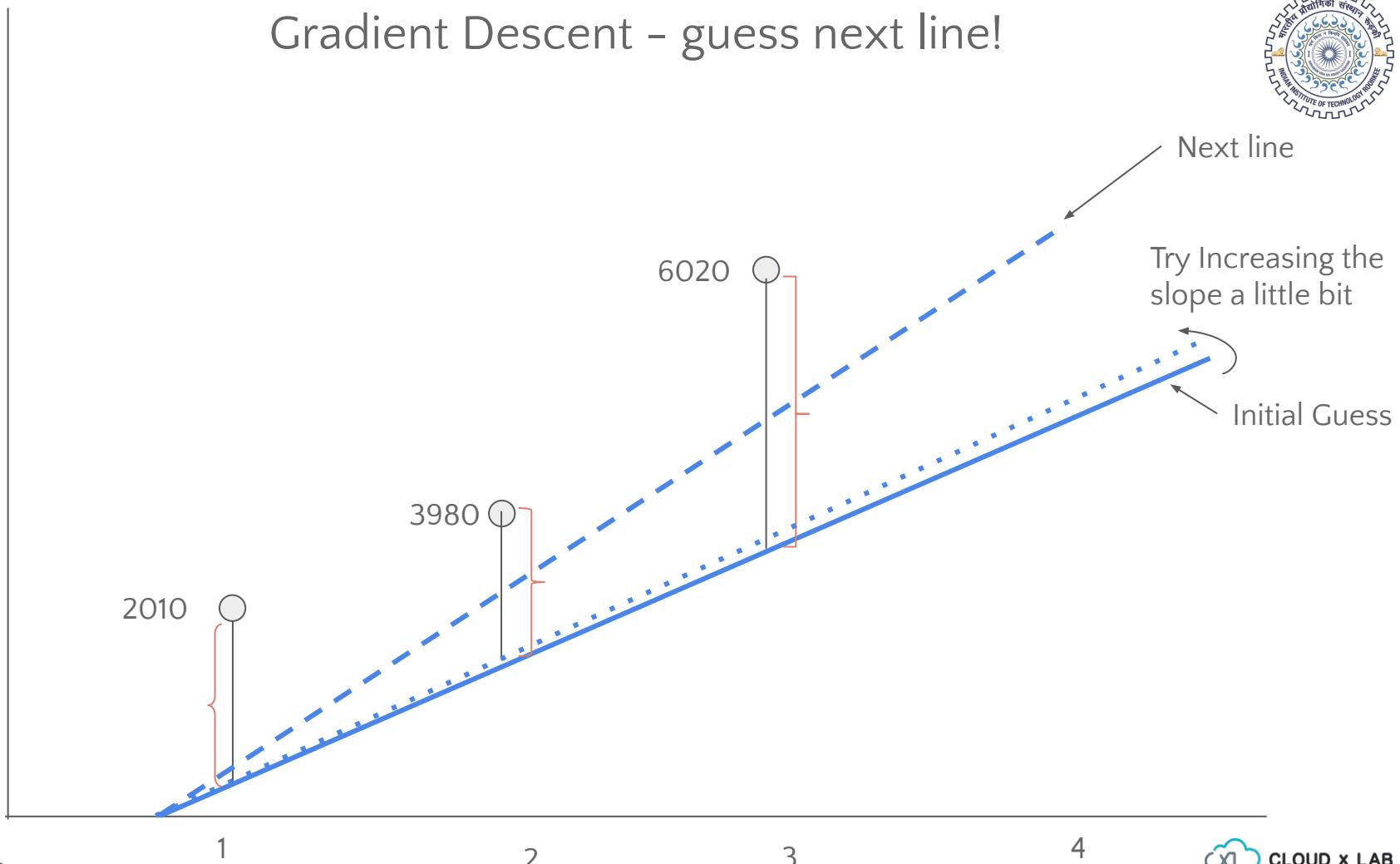


Gradient Descent - find impact of slight change





Gradient Descent - guess next line!





How to get the best fit line? Manual Attempt

Our objective is to reduce the difference between actual output value and predicted value by $h(x)$.

$$\text{Error} = h(x_i) - y_i = 0 \text{ //or close to 0 for } i^{\text{th}} \text{ training example}$$

$(x, y) = [(0,0), (1,1), (2,2), (3,3), (4,4)]$ // $y=0$ (bias term 'b' or ' θ_0 ') at $x=0$, line will start from origin,

$$h(x) = \theta_0 + \theta_1 * x \text{ //one feature only in the data}$$

- Find out the linear equation for the given data points.

$$\theta_0 = 0, \theta_1 = 0.5 \text{ //Initial guess}$$

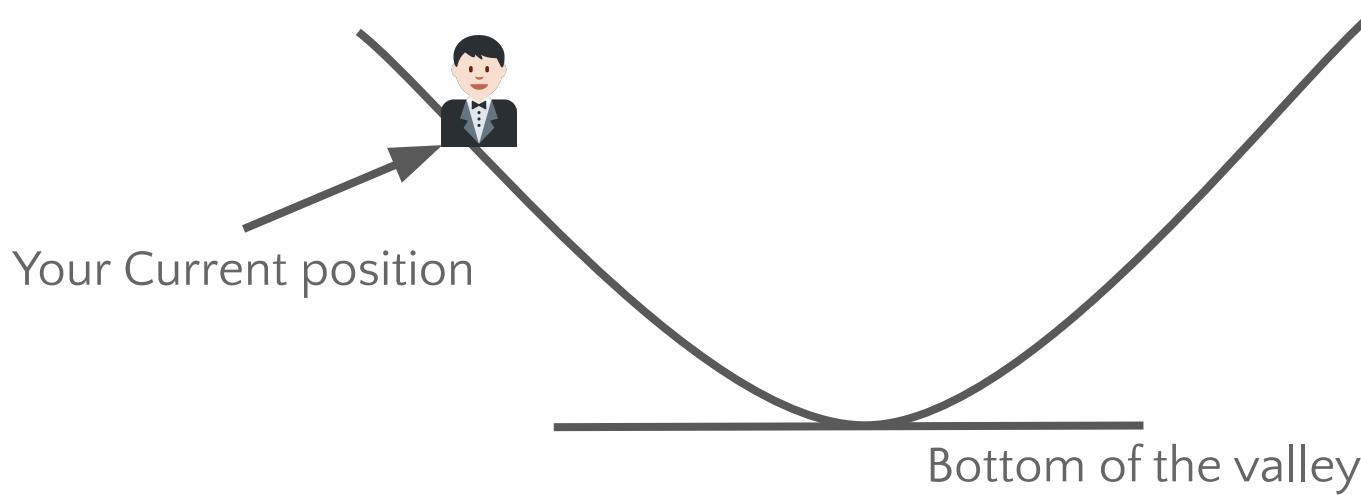
$$\theta_0 = 0, \theta_1 = 1 \text{ // Best fit}$$

$$\Rightarrow h(x) = 0 + 1 \cdot x = x \text{ (zero error)}$$

$$\tan\theta = 1 \Rightarrow \theta = 45 \text{ degrees}$$

Intuition of Gradient Descent

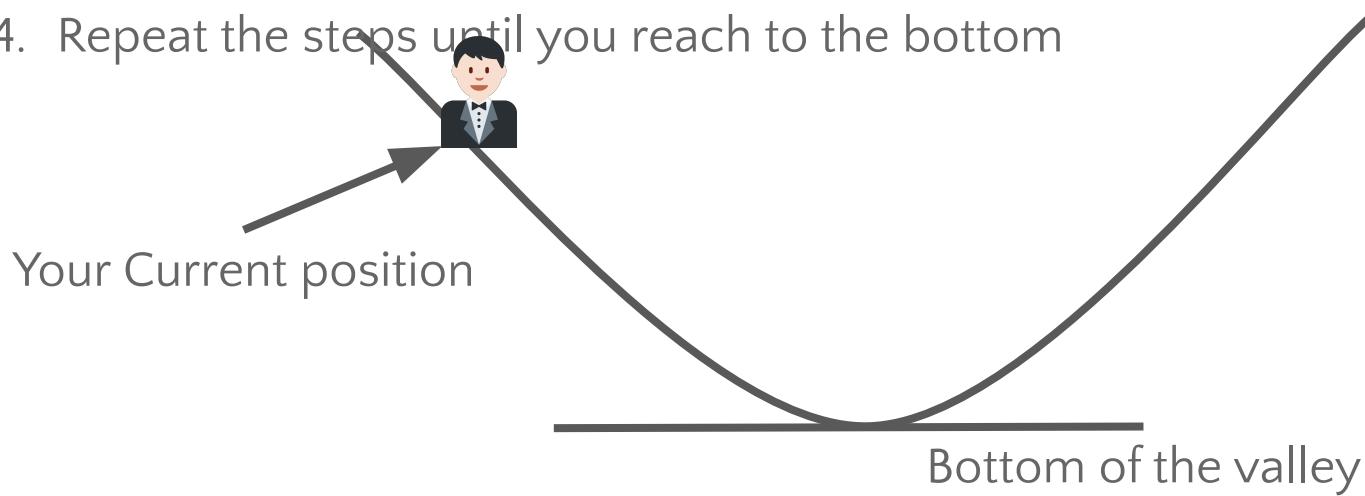
Question – If you are lost in the mountains in a dense fog and you can only feel the slope of the ground below your feet. Which strategy will you take to reach to the bottom of the valley quickly?



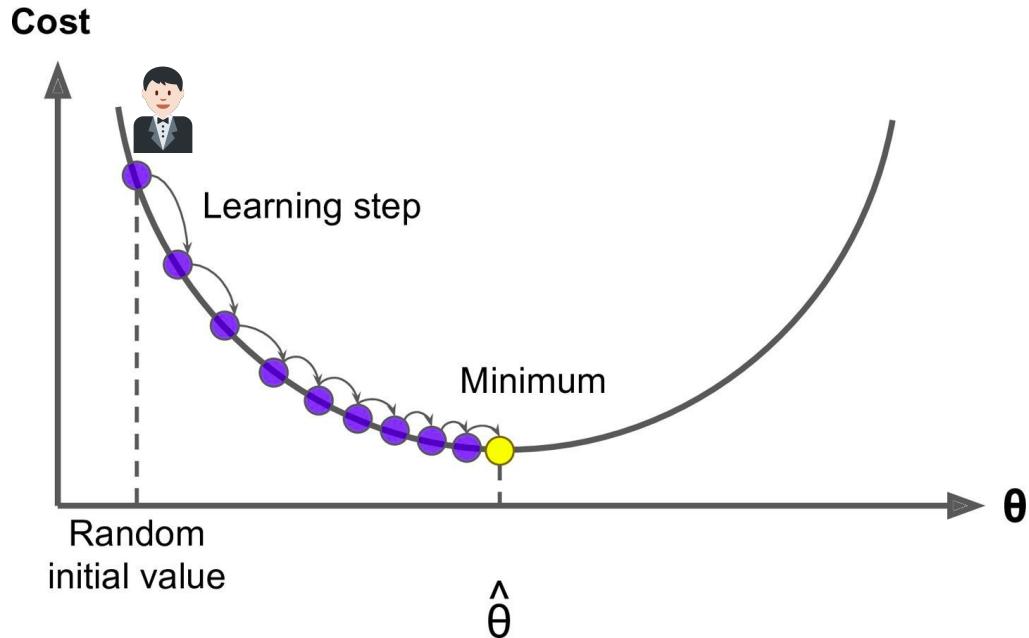
Intuition of Gradient Descent

Answer

1. Take a step
2. If the step is downhill, keep walking on the same direction
3. Else reverse the direction
4. Repeat the steps until you reach to the bottom

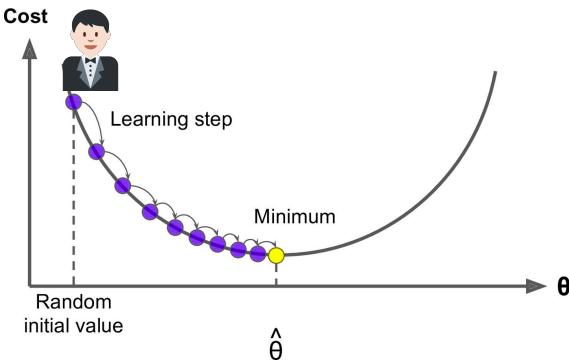


Gradient Descent



Gradient Descent

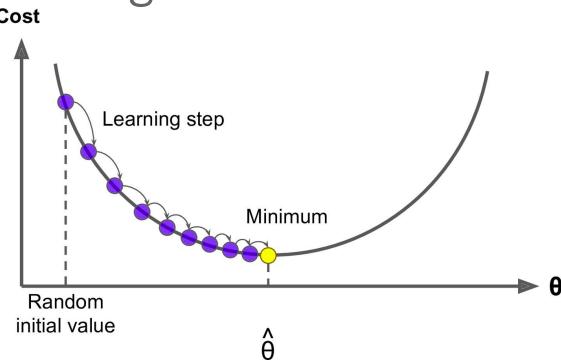
- Gradient Descent
 - Measures the local gradient (slope) of the cost function
 - With regards to the parameter vector θ and
 - It goes in the direction of descending gradient
 - Until the gradient becomes zero



Gradient Descent

Random Initialization

- We start by filling θ with random values – Random initialization
- Improve it gradually, taking one baby step at a time
- Each step attempts to decrease the cost function – MSE
- Until the algorithm converges to a minimum



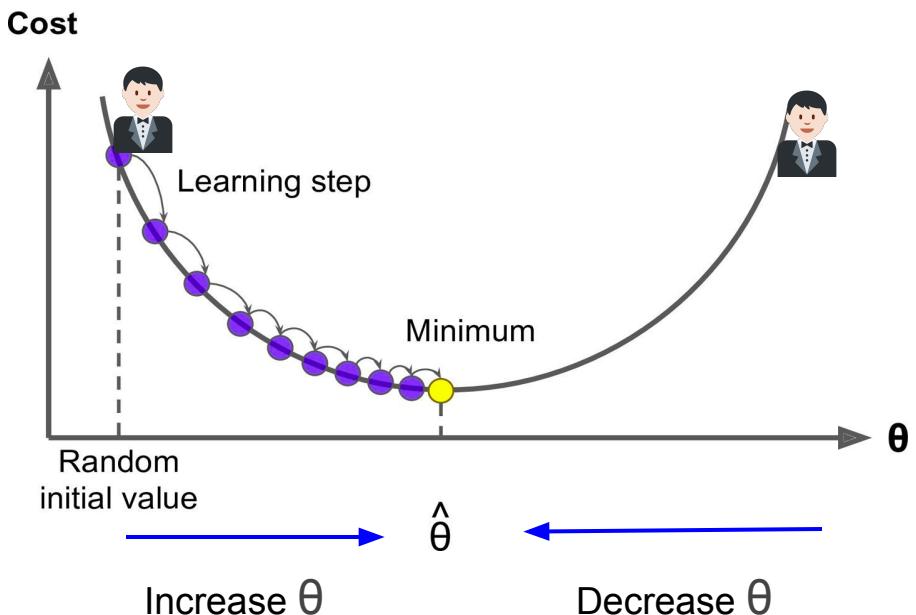
Gradient Descent: finding optimal θ

- Cost function $J(\theta)$ measures the accuracy of the hypothesis function. Objective is to minimize the cost function, that is, make $h(x)$ close to y , at least for the training examples we have.
- Cost function (Mean Square Error (MSE)) is average difference between $h(x)$ and y for all x and y .

$$J(\theta) = \frac{1}{2m} * \left(\sum_{i=1}^{i=m} (h_\theta(x^i) - y^i)^2 \right)$$

Gradient Descent for one of the parameter θ_j for j^{th} feature:

$$\theta_j = \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta)$$





Finding best θ using Optimization Algorithm: Gradient Descent

1. Start with initial guesses
 - Start at ($\theta = 0$) (or any other value).
2. Keeping changing θ a little bit to reduce $J(\theta)$
//Each time you change the parameters, you select the gradient (slope) which reduces $J(\theta)$ the most possible.
3. Repeat
4. Do so until you converge to a minimum.

Step-2 Formulation:

- Update θ by setting it to $(\theta - \alpha$ times the partial derivative of the cost function with respect to $\theta)$
- This update is simultaneously performed for all values of $j = 0, \dots, n$ //number of features

$$\theta_j = \theta_j - \alpha \frac{\delta}{\delta \theta_j} J(\theta)$$



Partial Derivative of Cost Function

Partial Derivative of Cost Function

- We compute the gradient of the cost function
 - With regards to each model parameter θ_j (Theta j)
- Below is the equation for finding partial derivative of the cost function with respect to parameter θ_j (Theta j)

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Gradient Descent

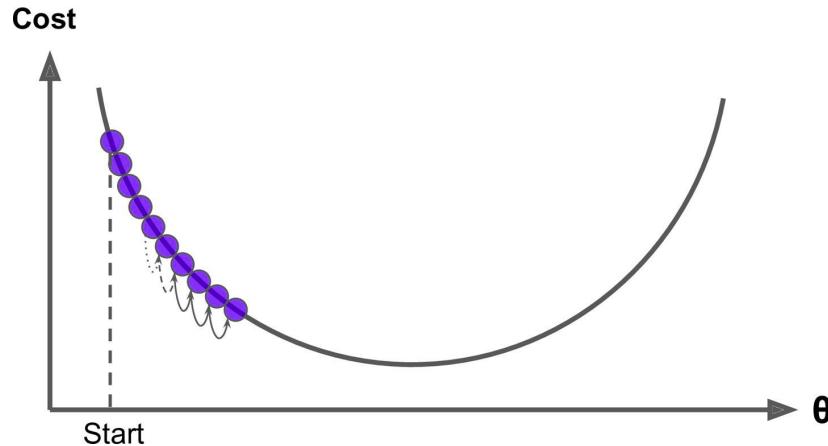
Learning Rate

- In Gradient Descent
 - The size of the steps – also called as **learning rate** plays an important role

Gradient Descent

Learning Rate

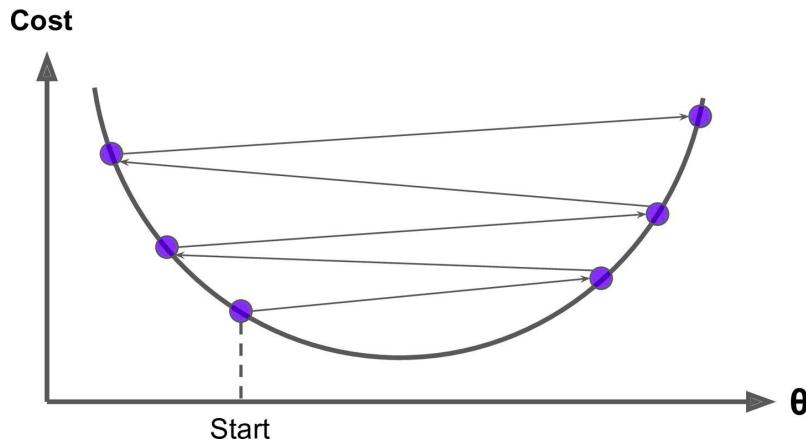
- If the learning rate is too small
 - Then the algorithm will have to go through many iterations to converge
 - Which will take a long time to converge



Gradient Descent

Learning Rate

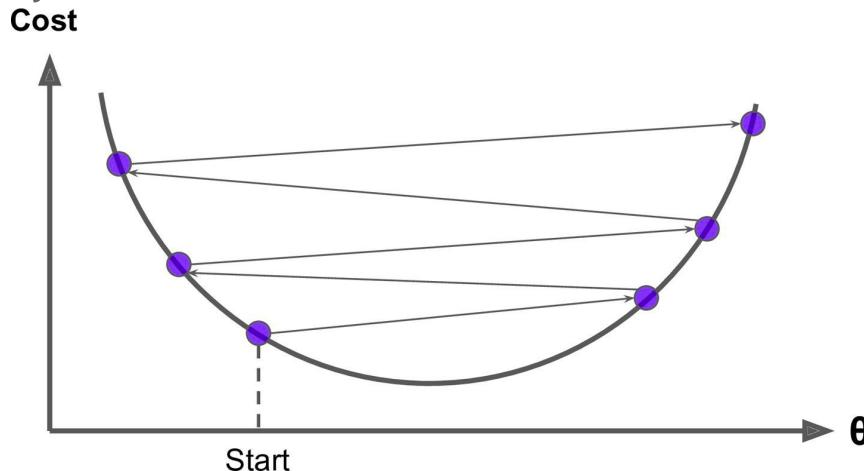
- If the learning rate is too high
 - We might jump across the valley
 - and end up on the other side, possibly even higher up than you were before



Gradient Descent

Learning Rate

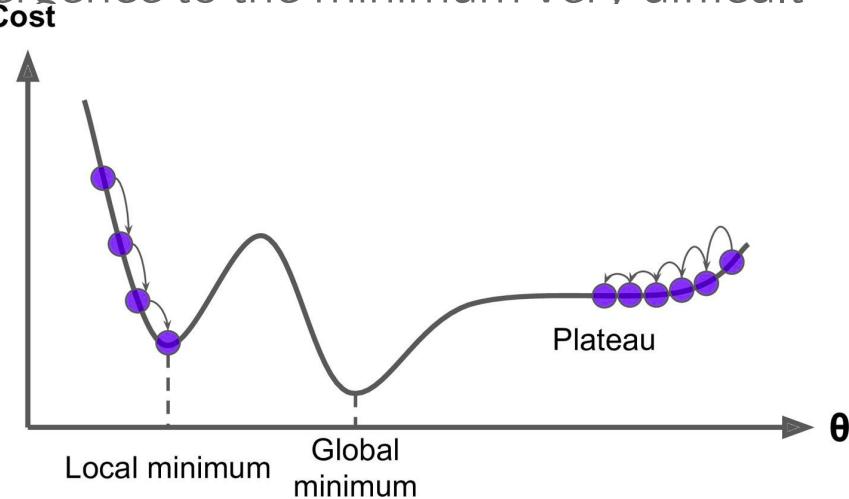
- If the learning rate is too high
 - This might make the algorithm diverge with Larger steps downhill
 - Algorithm may overshoot the minima



Gradient Descent

Global and Local Minimum

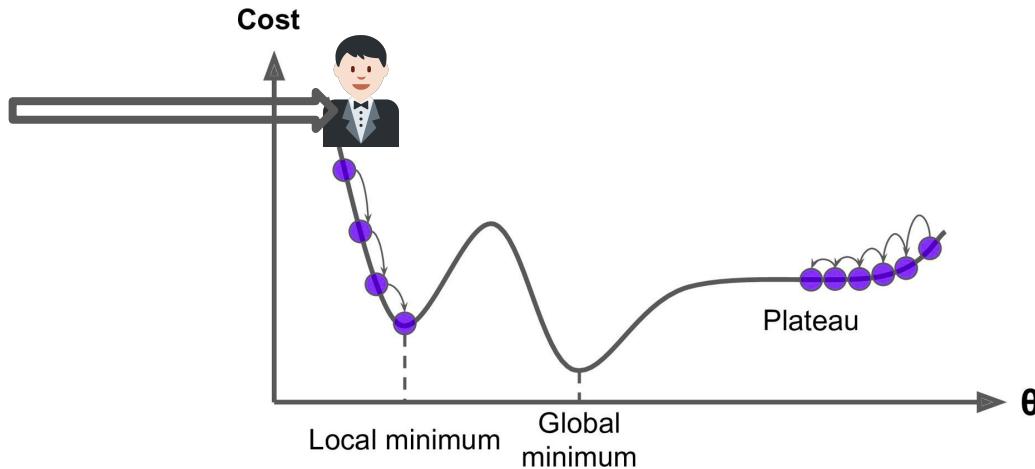
- Not all cost functions are regular bowls
- There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult



Gradient Descent

Global and Local Minimum

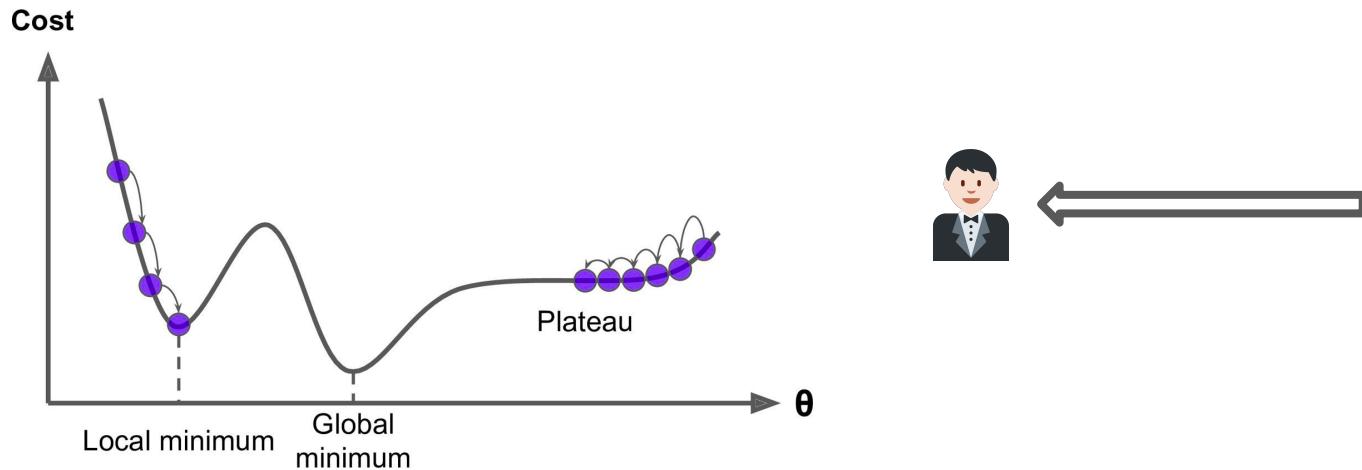
- If we start from left then Gradient Descent
 - Converges to a local minimum
 - Which is not as good as the global minimum



Gradient Descent

Global and Local Minimum

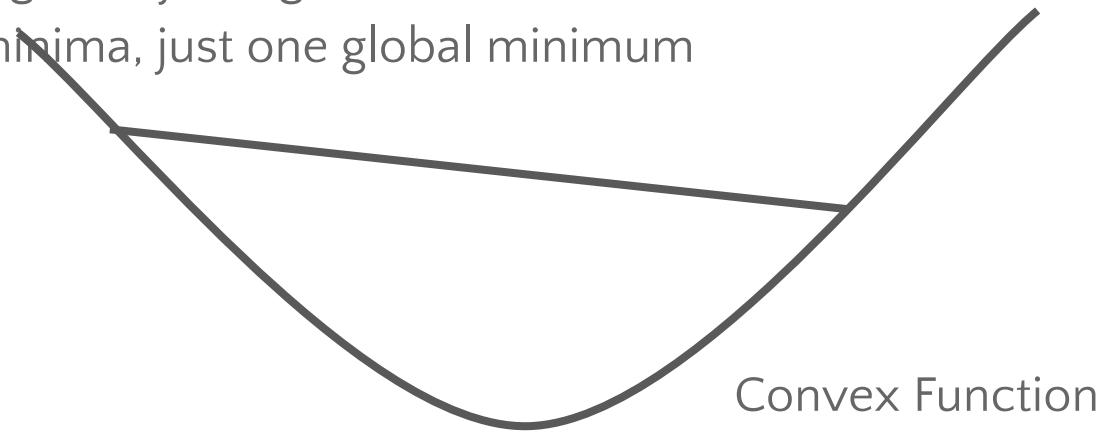
- If we start from right then Gradient Descent
 - Will take a very long time to cross the plateau and
 - If we stop too early then
 - We will never reach the global minimum



Gradient Descent

Global and Local Minimum

- Mean Square Error (MSE) cost function is a convex function (having one minima only)
 - If we pick any two points on the curve
 - The line segment joining them never crosses the curve
 - No local minima, just one global minimum



Obtain θ in one attempt: The Normal Equation (another optimization algorithm)

- It is a mathematical equation that gives the optimal (minimum error or cost) θ directly

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$



Value of θ that minimizes the cost function



y is the vector of target values containing $y^{(1)}$ to $y^{(m)}$

[More on wikipedia](#)

The Normal Equation

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

X is the feature matrix of training dataset

y is the output vector

The Normal Equation

Let's test the equation

The Normal Equation

- First, generate some linear looking data

```
>>> import numpy as np
```

```
# Uniformly distributed array with dimension (100, 1)
```

```
>>> X = 2 * np.random.rand(100, 1)
```

```
# A variable y which a linear function of X
```

```
>>> y = 4 + 3 * X + np.random.randn(100, 1)
```

$$\theta_0 + \theta_1 X$$

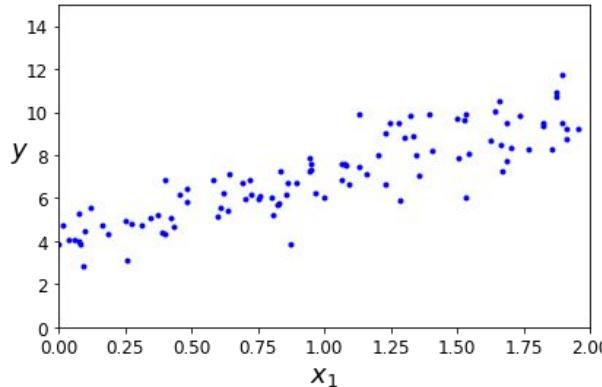
Gaussian Noise

The Normal Equation

- Let plot the chart to see if X and y have linear relation

```
>>> plt.plot(X, y, "b.")  
>>> plt.xlabel("$x_1$", fontsize=18)  
>>> plt.ylabel("$y$", rotation=0, fontsize=18)  
>>> plt.axis([0, 2, 0, 15])  
>>> plt.show()
```

Output



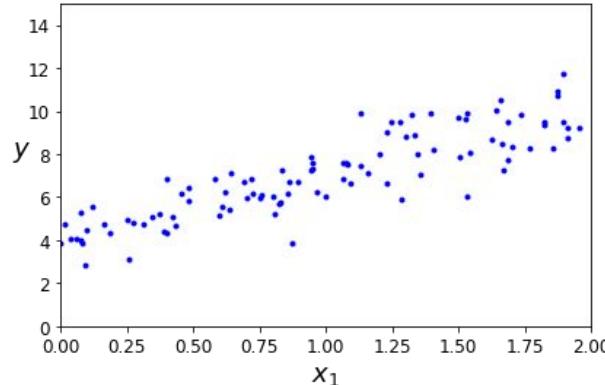
The Normal Equation

- Let plot the chart to see if X and y have linear relation

```
>>> plt.plot(X, y, "b.")  
>>> plt.xlabel("$x_1$", fontsize=18)  
>>> plt.ylabel("$y$", rotation=0, fontsize=18)  
>>> plt.axis([0, 2, 0, 15])  
>>> plt.show()
```

Output

X and y have linear
relation



The Normal Equation

$$\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$$

- Let's compute theta hat using the Normal Equation

Add Bias term $X_0 = 1$ ($w_0 X_0 + w_1 \cdot X$), so there won't be any effect of X_0 in computation

```
>>> X_b = np.c_[np.ones((100, 1)), X]
```

Apply the normal equation

```
>>> theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
array([[ 4.14766894],  
       [ 2.8143552 ]])
```

The Normal Equation

Weights in Original Equation

```
y = 4 + 3 * X + np.random.randn(100, 1)
```

Original

$$\theta_0 = 4$$

$$\theta_1 = 3$$

Learned Weights from Normal Equation

```
array([[ 4.14766894],  
       [ 2.8143552 ]])
```

From Normal Equation

$$\theta_0 = 4.14$$

$$\theta_1 = 2.81$$

The Normal Equation

Weights in Original Equation

```
y = 4 + 3 * X + np.random.randn(100, 1)
```

Original

$$\theta_0 = 4$$

$$\theta_1 = 3$$

Learned Weights from Normal Equation

```
array([[ 4.14766894],  
       [ 2.8143552 ]])
```

From Normal Equation

$$\theta_0 = 4.14$$

$$\theta_1 = 2.81$$

Not bad!

Learned weights are approximately same as original weights

The Normal Equation

- Let's make predictions using theta_best

```
>>> X_new = np.array([[0], [2]])
```

```
# add x0 = 1 to each instance
```

```
>>> X_new_b = np.c_[np.ones((2, 1)), X_new]
```

```
>>> y_predict = X_new_b.dot(theta_best)
```

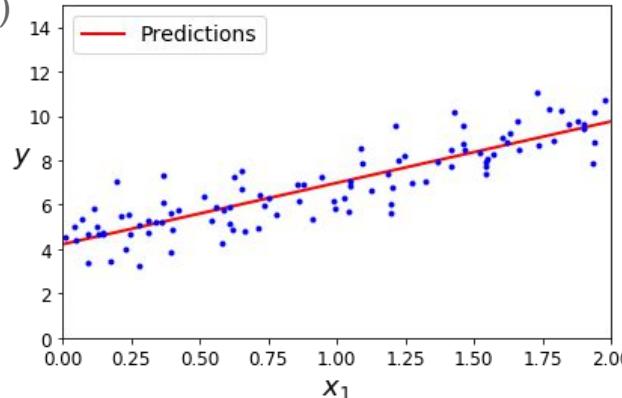
Output-

```
array([[ 4.21509616],  
       [ 9.75532293]])
```

The Normal Equation

- Let's plot the models predictions

```
>>> plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
>>> plt.plot(X, y, "b.")
>>> plt.xlabel("$x_1$", fontsize=18)
>>> plt.ylabel("$y$", rotation=0, fontsize=18)
>>> plt.legend(loc="upper left", fontsize=14)
>>> plt.axis([0, 2, 0, 15])
>>> plt.show()
```



Output-

The Normal Equation

- We can also use Scikit-Learn for the previous steps

```
>>> from sklearn.linear_model import LinearRegression  
>>> lin_reg = LinearRegression()  
>>> lin_reg.fit(X, y)  
>>> lin_reg.intercept_, lin_reg.coef_
```

Output-

```
(array([ 4.21509616]), array([[ 2.77011339]]))
```

The Normal Equation

```
>>> lin_reg.predict(X_new)
```

Output-

```
array([[ 4.21509616],  
       [ 9.75532293]])
```

The Normal Equation

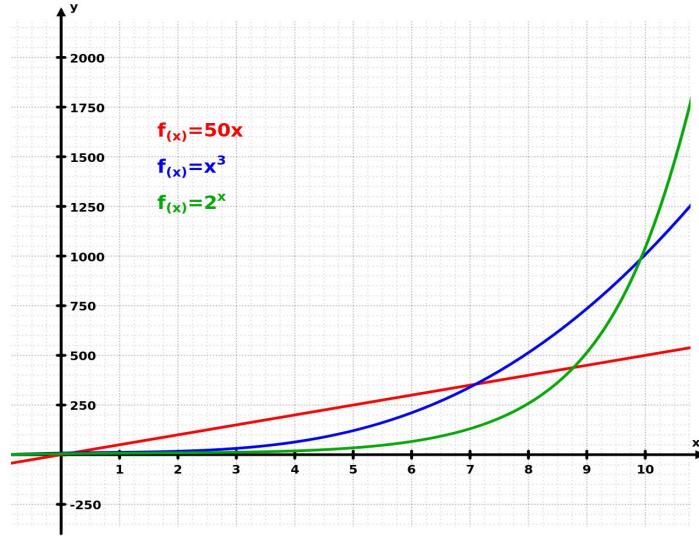
Computational Complexity – For Features

- The Normal Equation
 - Computes the inverse of $X^T \cdot X$
 - Which is an $n * n$ matrix (where n is the number of features)
- Depending on the the number of features
 - The computational complexity of inverting such a matrix is very high
 - Typically about $O(n^{2.4})$ to $O(n^3)$ depending on the implementation
 - Use gradient descent when n is large

The Normal Equation

Computational Complexity – For Features

- If we double the number of features
 - Then the computation time gets increased by
 - $2^{2.4}$ to 2^3 = ($\sim=$ 5.3 to 8) times
- That is why the Normal Equation gets very slow
 - When the number of features grows large



The Normal Equation

Computational Complexity – For Training Set

- The normal equation is
 - Linear with regards to the number of instances in the training set
 - So it handles large training sets efficiently
 - Provided they can fit in memory

The Normal Equation

Computational Complexity

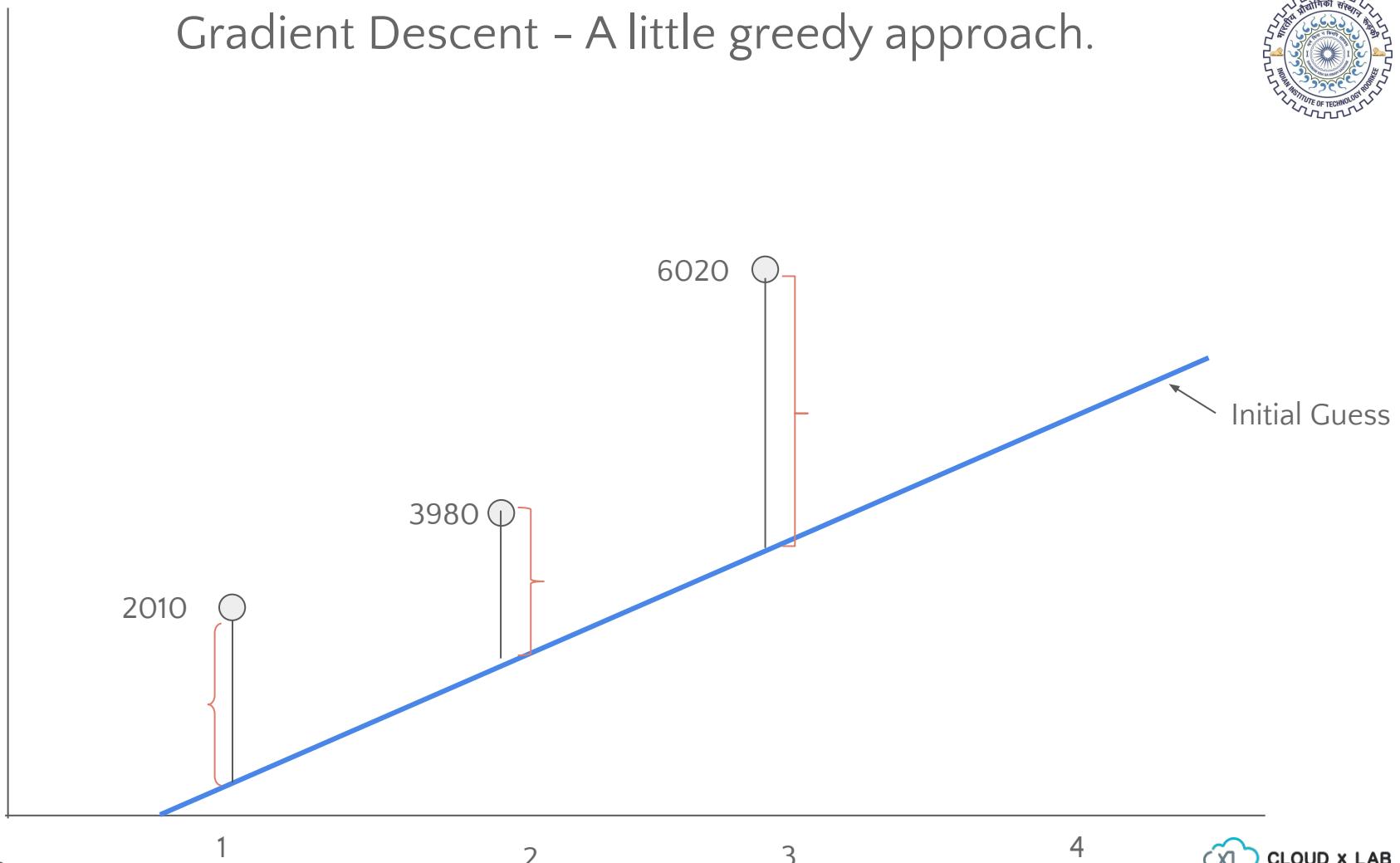
- Let's explore different ways to train a Linear Regression model which are
 - Better suited for cases where there are a large number of features
 - Or too many training instances to fit in memory

Better way to Train Linear Regression Model



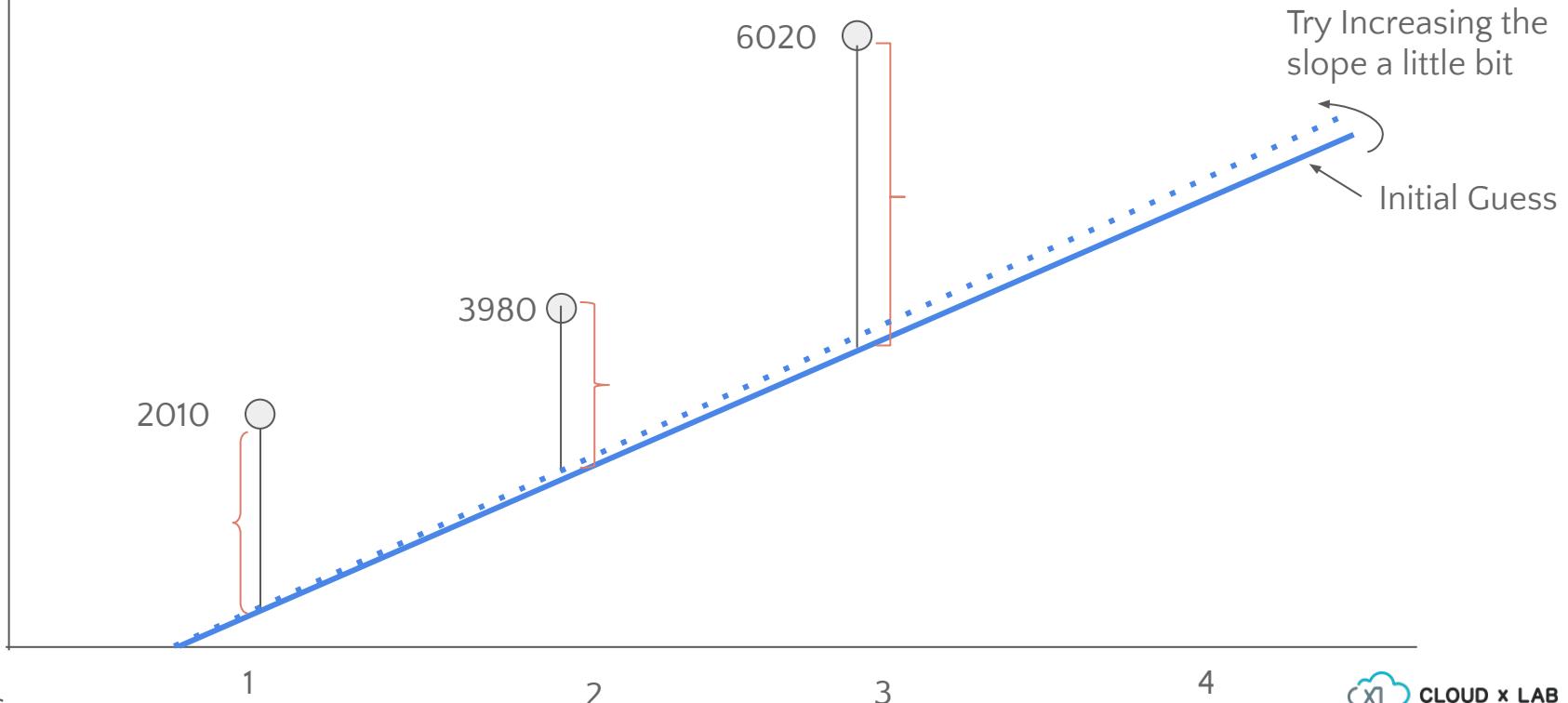
Gradient Descent - Tweaks parameters iteratively in
order to minimize a cost function

Gradient Descent - A little greedy approach.



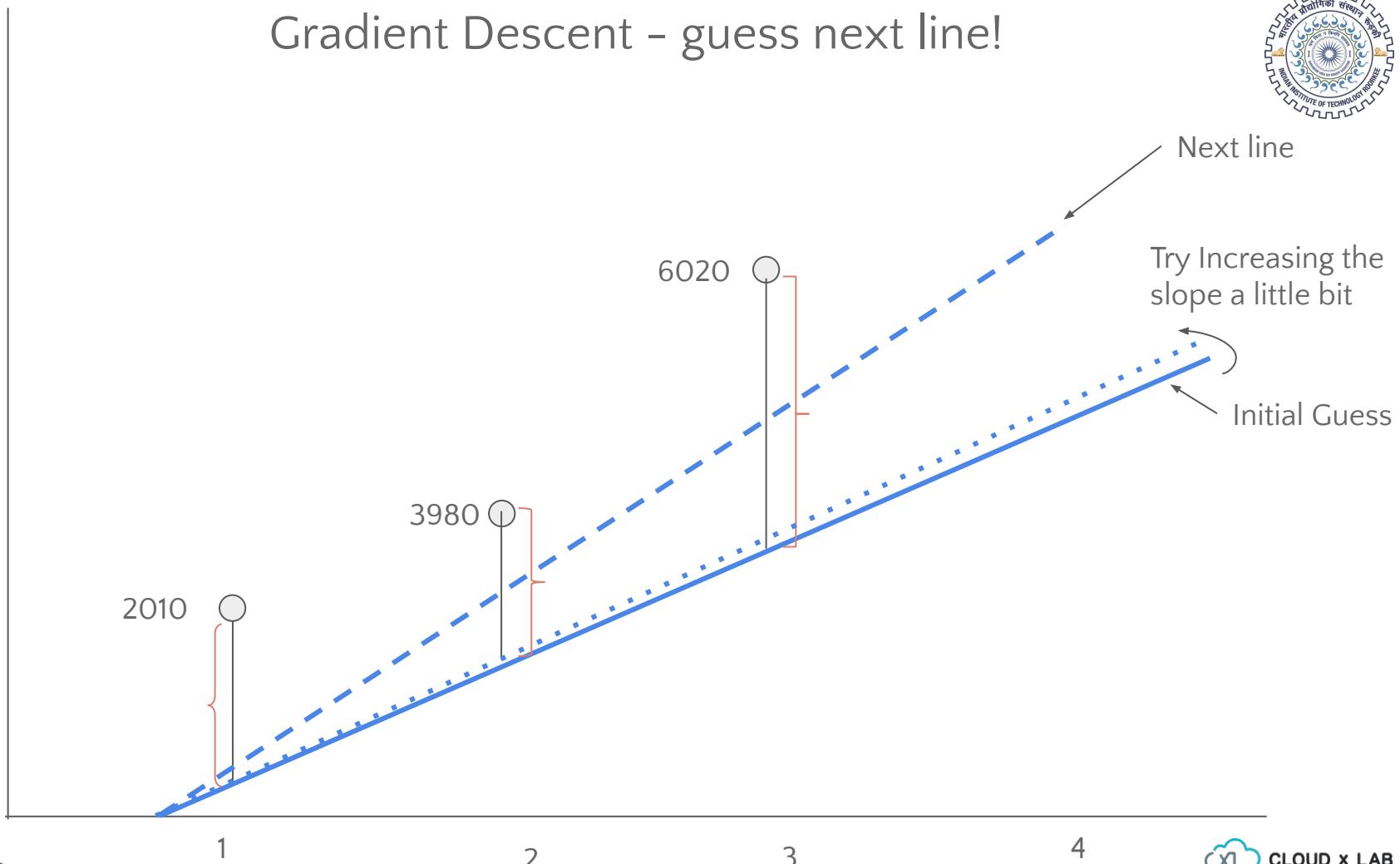


Gradient Descent - find impact of slight change



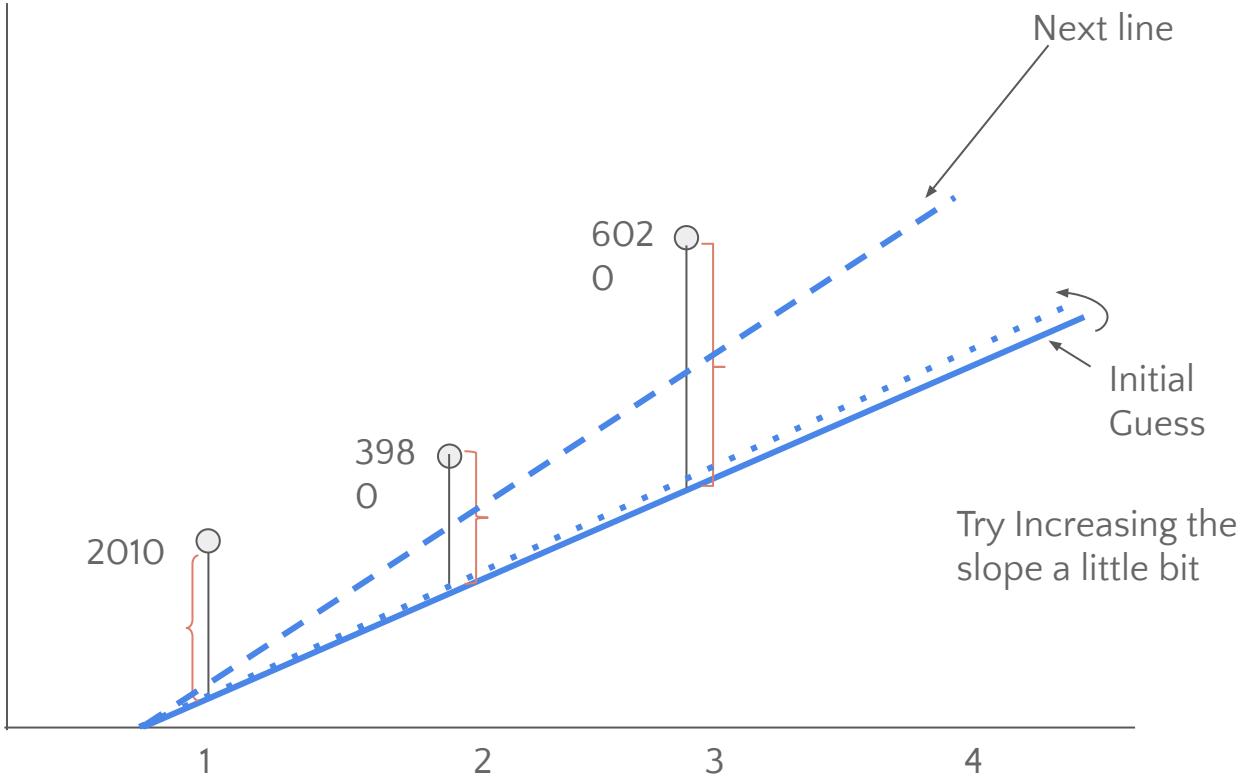


Gradient Descent - guess next line!



Gradient Descent

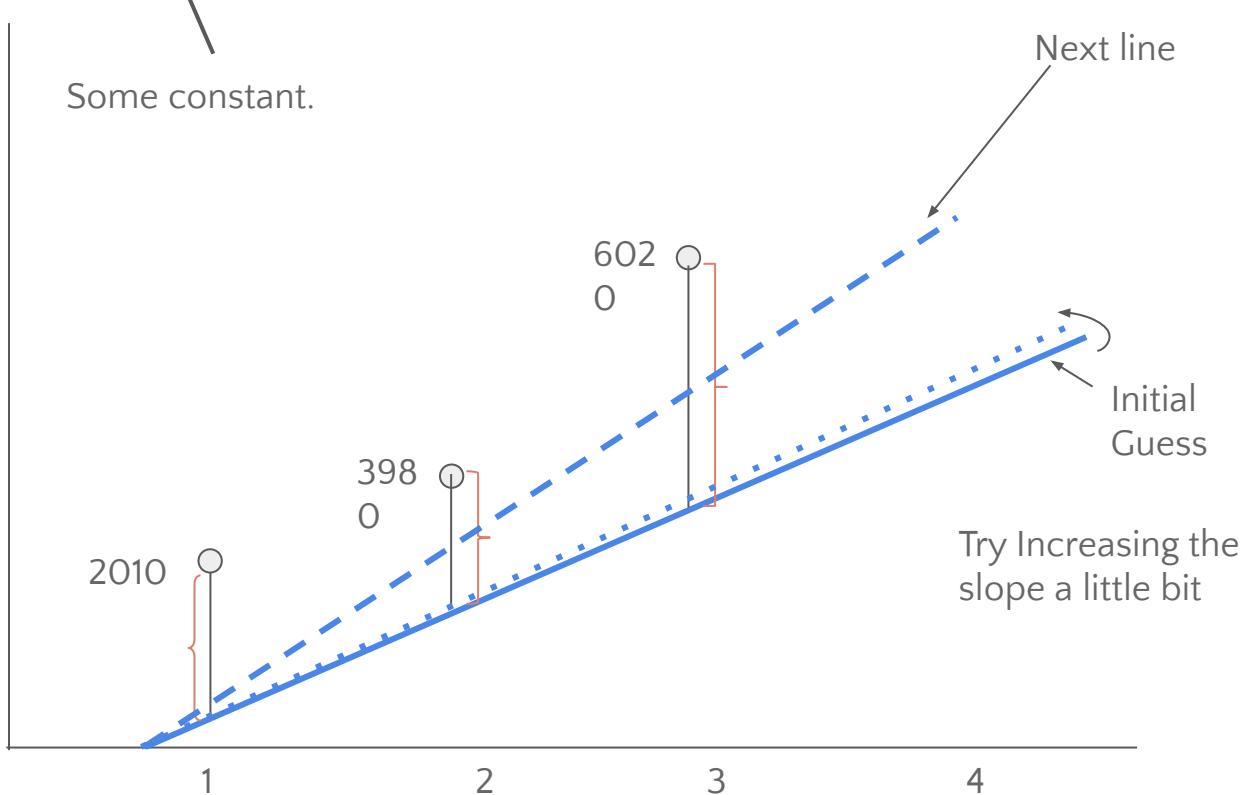
Increase in slope is proportional to Rate of decrease of error w.r.t change in slope





Gradient Descent

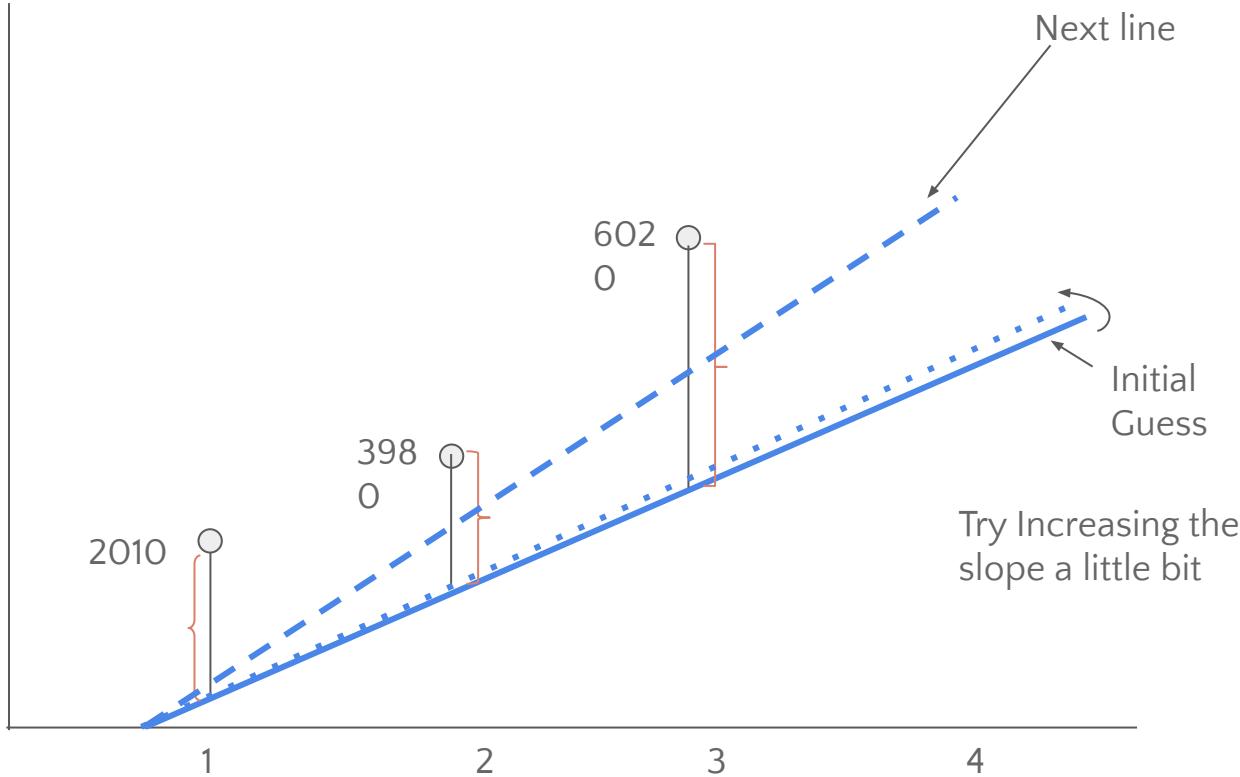
Increase in slope = learning rate * Rate of decrease of error wrt change in slope

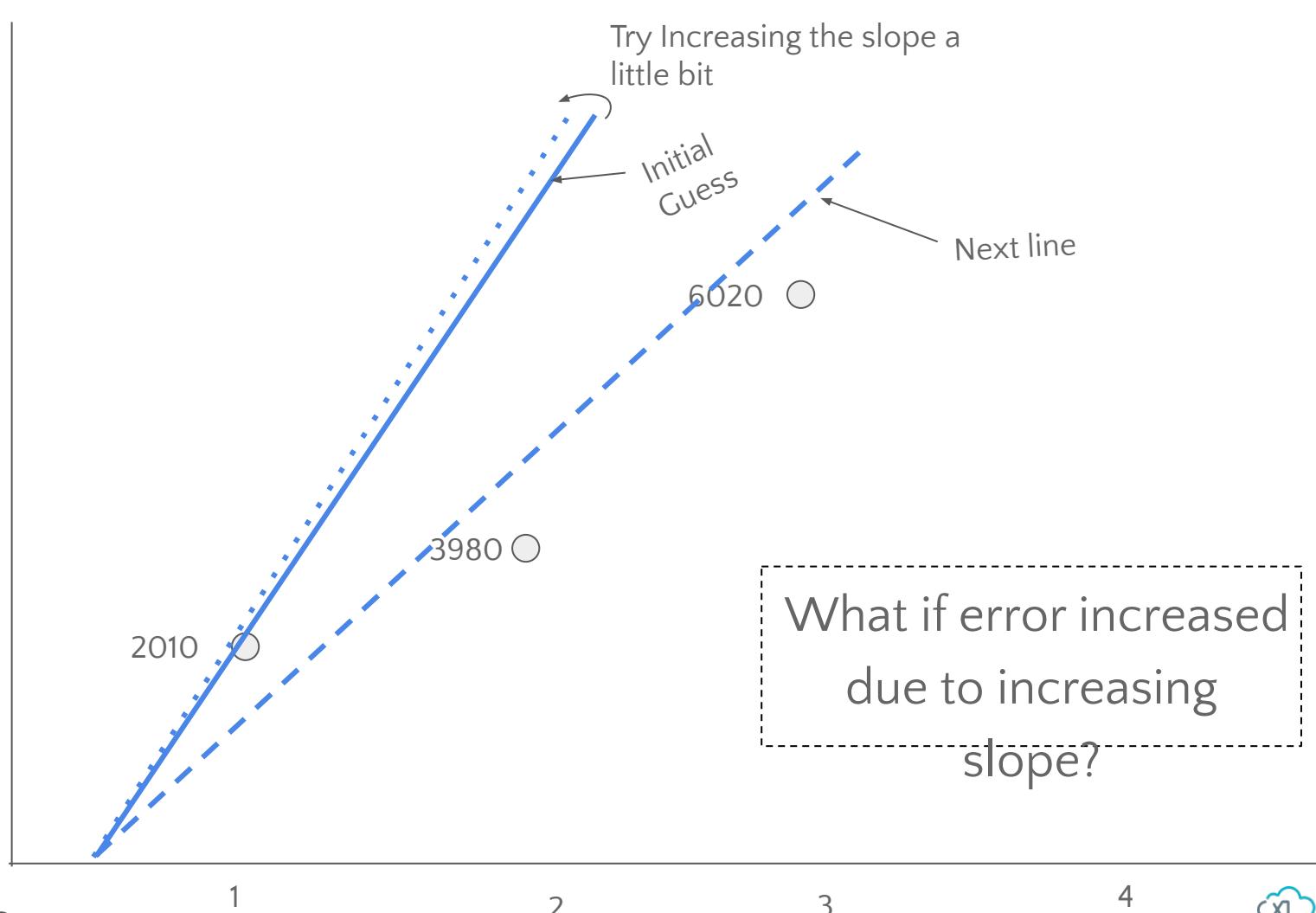




Gradient Descent

New Slope = Old Slope + Learning Rate * Rate of decrease of error wrt change in slope





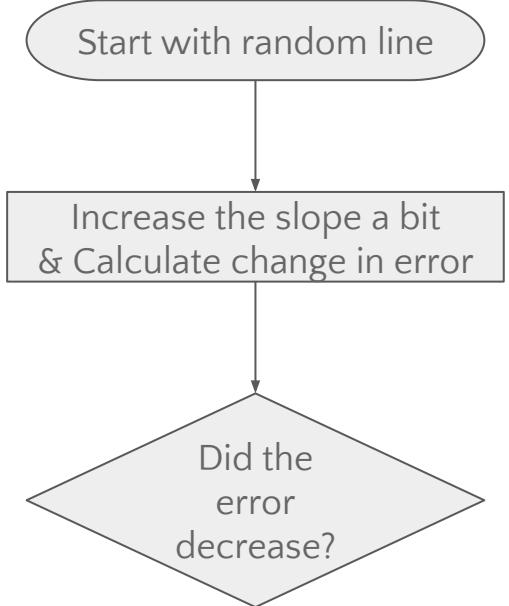


Gradient Descent

Start with random line

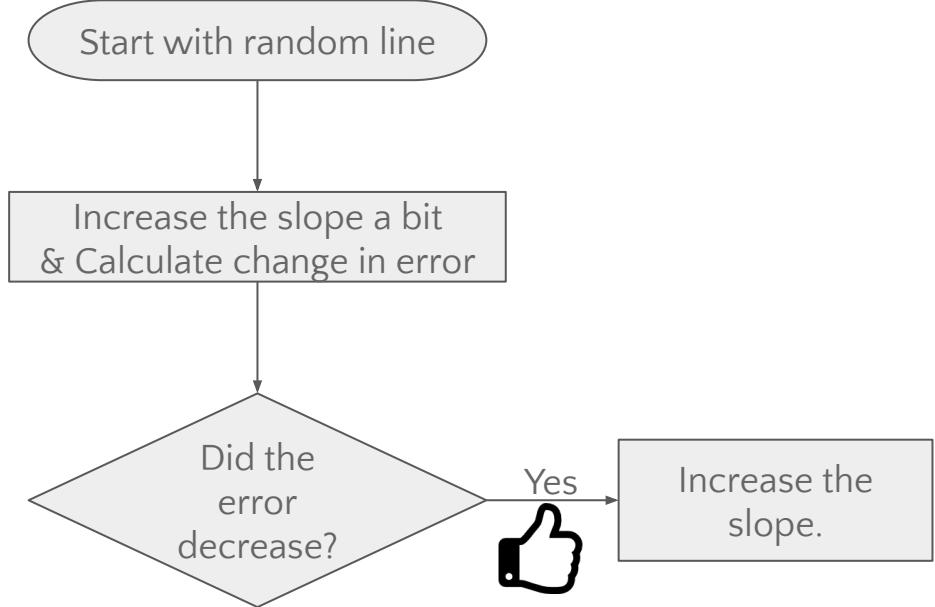


Gradient Descent



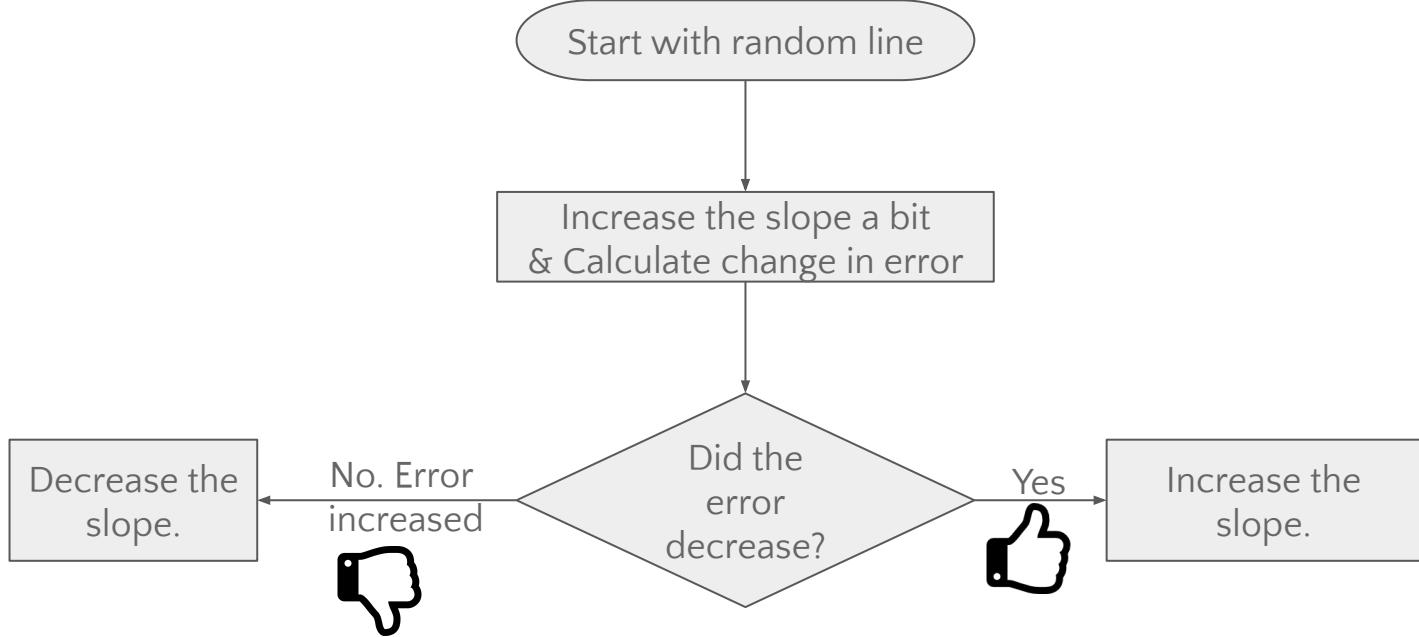


Gradient Descent

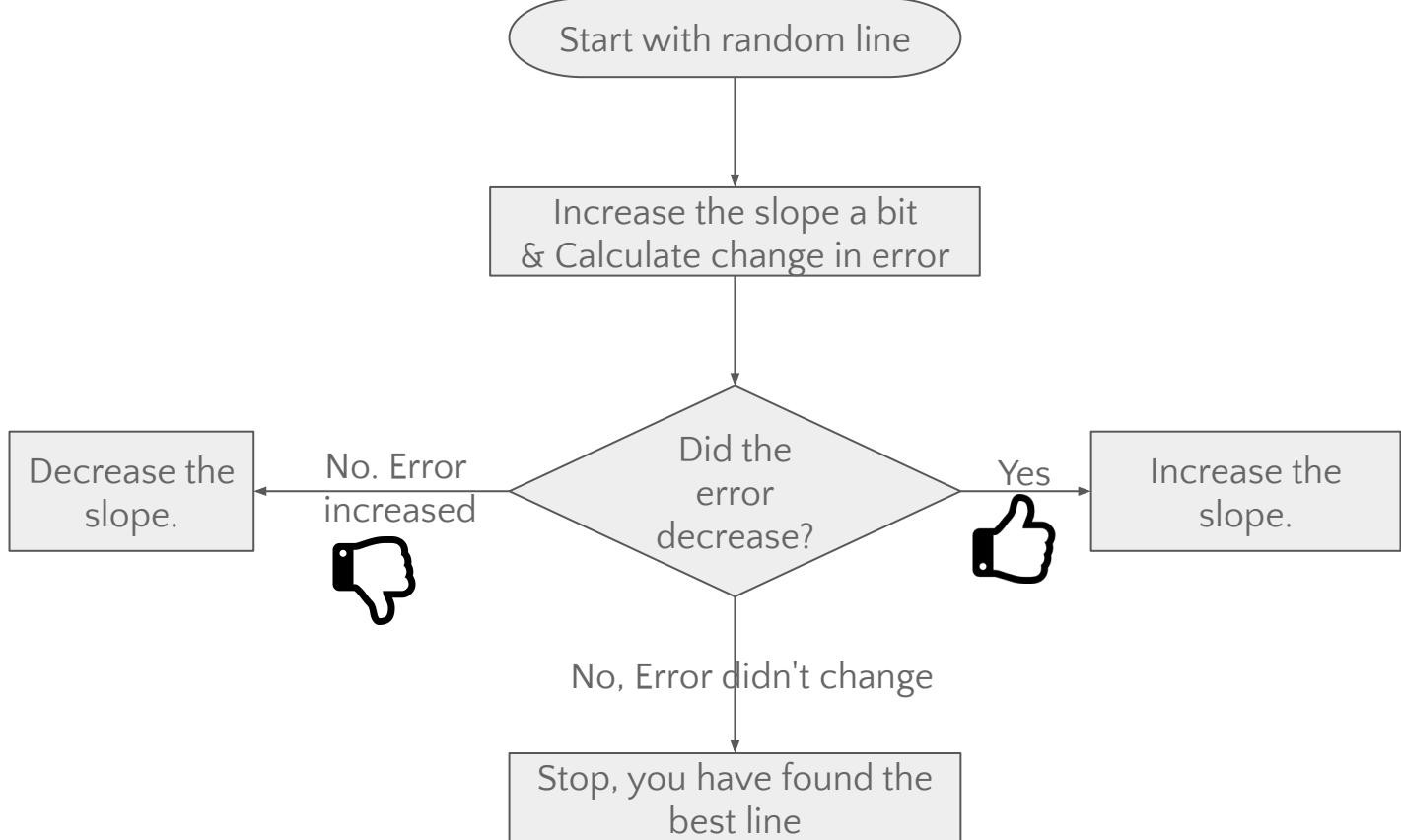




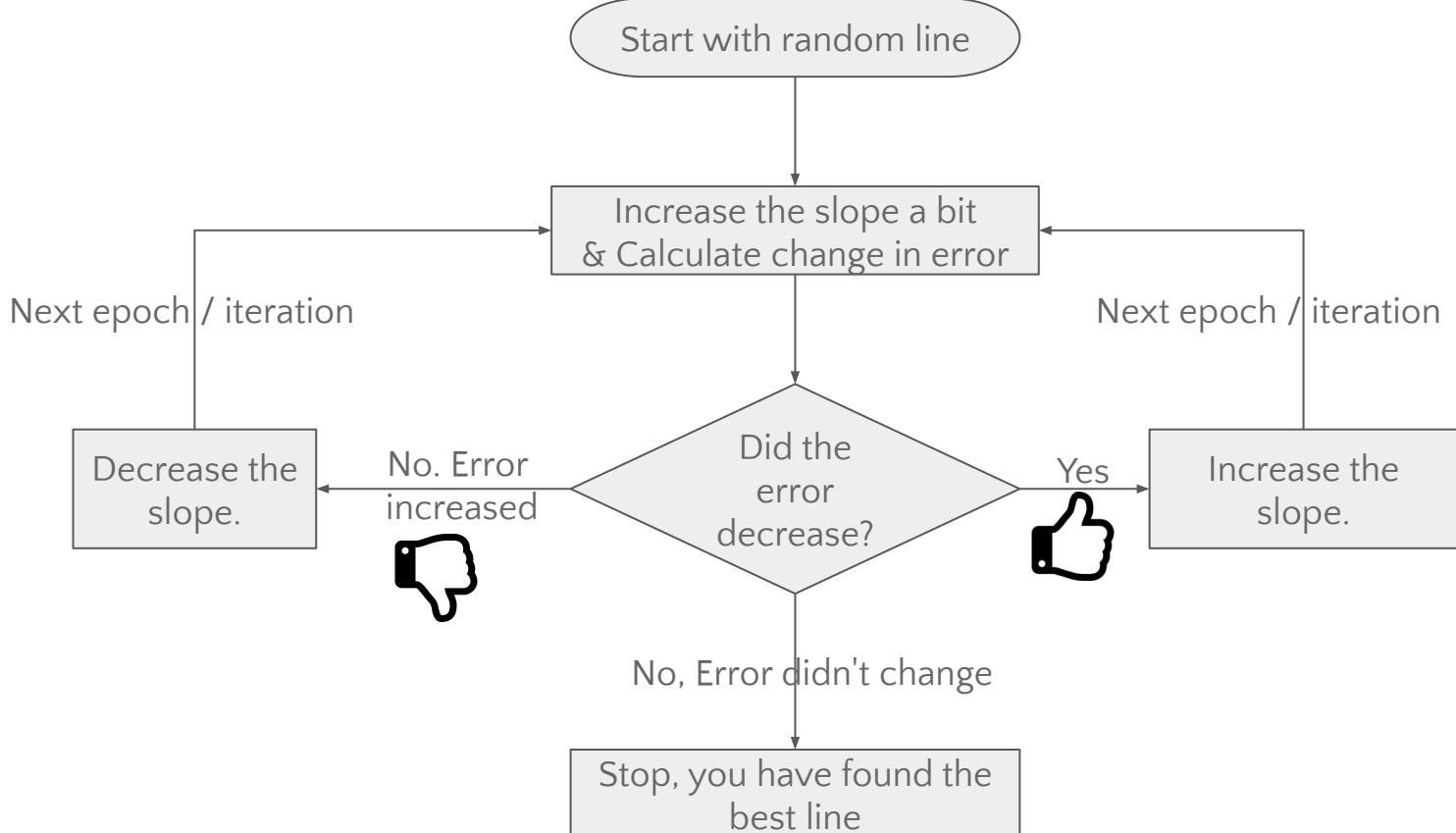
Gradient Descent



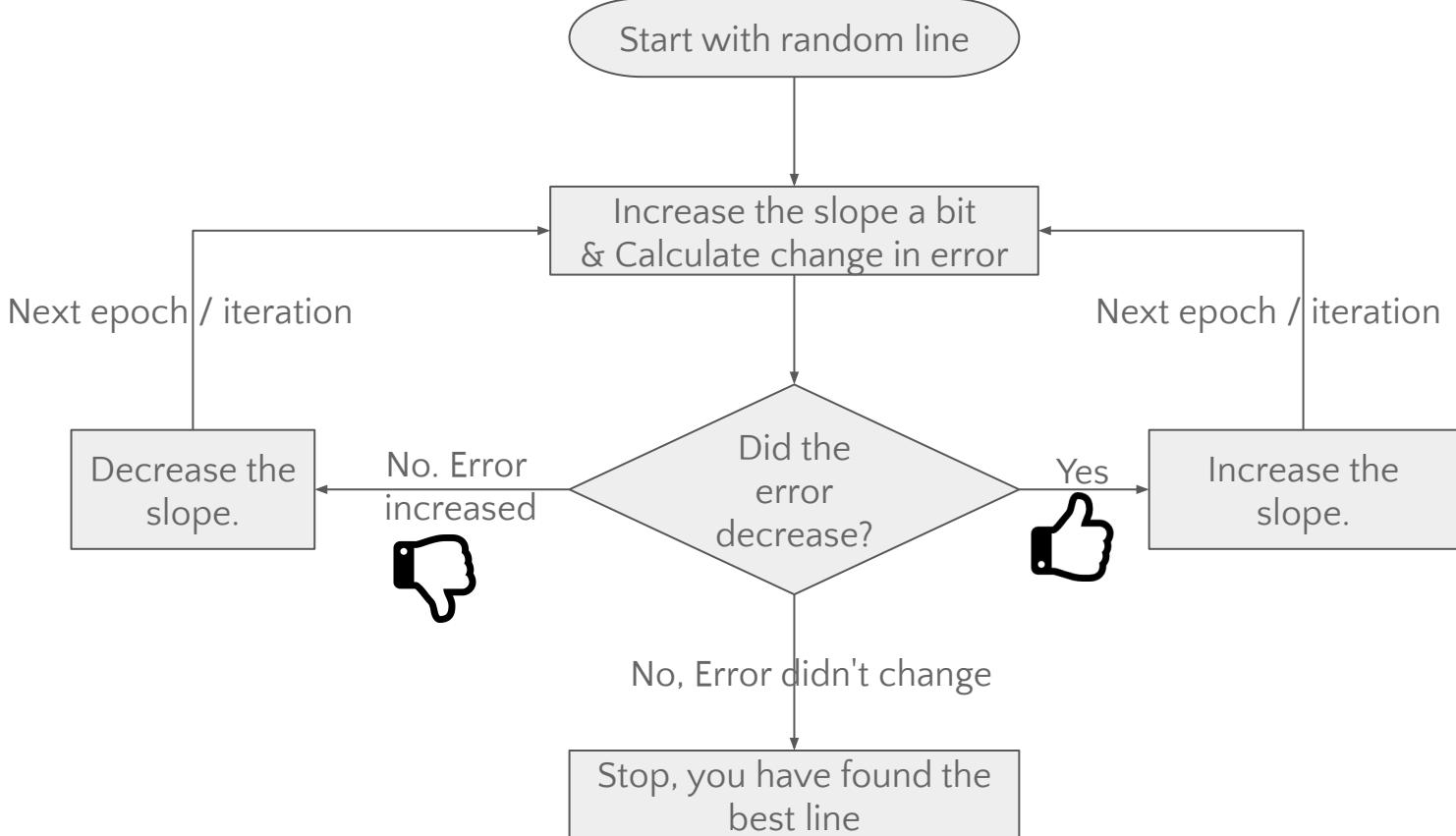
Gradient Descent



Gradient Descent



Gradient Descent



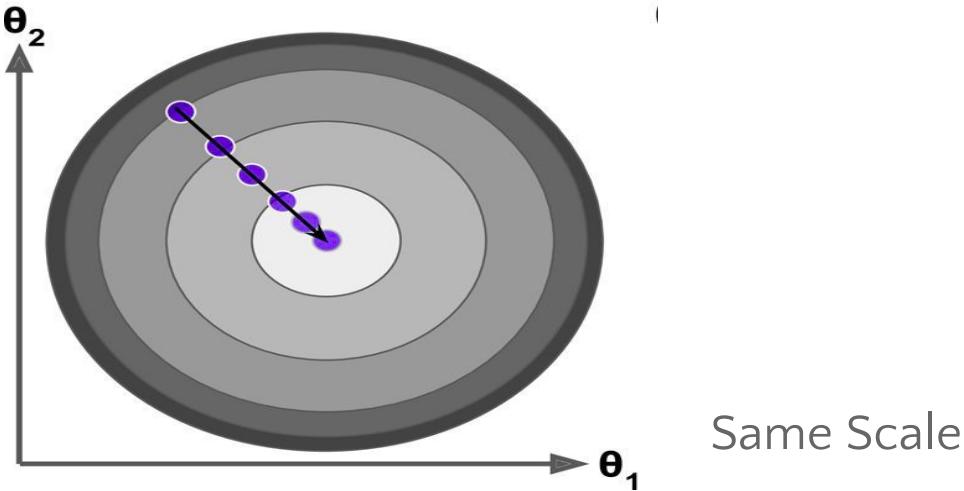
Feature Scaling

- When using Gradient Descent, we should ensure that
 - All features have a similar scale
 - Using Scikit-Learn's **StandardScaler** class
 - Else it will take much longer to converge

Feature Scaling

Feature Scaling – Same Scales

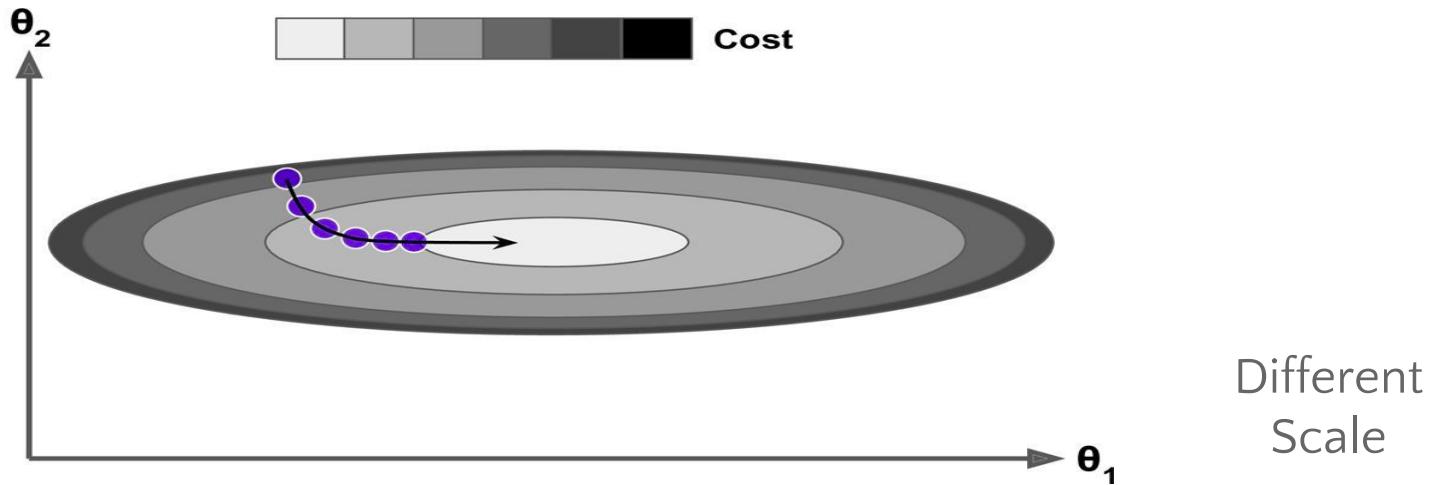
- See the below diagram where feature 1 and 2 have same scales
- Gradient Descent algorithm goes straight toward the minimum
 - Thereby reaching it quickly



Feature Scaling

Feature Scaling – Different Scales / Not scaled

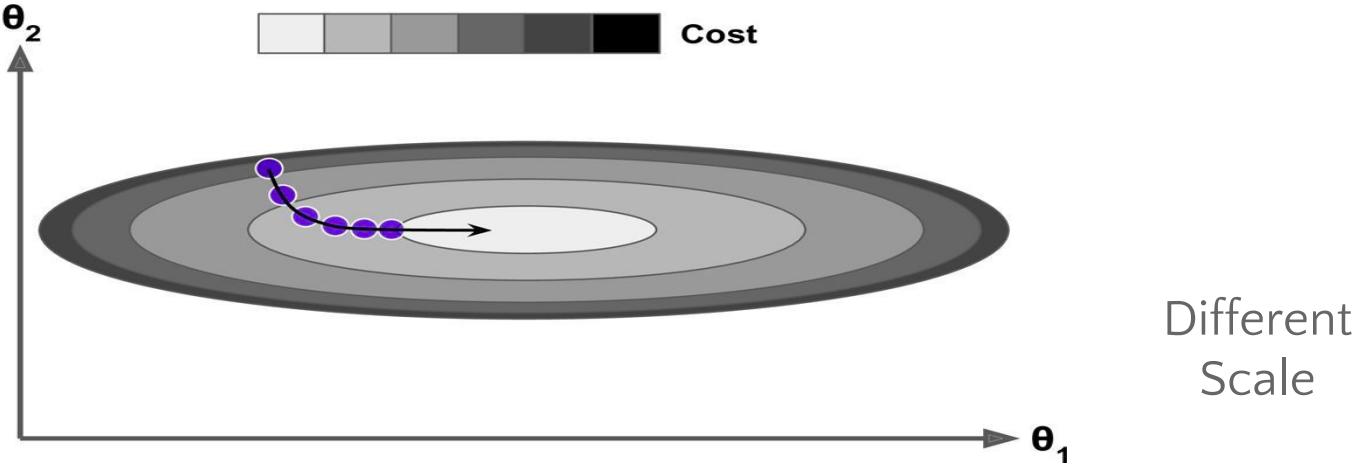
- See the below diagram where feature 1 and 2 have different scales



Feature Scaling

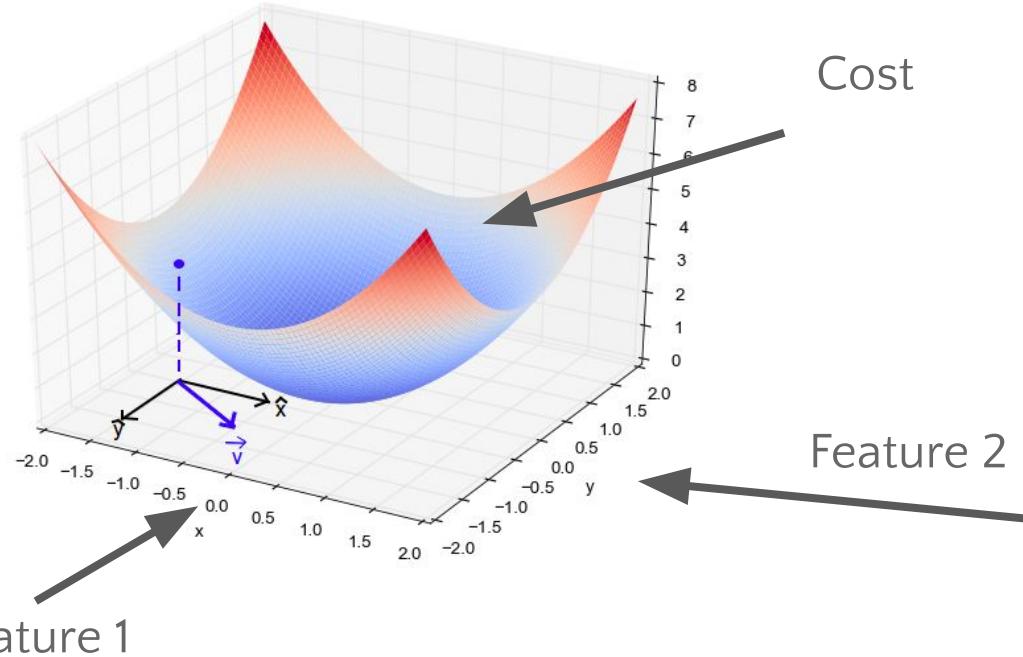
Feature Scaling - Different Scales

- Gradient Descent algorithm goes almost orthogonal to
 - The direction of the global minimum and
 - It ends with a long march down an almost flat valley
 - It will eventually reach the minimum, but it will take a long time



Gradient Descent

Features and Cost Function



Scaling: Normalization of Features

- Features should be on the same scale, else Gradient descent may take a long time to converge.

Example scenario: No of bedrooms and Size of a house

$$X_1 = (x_1 - \text{mean}(x_1)) / \text{std}(x_1)$$

$$X_2 = (x_2 - \text{mean}(x_2)) / \text{std}(x_2)$$

.....

$$z = \frac{x - \text{mean}(x)}{\text{stdev}(x)}$$

$$X_1 = (x_1 - \text{min}(x_1)) / (\text{max}(x_1) - \text{min}(x_1))$$

$$X_2 = (x_2 - \text{min}(x_2)) / (\text{max}(x_2) - \text{min}(x_2))$$

.....

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

$$z = \frac{1}{1 + \exp(-x)}$$

// Logistic
Normalization

Gradient Descent

- Again remember that training a model means
 - Searching for a best possible model's parameters
 - Like θ_0 , θ_1 and others
 - That minimizes the cost function like MSE
- Algorithm will have to search in more dimensions if
 - Model has more features
 - And search becomes complex



Types of Gradient Descent based on size of input data

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent



Types of Gradient Descent based on part of the input data to update parameter

Batch Gradient Descent: Looks at every example in the entire training set before taking a single step. //Can be very costly operation if m is too large.

Repeat until convergence

$$\theta_j = \theta_j - \alpha * \left(\sum_{i=1}^{i=m} ((h_\theta(x^i) - y^i) * x_j^i) \right)$$

Stochastic Gradient Descent: Update θ with each training example

Repeat until convergence

$$\theta_j = \theta_j - \alpha * (h_\theta(x^i) - y^i) * x_j^i$$



Stochastic Vs Batch

- Stochastic gradient descent can get θ close to the minimum much faster than batch gradient descent.
- But may not converge to minima.
- However, approximation of the true minima is reasonably good in practice.
Learning rate can help.



Batch Gradient Descent

Partial Derivative

- To implement Gradient Descent
 - First initialize model parameters($\theta_1, \theta_2, \dots$) - Random Initialization
 - For each of the weights, we change one weight and keeping others fixed and calculate the change in cost
 - This is called a **Partial Derivative**



Batch Gradient Descent

Partial Derivative

- It is like asking
 - What is the slope of the mountain under my feet if I face east?
 - And then asking the same question facing north
 - And so on for all other dimensions



Batch Gradient Descent

Partial Derivative of Cost Function

- In general, we compute the gradient of the cost function
 - With regards to each model parameter



Batch Gradient Descent

Partial Derivative of Cost Function

- In this equation
 - We calculate partial derivative wrt each parameter θ_j (Theta j) individually
 - Instead we can compute them all in one go

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$



Batch Gradient Descent

Gradient Vector of the Cost Function

- The gradient vector $\nabla_{\theta} \text{MSE}(\theta)$
 - Contains all the partial derivatives of the cost function
 - (one for each model parameter)

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \boldsymbol{\theta} - \mathbf{y})$$



Batch Gradient Descent

Gradient Vector of the Cost Function

- The gradient vector $\nabla_{\theta} \text{MSE}(\theta)$
 - Contains all the partial derivatives of the cost function
 - (one for each model parameter)



Batch Gradient Descent

Gradient Vector of the Cost Function

- This formula involves calculations over the full training set X at every step
- This is why the name **Batch Gradient Descent**



Batch Gradient Descent

Gradient Vector of the Cost Function

- This formula involves calculations over the full training set \mathbf{X} at every step
- This is why the name **Batch Gradient Descent**

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$



Batch Gradient Descent

Gradient Vector of the Cost Function

- Since Batch Gradient Descent uses entire training data at every step
 - It is terribly slow on large training sets
- But Gradient Descent scales well with the number of features
- It is faster to train Linear Regression model when there are thousand of features
 - Using Gradient Descent than using the Normal Equation

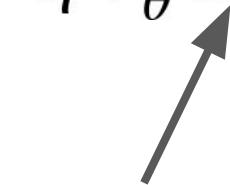


Batch Gradient Descent

Gradient Descent Step

- Once we have the gradient vector, which points uphill,
 - Just go in the opposite direction to go downhill
- We can calculate the size of the downhill step using below equation

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$



Learning Rate

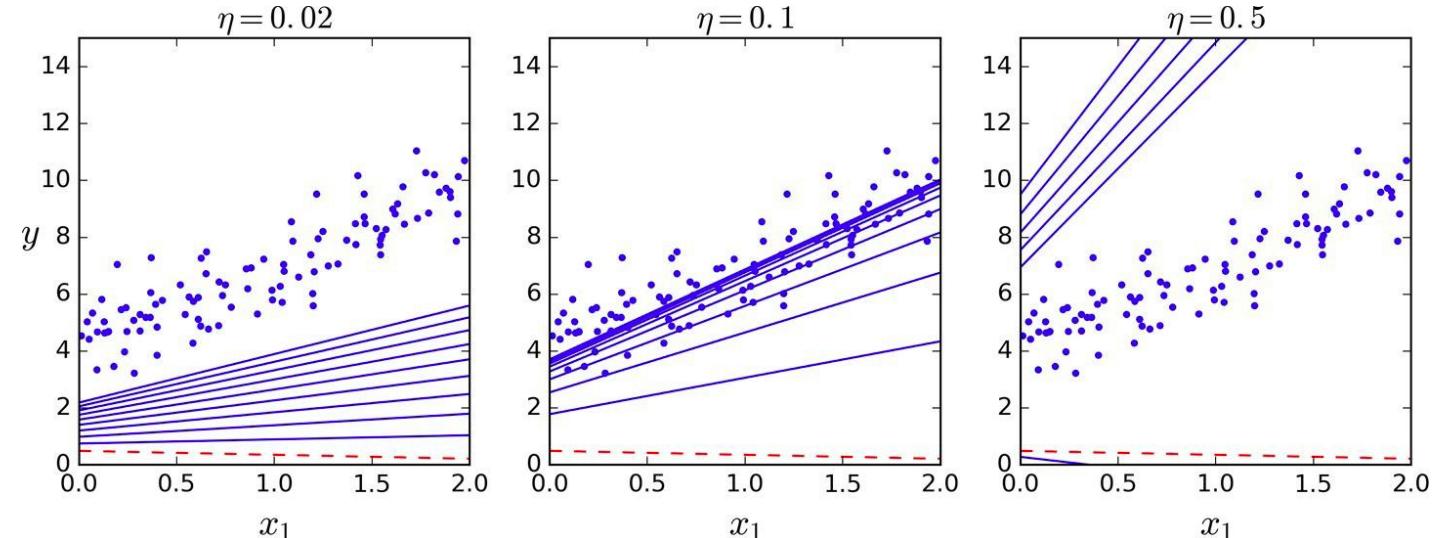


Batch Gradient Descent

Check the code of Batch Gradient Descent in Notebook

Batch Gradient Descent

- First 10 steps of Gradient Descent using three different learning rates
- The dashed line - - - - represents the starting point



Gradient Descent with Various Learning Rates (See the code in notebook)



Stochastic Gradient Descent (SGD)



Stochastic Gradient Descent - SGD

Problem with Batch Gradient Descent

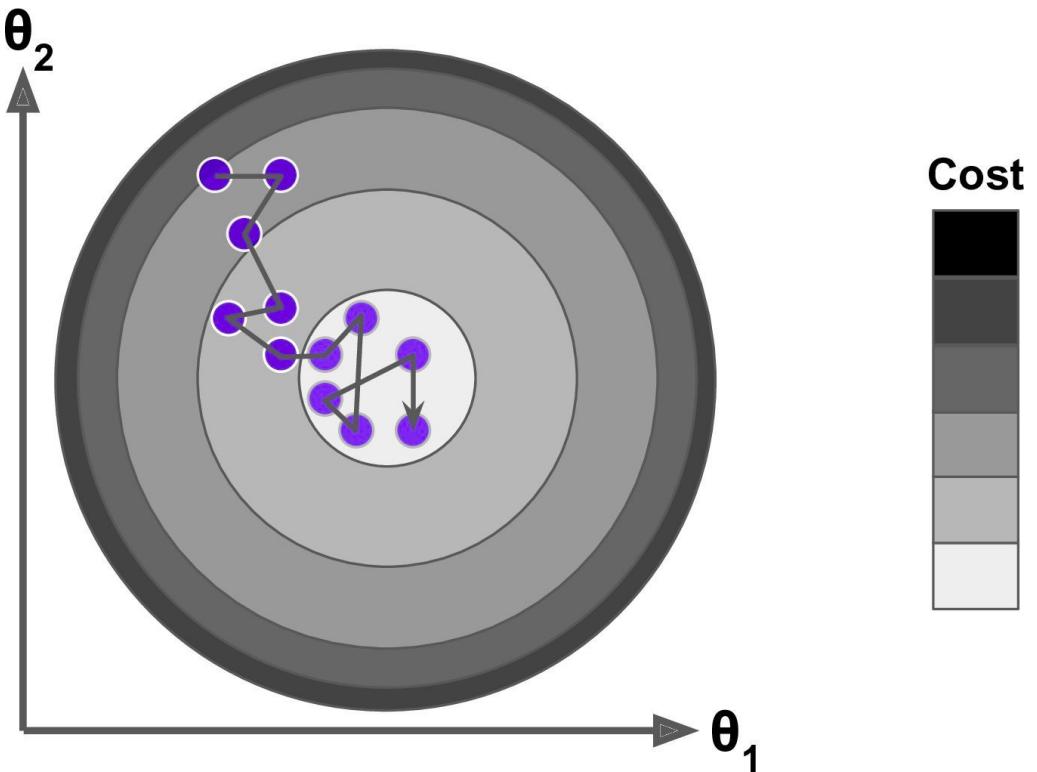
- Batch Gradient Descent
 - Slow when the training set is large
 - As it uses the whole training set to compute the gradient at every step
- Stochastic Gradient Descent
 - Just picks a random instance in the training set at every step and
 - Computes the gradients based only on that single instance



Stochastic Gradient Descent - SGD

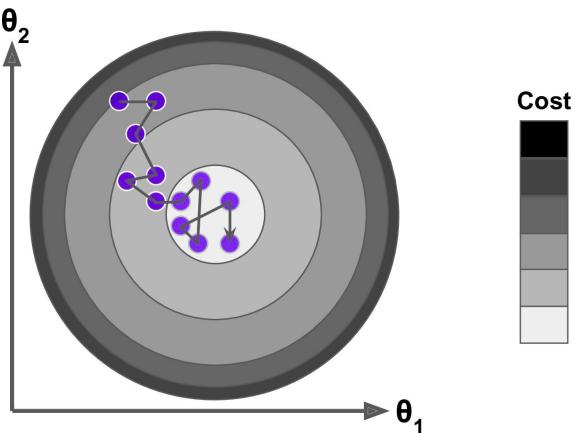
- This makes the algorithm much faster
 - As very little data gets manipulated at every iteration
- Also huge training sets can be trained
 - As only one instance needs to be in memory at each iteration

Stochastic Gradient Descent - SGD



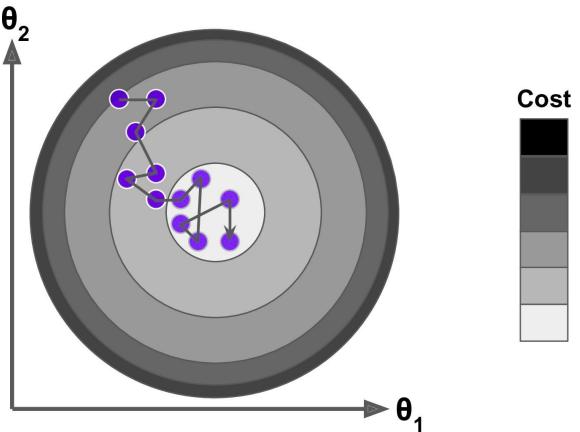
Stochastic Gradient Descent – SGD

- Instead of gently decreasing until it reaches the minimum
 - The cost function bounces up and down, decreasing only on average
 - Over time it ends up very close to the minimum
 - But once it gets there it continues to bounce around, never settling down



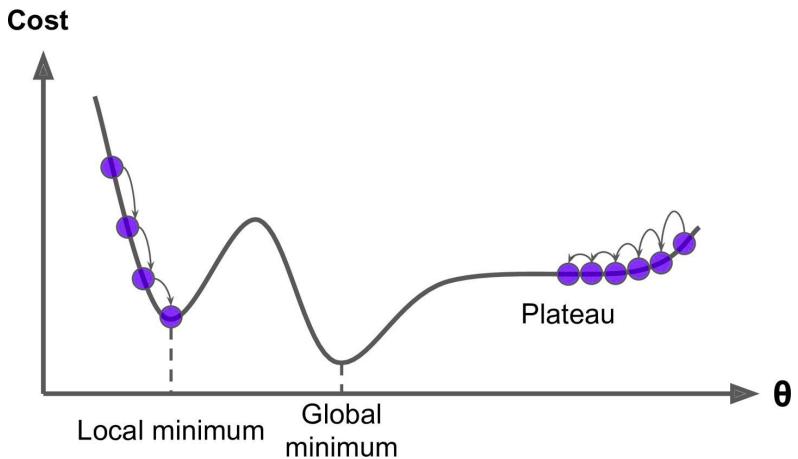
Stochastic Gradient Descent - SGD

- Once the algorithm stops
 - The final parameter values are good, but not optimal
 - As algorithm never settles down



Stochastic Gradient Descent - SGD

- When the cost function is very irregular then
 - SGD has a better chance of finding the global minimum than Batch Gradient Descent
 - As randomness may help algorithm to jump out of local minima





Stochastic Gradient Descent - SGD

- As seen earlier that randomness in SGD
 - Is good to escape from local minimum
 - But bad as algorithm can never settle at minimum



Stochastic Gradient Descent - SGD

- As seen earlier that randomness in SGD
 - Is good to escape from local minimum
 - But bad as algorithm can never settle at minimum

What is the solution?



Stochastic Gradient Descent – SGD

Simulated Annealing

- One solution is to gradually reduce the learning rate
- The steps start out large
 - This helps make quick progress and escape local minima
- Then steps get smaller and smaller
 - This helps the algorithm to settle at the global minimum
- This process is called **Simulated Annealing**
 - Because it resembles the process of annealing in metallurgy where molten metal is slowly cooled down



Stochastic Gradient Descent - SGD

Learning Schedule

- The function that determines the learning rate
 - At each iteration is called the **learning schedule**
- If the learning rate is reduced too quickly
 - We may get stuck in a local minimum Or
 - Even end up frozen halfway to the minimum
- If the learning rate is reduced too slowly
 - We may jump around the minimum for a long time and
 - End up with a suboptimal solution if we halt training too early



Stochastic Gradient Descent - SGD

Check the code of SGD in Notebook



Mini-batch Gradient Descent



Mini-batch Gradient Descent

- The last Gradient Descent algorithm we will look into
- Quite easy to understand if we know both
 - Batch and
 - Stochastic Gradient Descent



Mini-batch Gradient Descent

- At each step
 - Instead of computing the gradients based on the full training set (as in Batch GD) or
 - Based on just one instance (as in Stochastic GD)
 - Mini-batch GD computes the gradients on small random sets of instances called **mini-batches**



Mini-batch Gradient Descent

Advantages

- The main advantage of Mini-batch GD over Stochastic GD is that
 - We can get a performance boost
 - From hardware optimization of matrix operations
 - Especially when using GPUs



Mini-batch Gradient Descent

Advantages

- Mini-batch GD algorithm's progress is
 - Less erratic than SGD
 - Especially with fairly large mini-batches
 - As a result it goes more closer to minimum than SGD



Mini-batch Gradient Descent

Disadvantages

- It is harder for Mini-batch GD to escape from local minima



Mini-batch Gradient Descent

Check the code of Mini-batch Gradient Descent in
Notebook



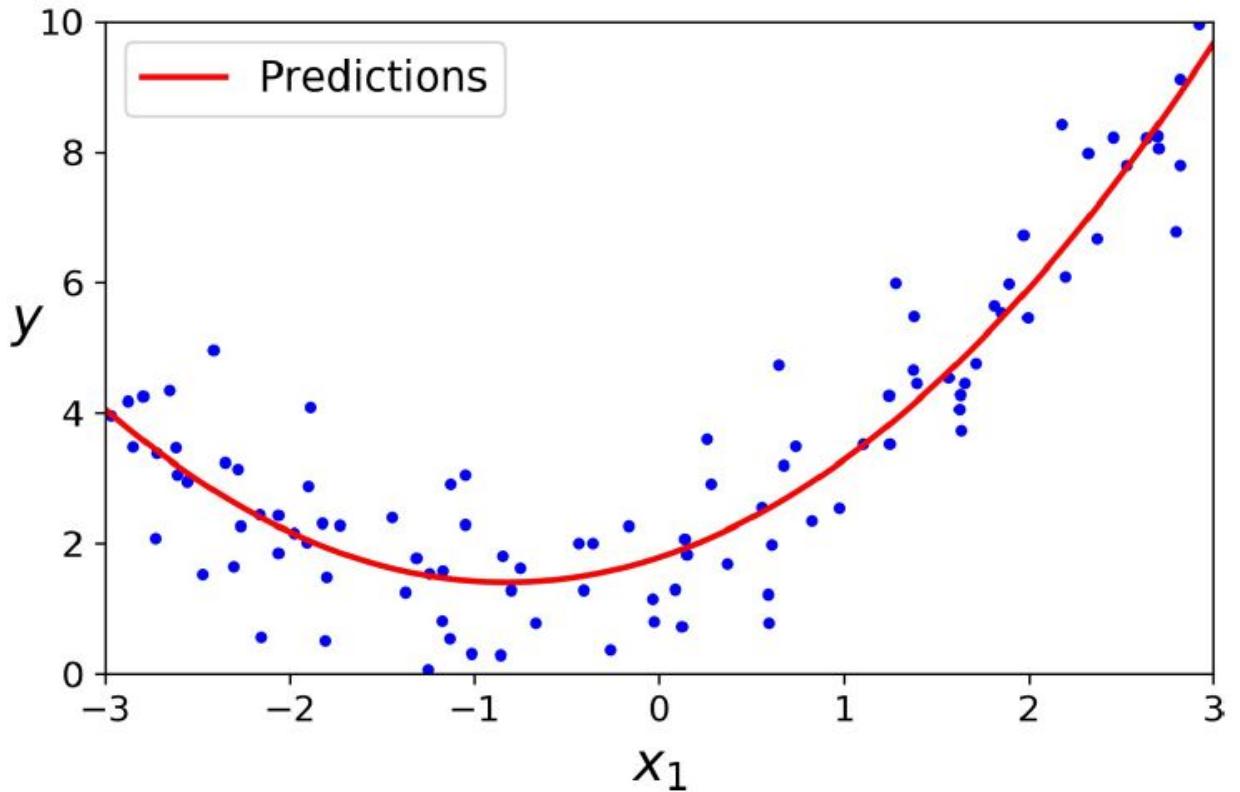
Comparison

Algorithm	Large m	Out-of-core support	Large n	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	LinearRegression
Batch GD	Slow	No	Fast	2	Yes	n/a
Stochastic GD	Fast	Yes	Fast	≥ 2	Yes	SGDRegressor
Mini-batch GD	Fast	Yes	Fast	≥ 2	Yes	n/a

m is the number of training instances
 n is the number of features



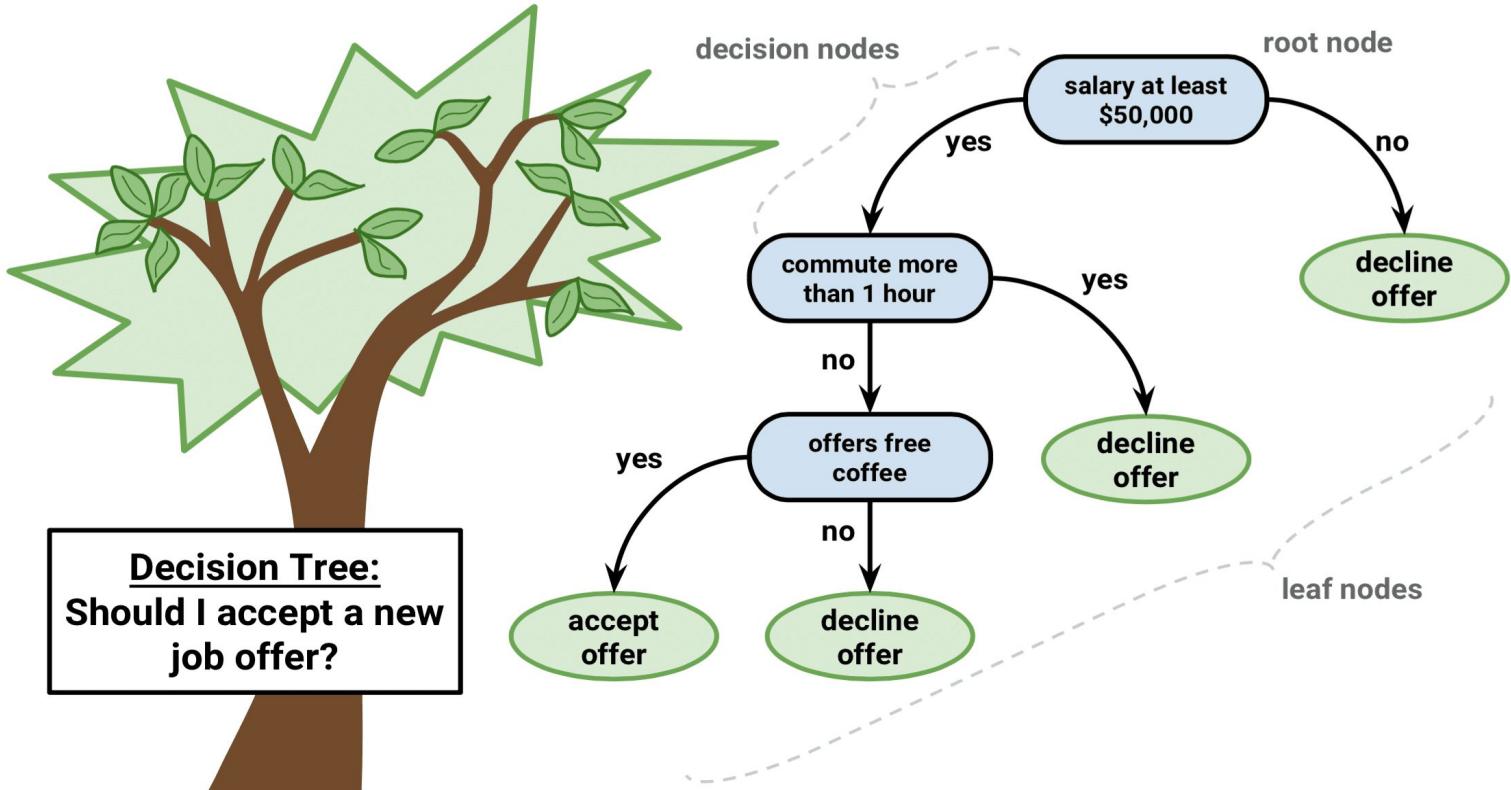
Polynomial Regression





Decision Trees

Decision Tree





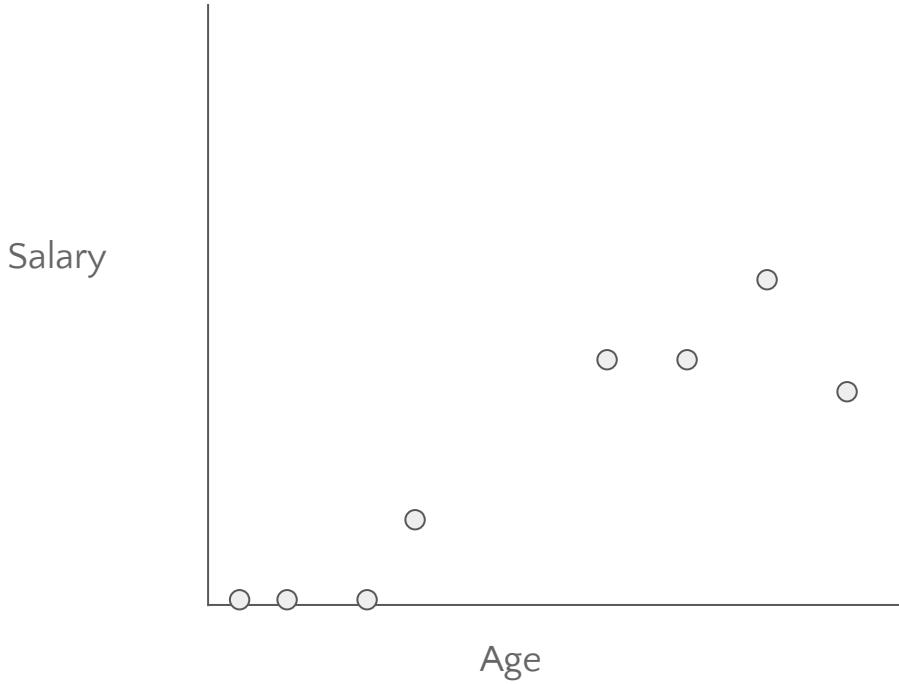
Decision Tree

Age	Salary
10	0
5	0
2	0
13	5
25	20
30	20
35	25
40	18
age	?



Decision Tree

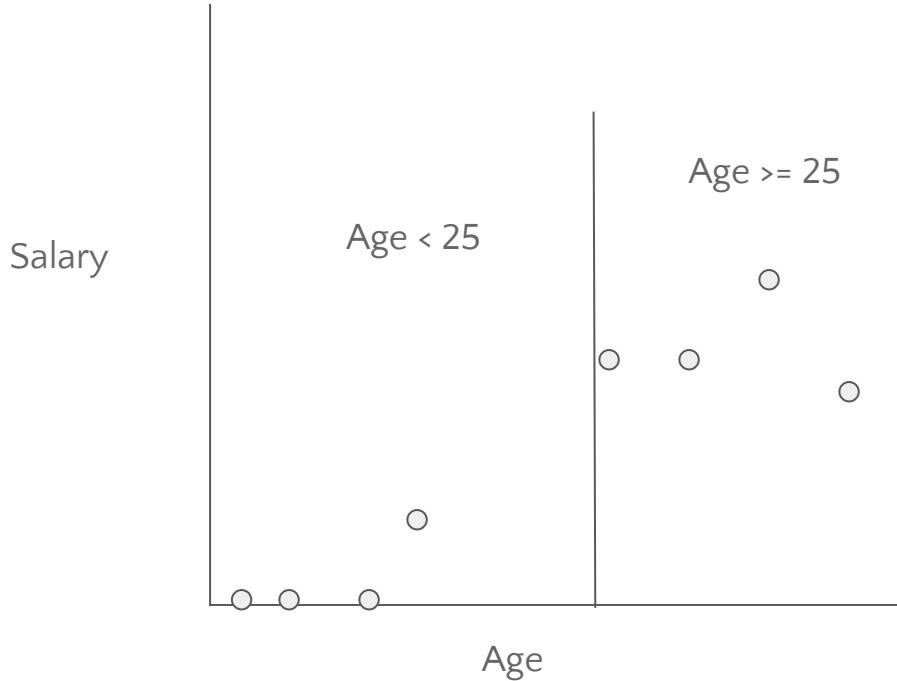
Age	Salary
10	0
5	0
2	0
13	5
25	20
30	20
35	25
40	18
age	?





Decision Tree

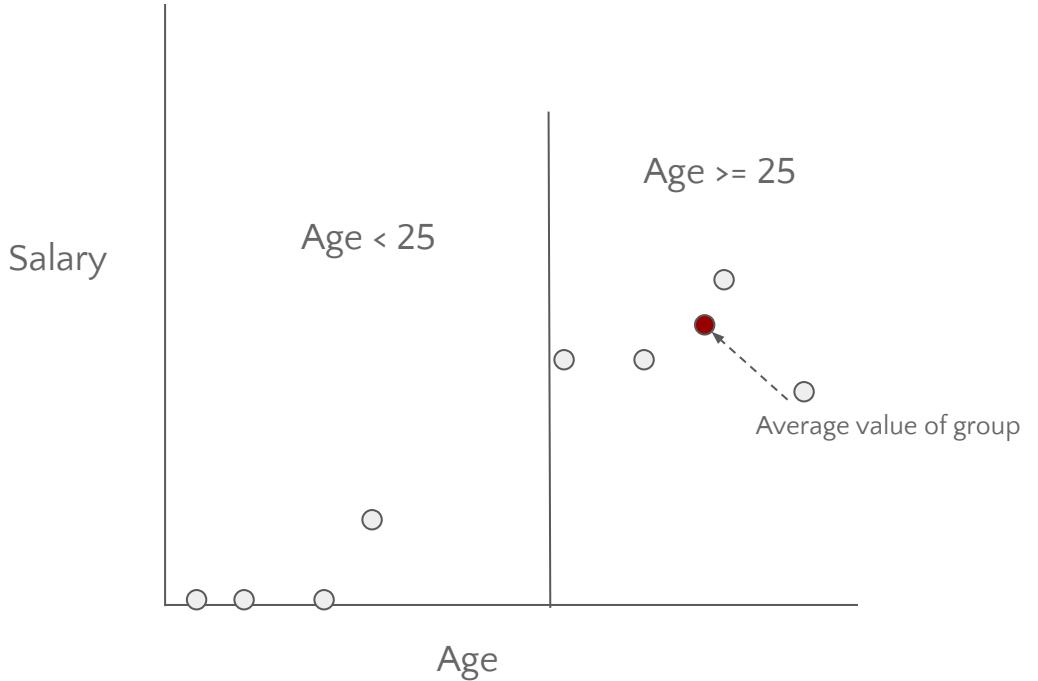
Age	Salary
10	0
5	0
2	0
13	5
25	20
30	20
35	25
40	18
age	?





Decision Tree

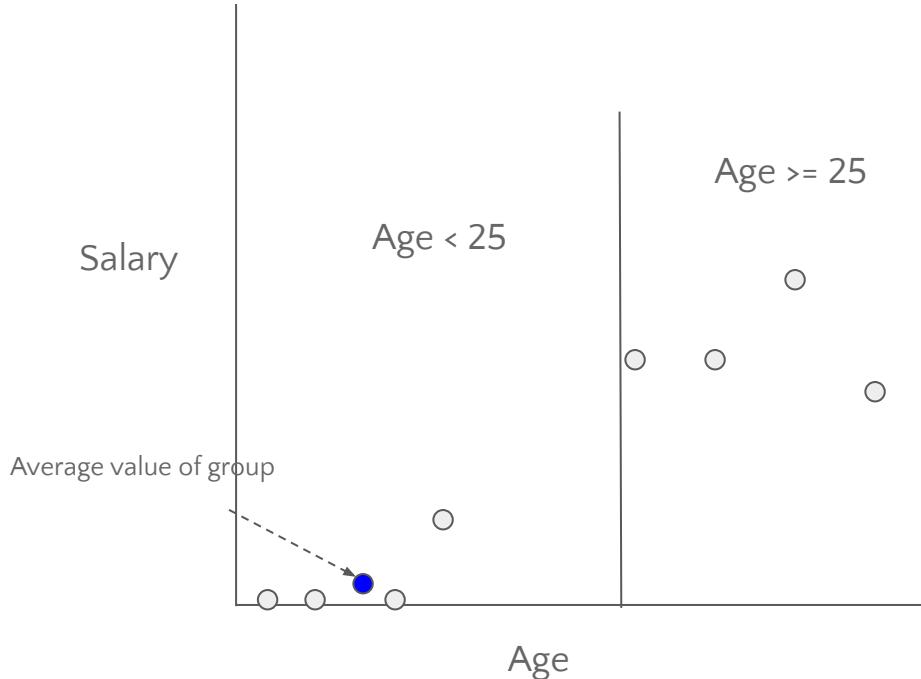
Age	Salary
10	0
5	0
2	0
13	5
25	20
30	20
35	25
40	18
age	?



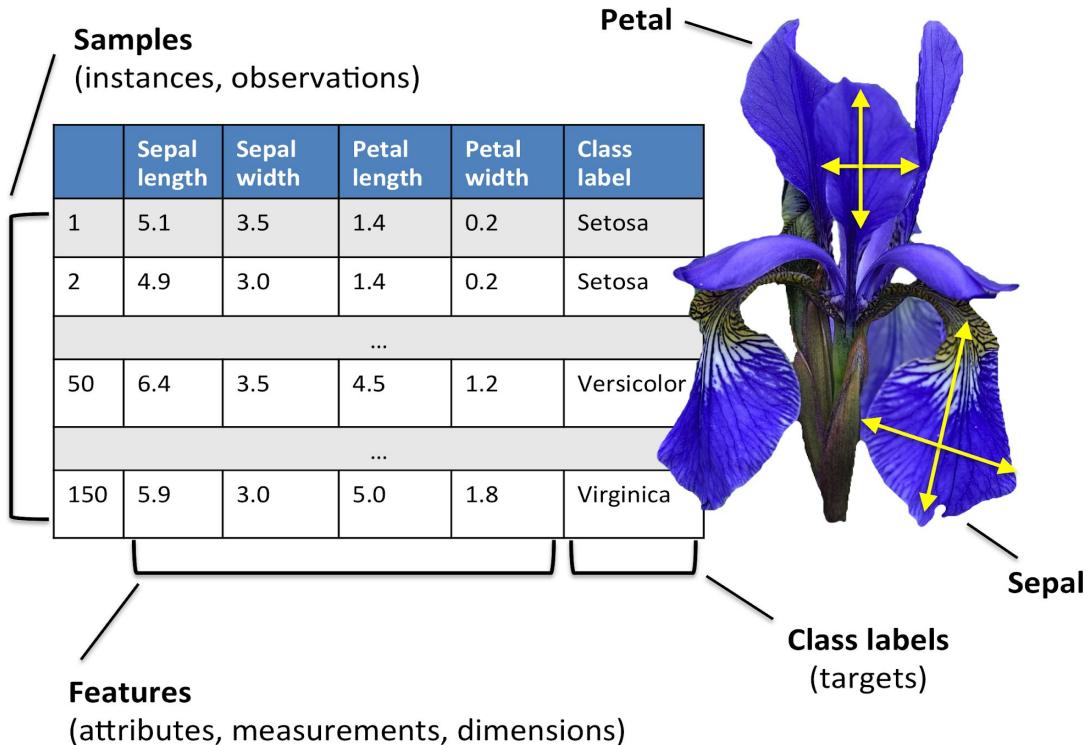


Decision Tree

Age	Salary
10	0
5	0
2	0
13	5
25	20
30	20
35	25
40	18
age	?

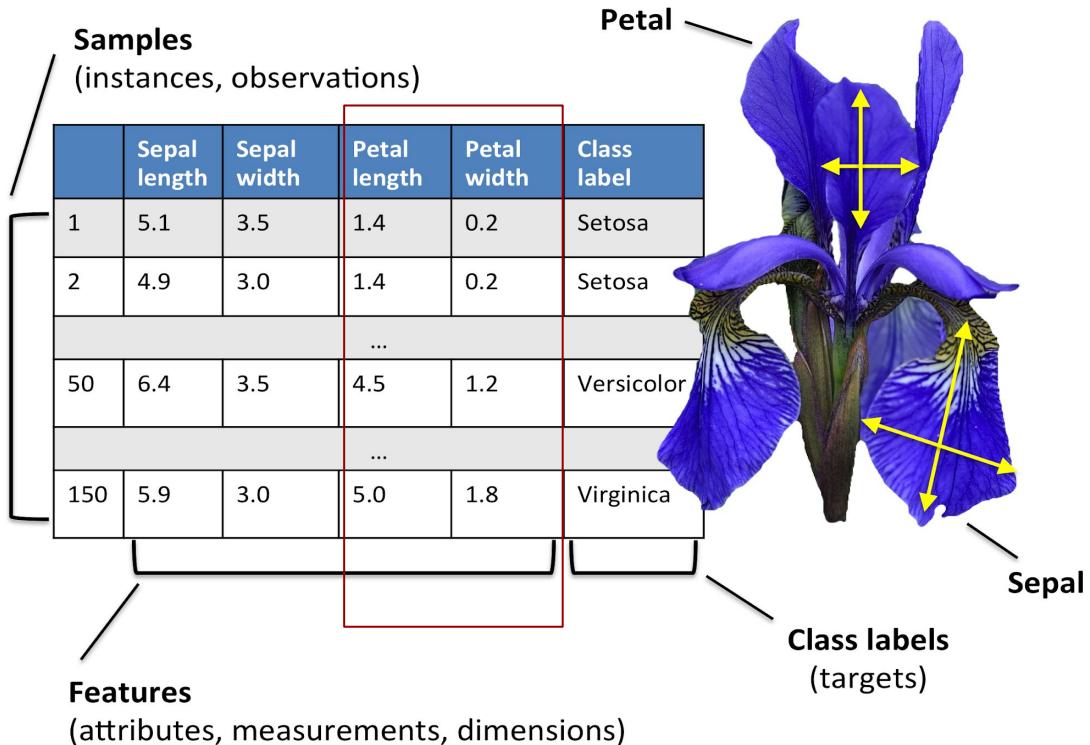


IRIS dataset



We'll only use two features i.e. petal length and petal width.

IRIS dataset



We'll only use two features i.e. petal length and petal width.

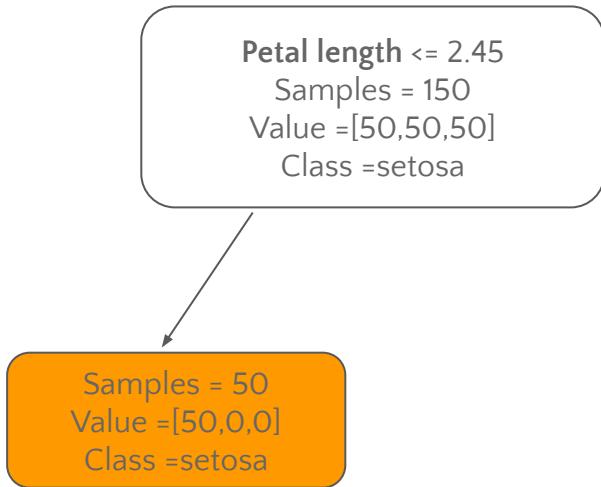


Decision Tree - IRIS dataset - Model

Petal length <= 2.45
Samples = 150
Value =[50,50,50]
Class =setosa

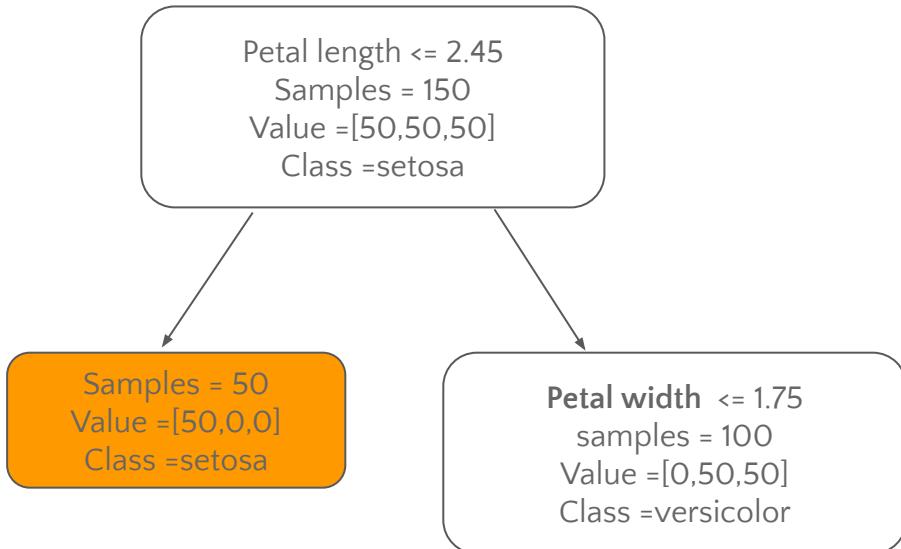


Decision Tree - IRIS dataset - Model



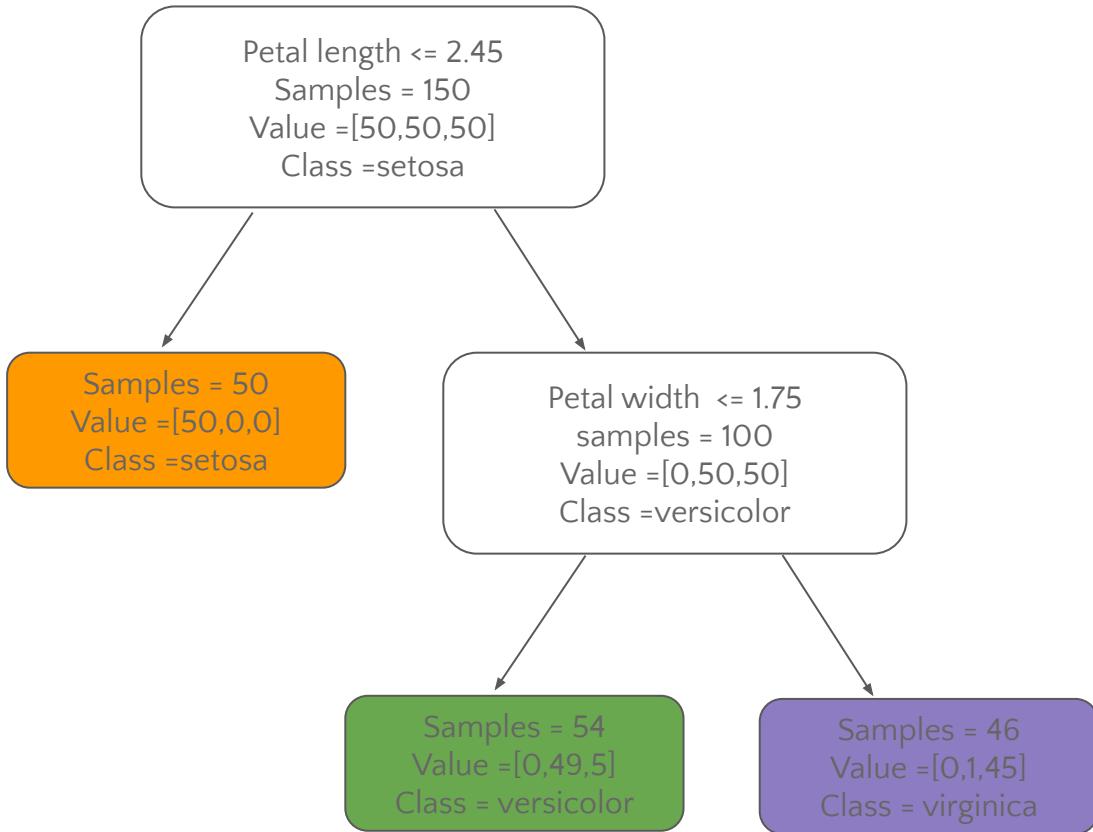


Decision Tree - IRIS dataset - Model



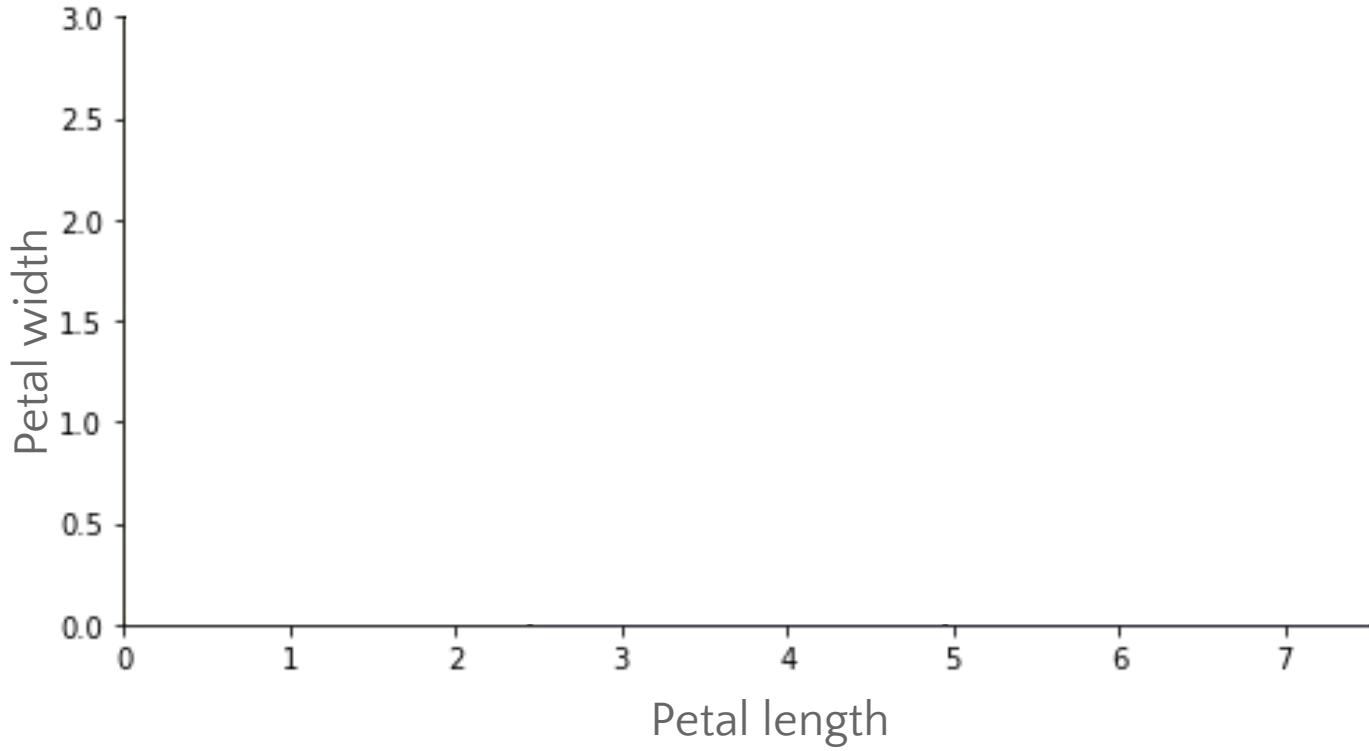


Decision Tree - IRIS dataset - Model



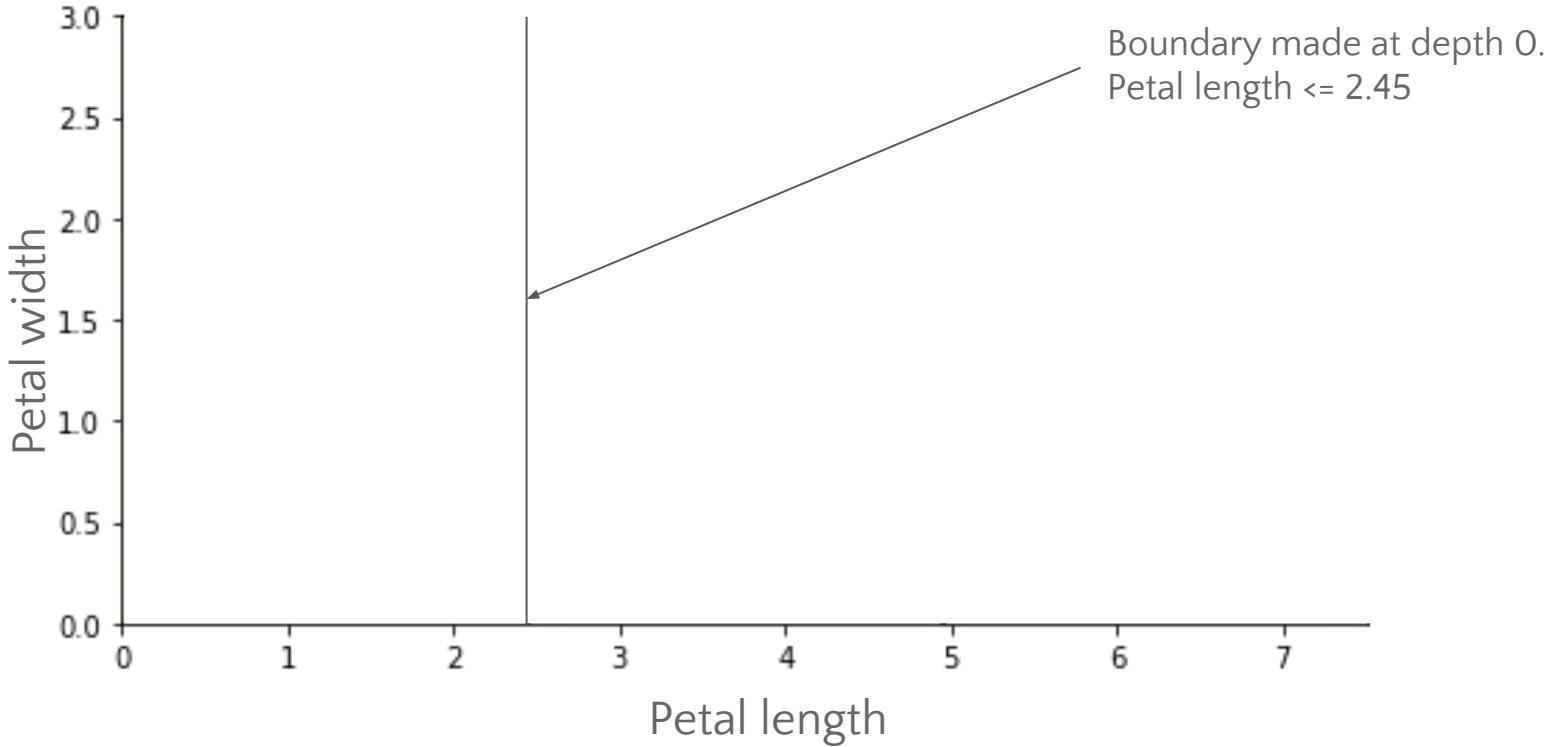


Decision Tree - Decision Boundaries



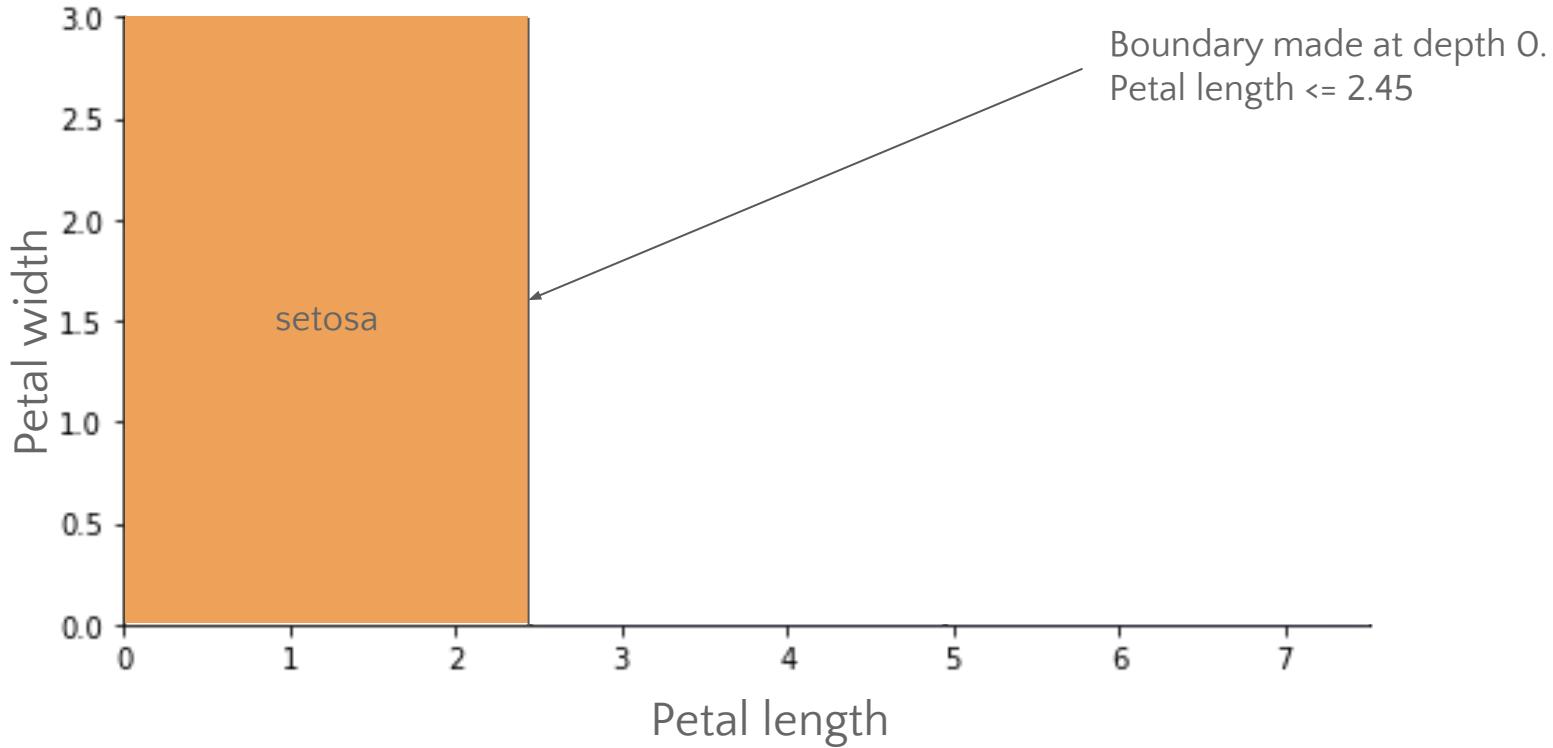


Decision Tree - Decision Boundaries



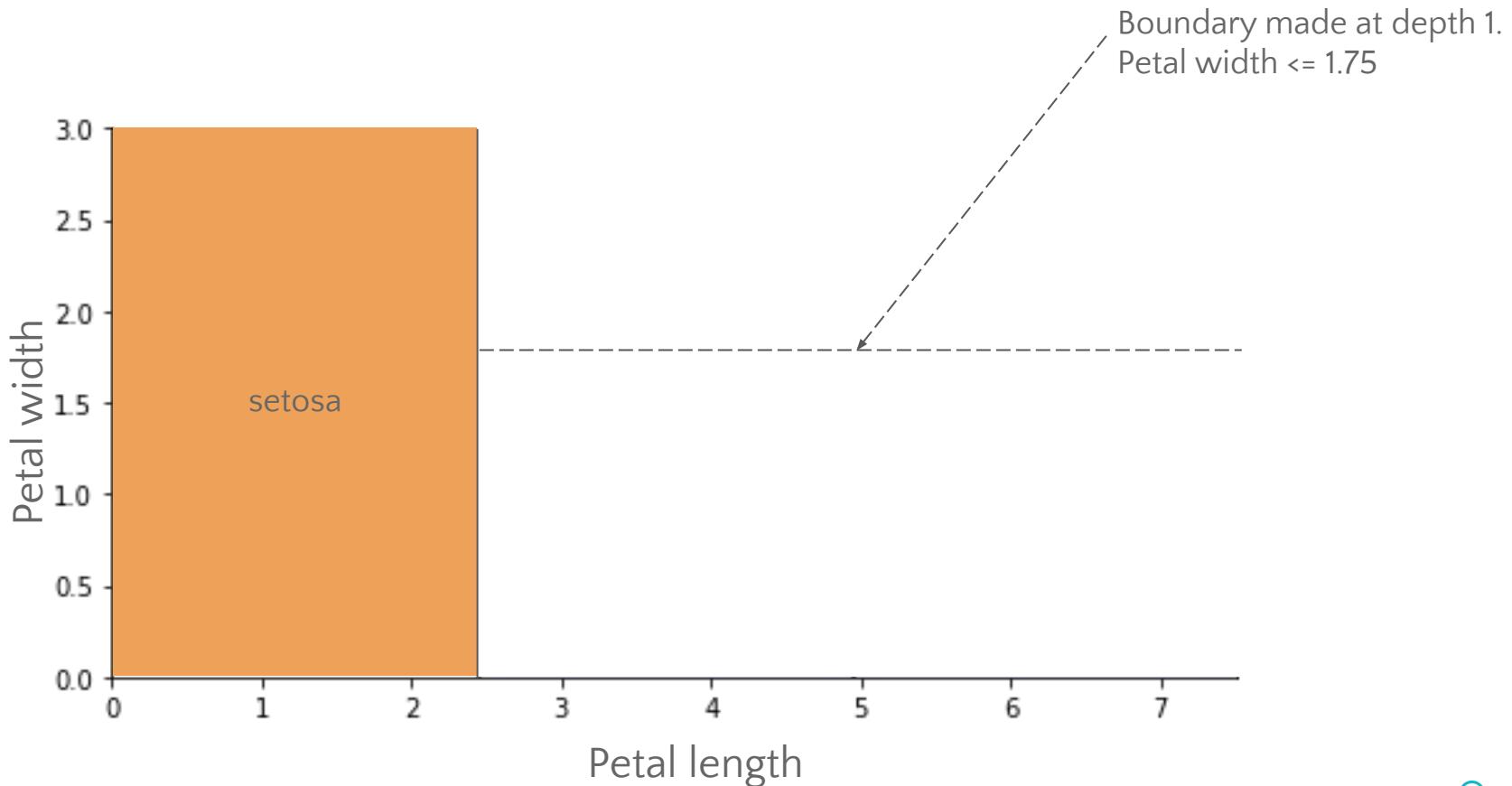


Decision Tree - Decision Boundaries



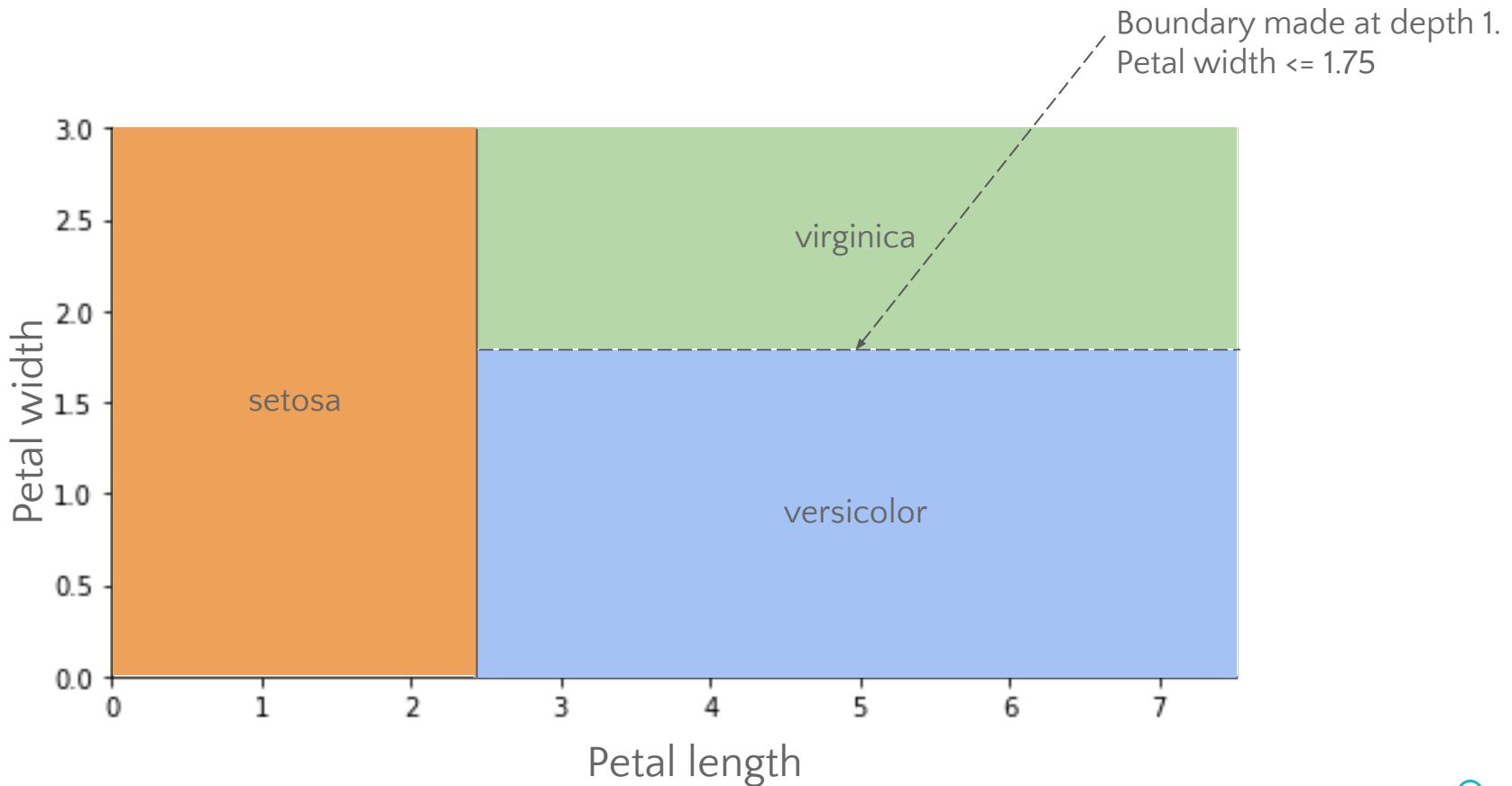


Decision Tree - Decision Boundaries



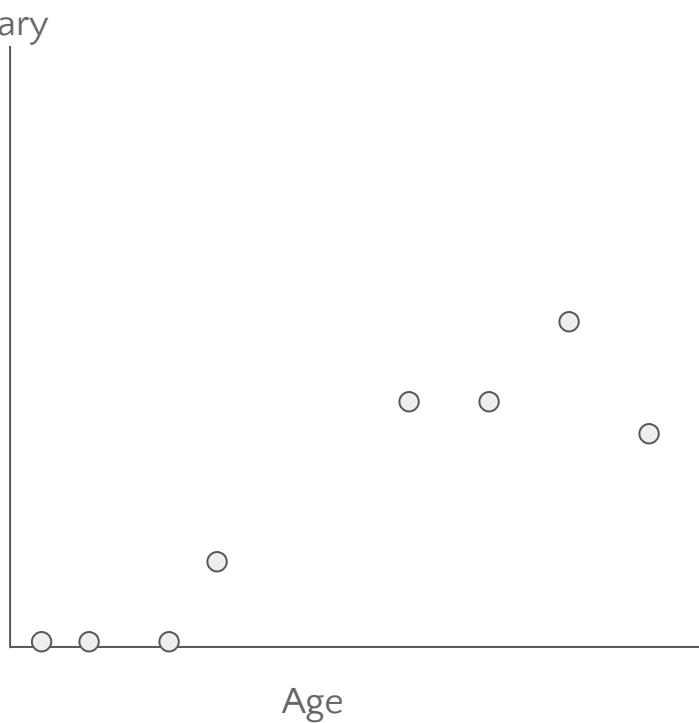


Decision Tree - Decision Boundaries





Decision Tree - how does it work?

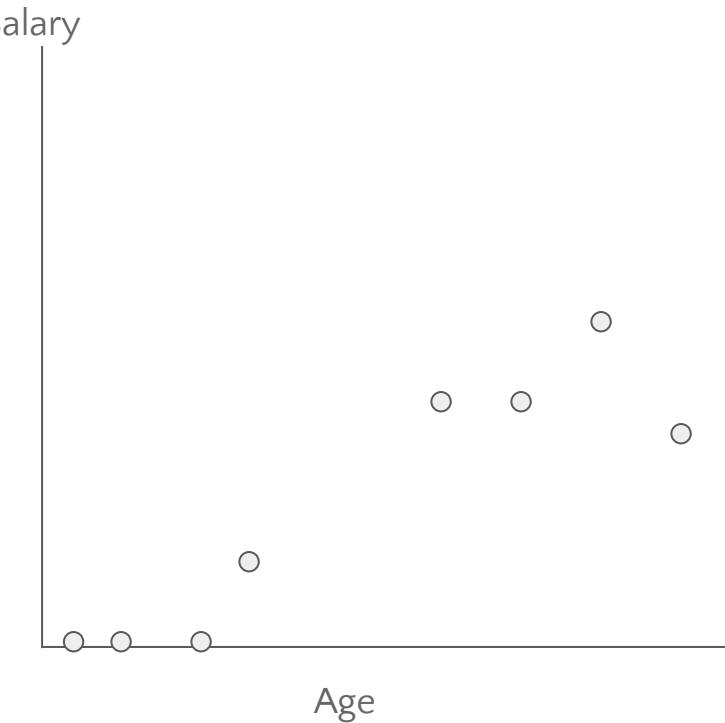




Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

1. Pick a feature

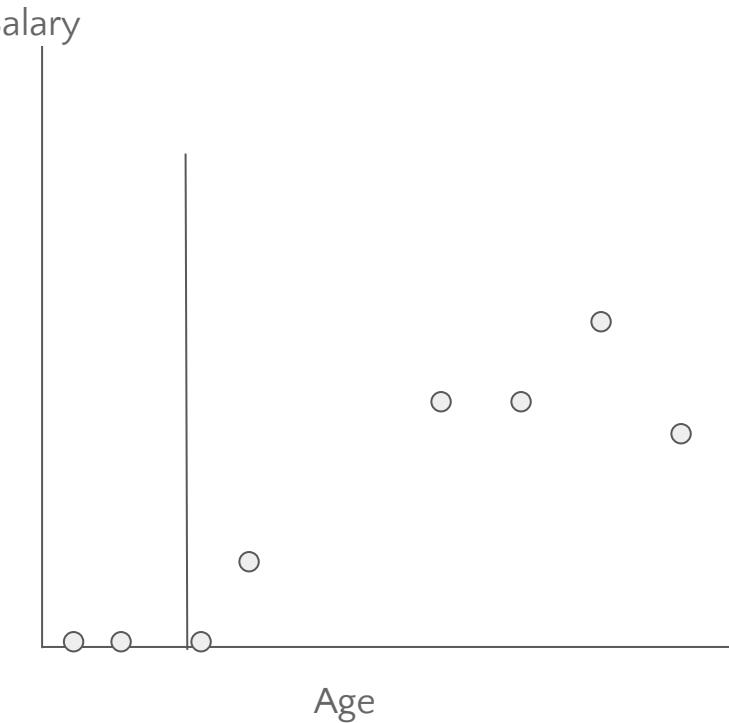




Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

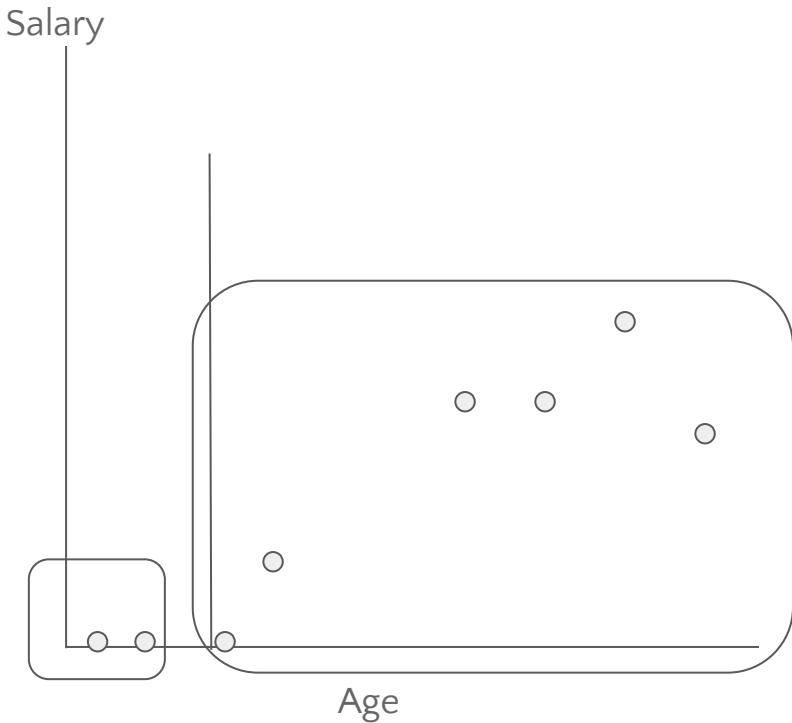
1. Pick a feature
2. Pick some boundary value



Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

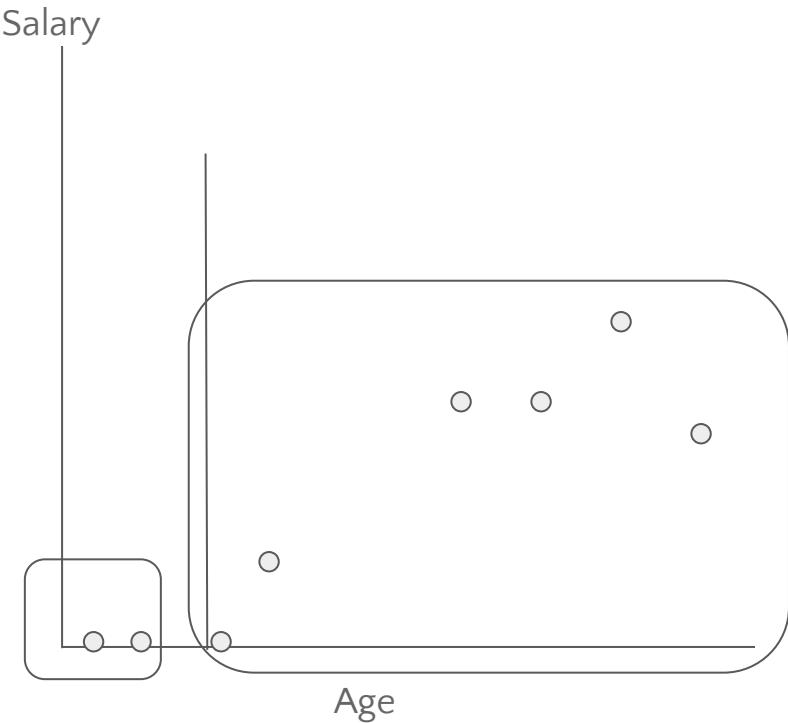
1. Pick a feature
2. Pick some boundary value
3. Find the similarity in instances on same side



Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

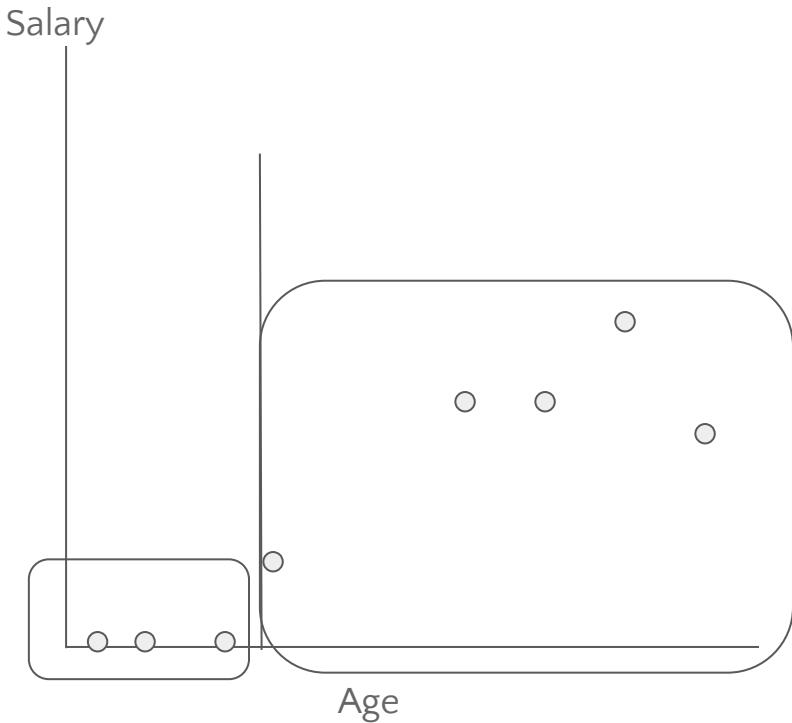
1. Pick a feature
2. Pick some boundary value
3. Find the similarity in instances on same side
4. If the similarity is better, keep it.
5. Go to step 1



Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

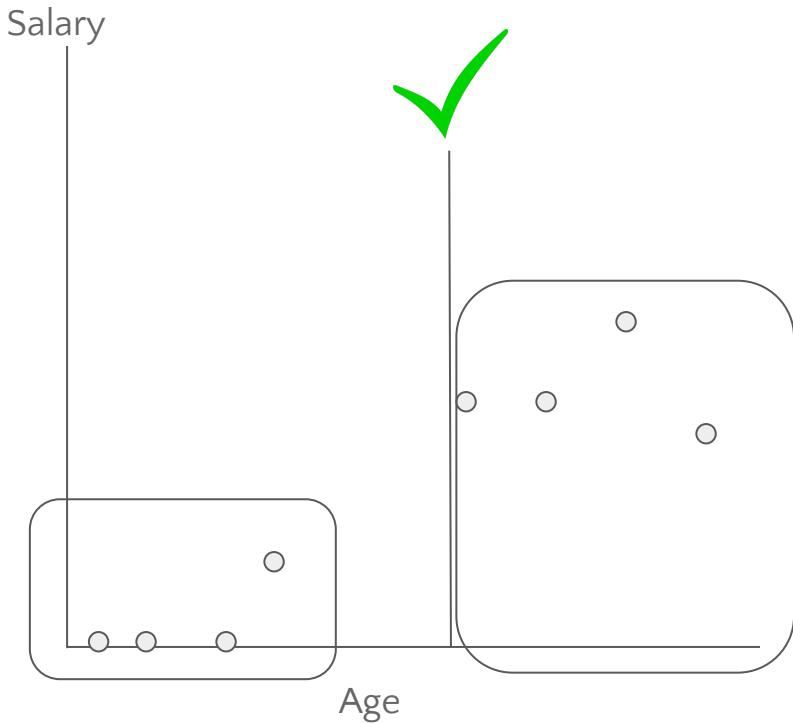
1. Pick a feature
2. Pick some boundary value
3. Find the similarity in instances on same side
4. If the similarity is better, keep it.
5. Go to step 1

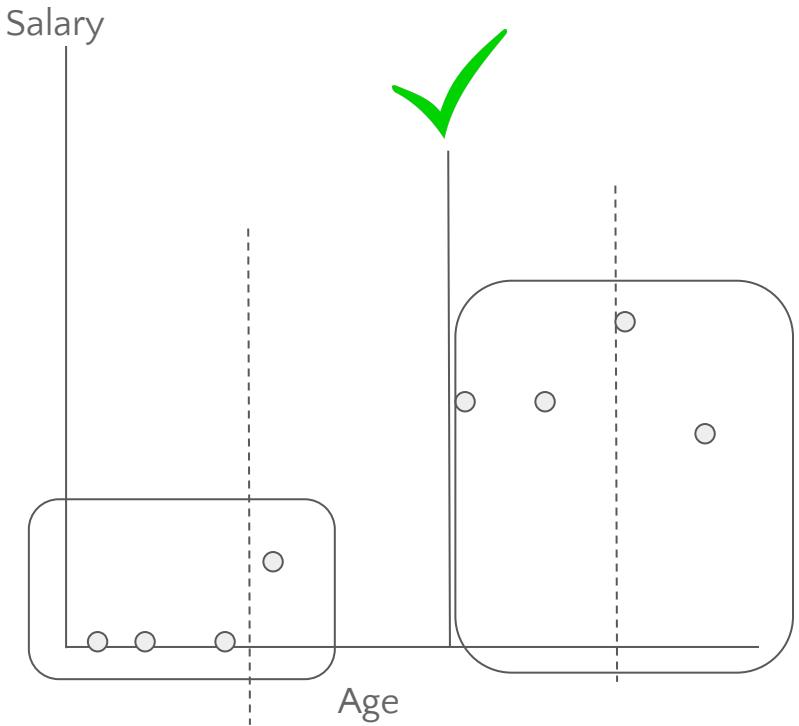
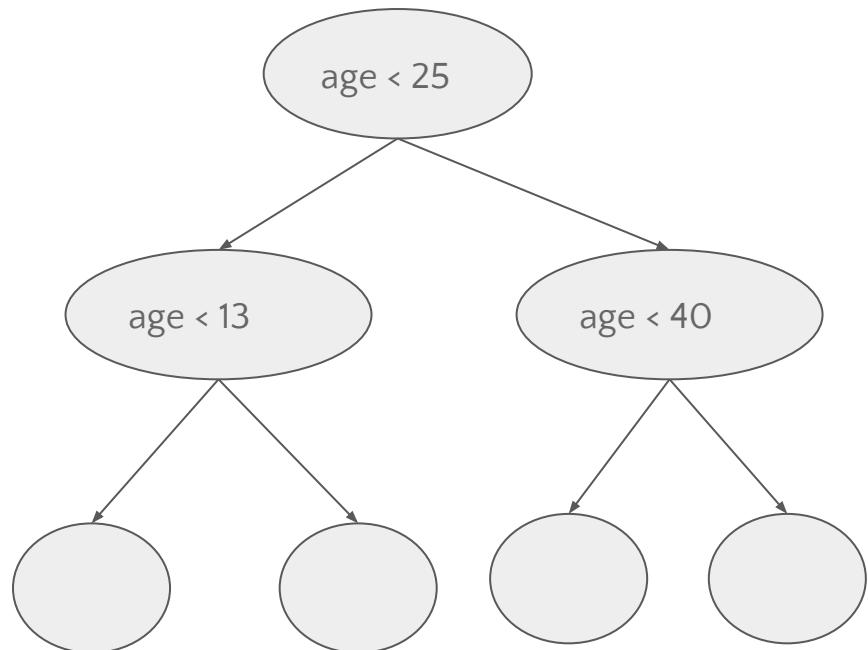


Decision Tree - how does it work?

It tries to find a boundary that separates similar instances. We go over all the features.

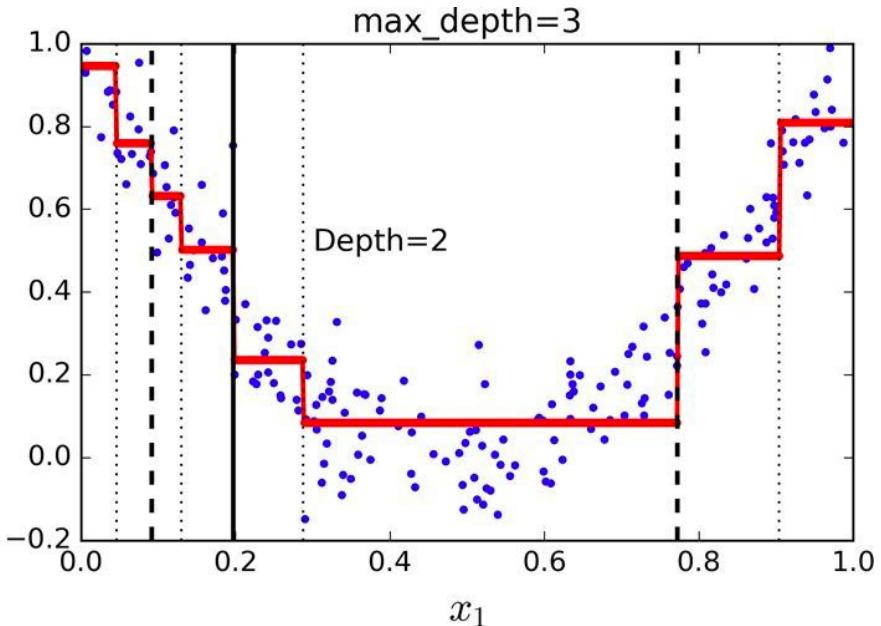
1. Pick a feature
2. Pick some boundary value
3. Find the similarity in instances on same side
4. If the similarity is better, keep it.
5. Go to step 1





Decision Tree - Complex Models?

Decision Trees can come up with really complex models.





Decision Tree

Check the code of Decision Trees in Notebook



Decision Tree – Final Word

1. Easy to Visualize



Decision Tree – Final Word

1. Easy to Visualize
2. Easy to Interpret & explain



Decision Tree – Final Word

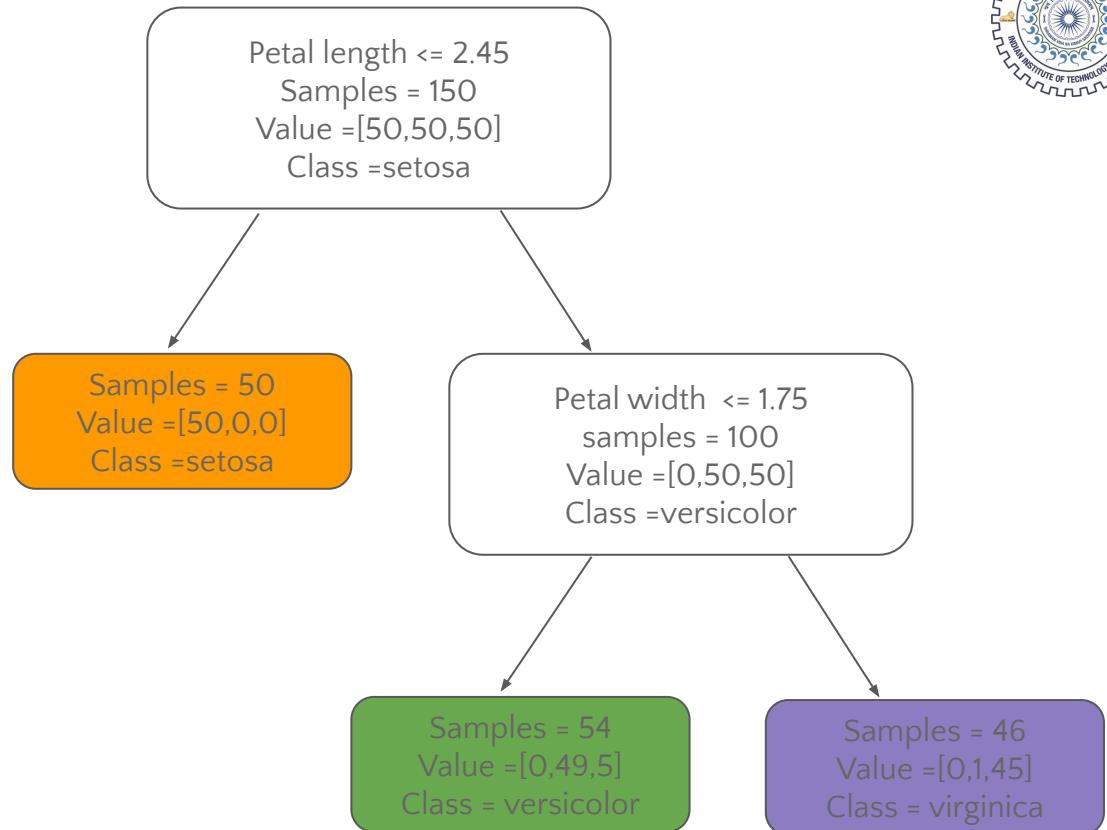
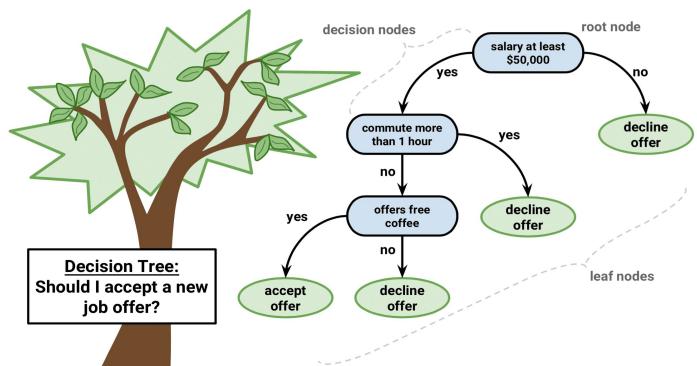
1. Easy to Visualize
2. Easy to Interpret & explain
3. If not designed well, ends up overfitting

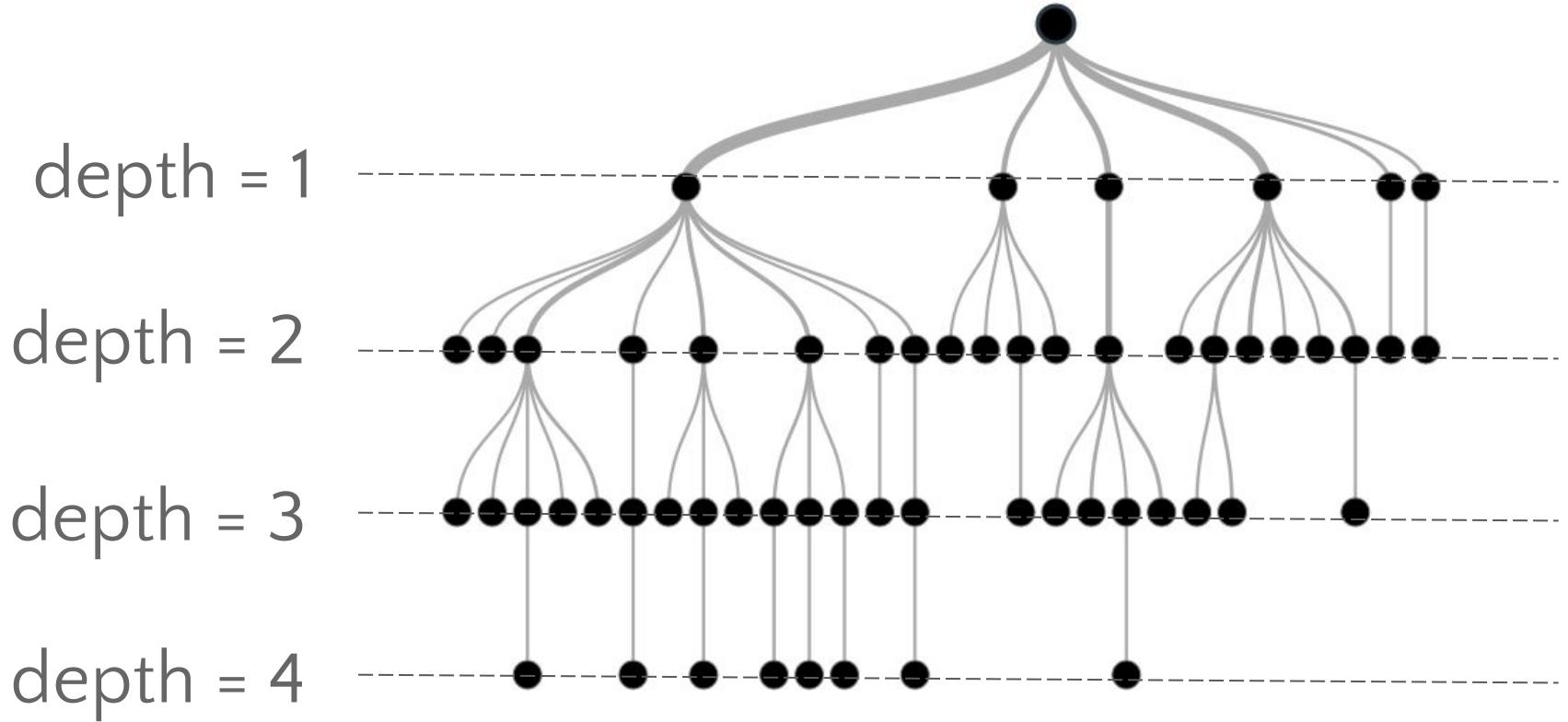


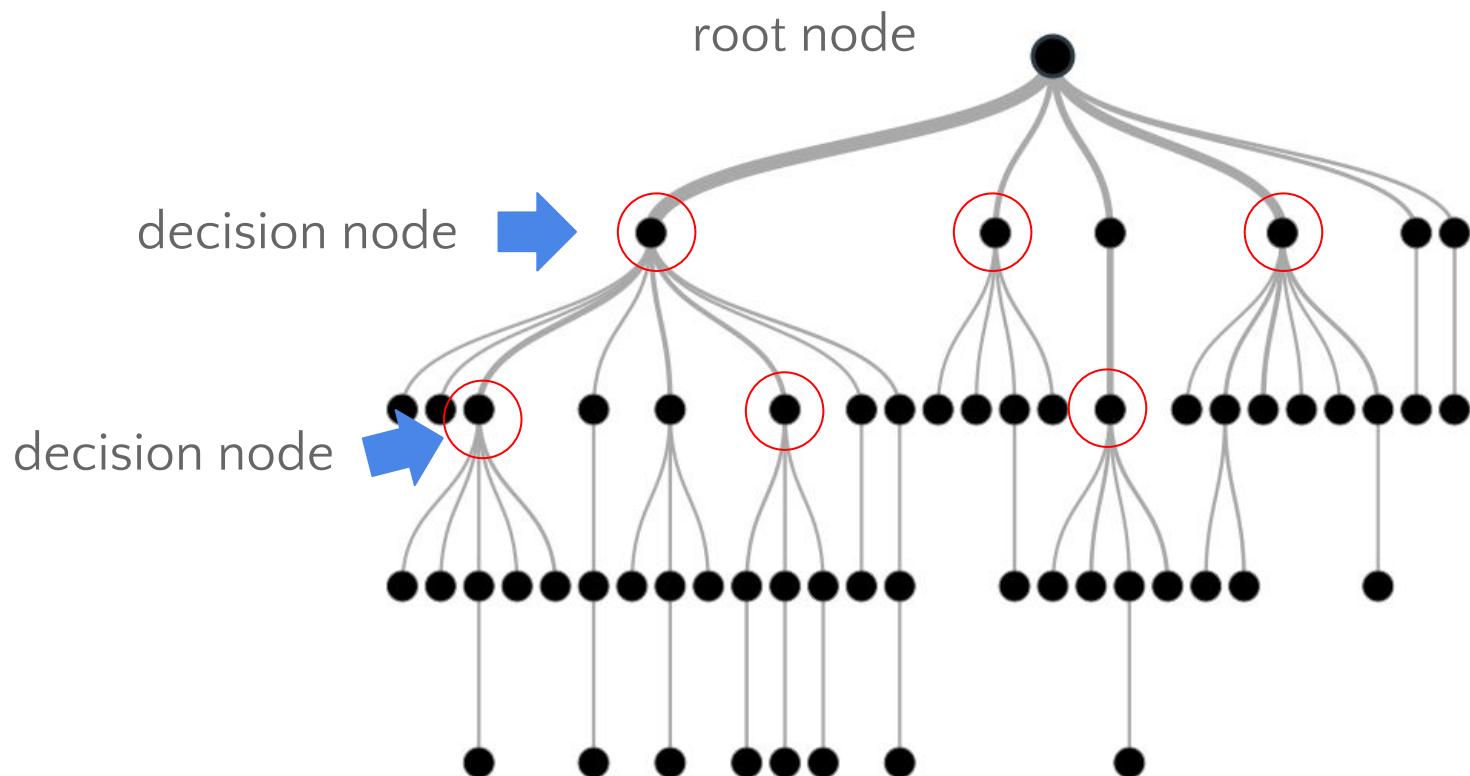
Regularization

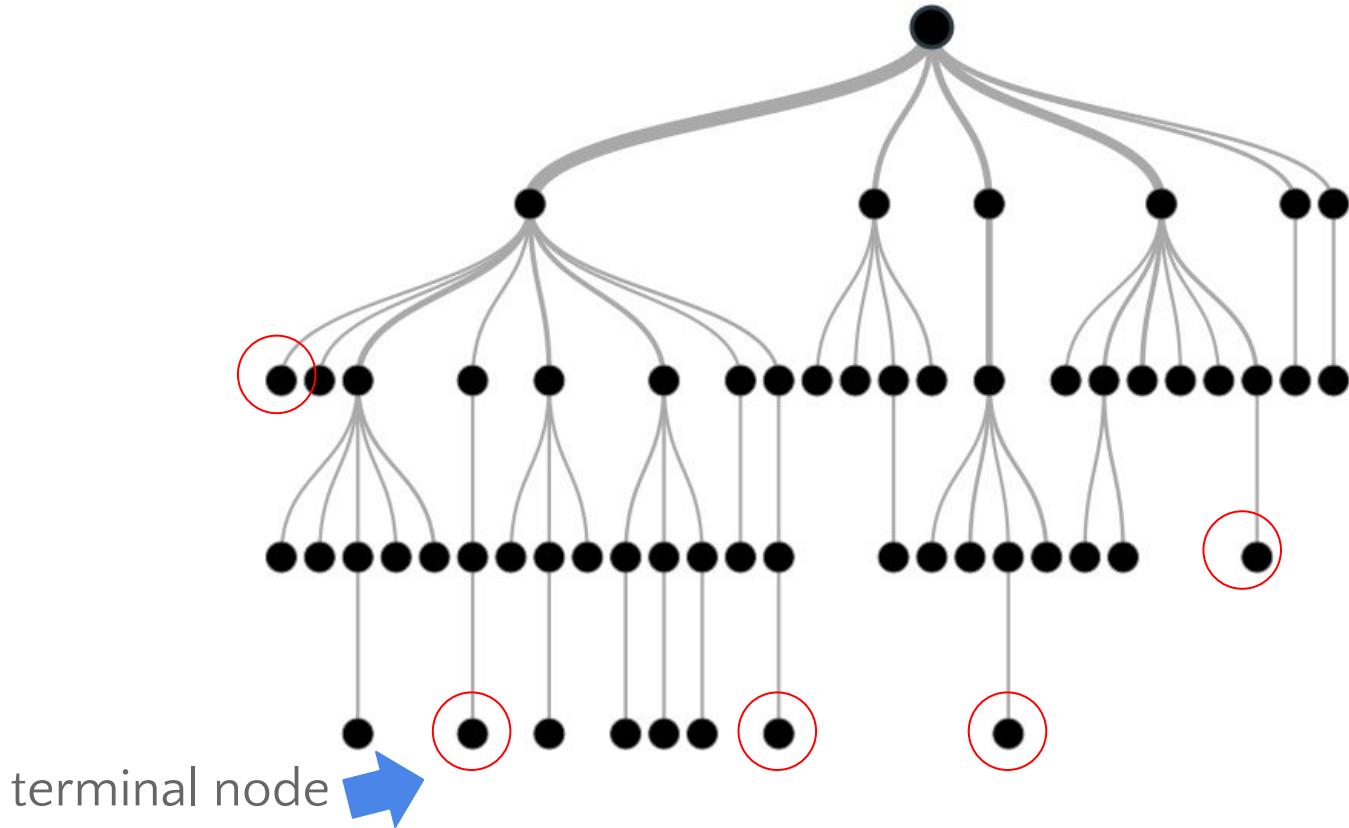


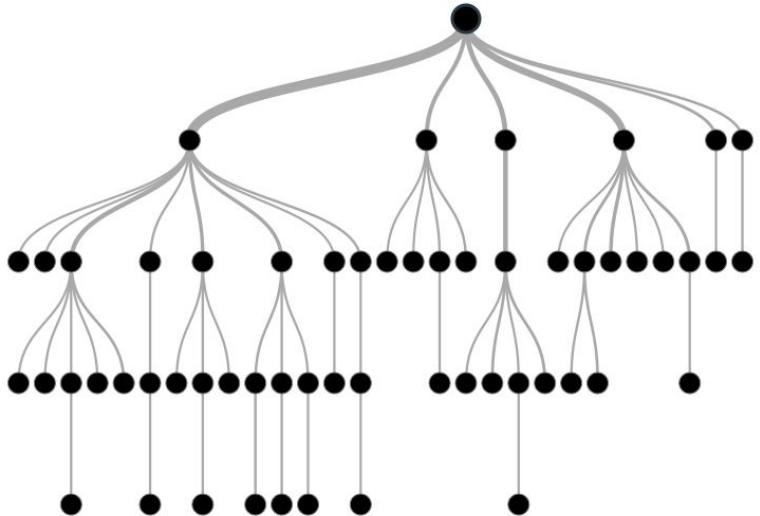
Avoiding overfitting and underfitting

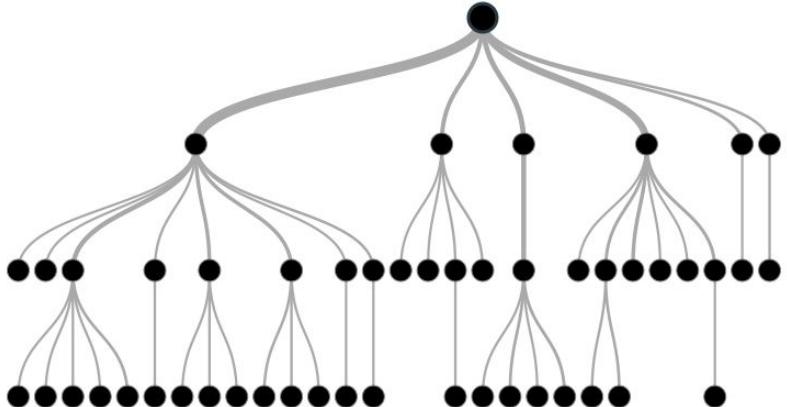
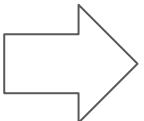
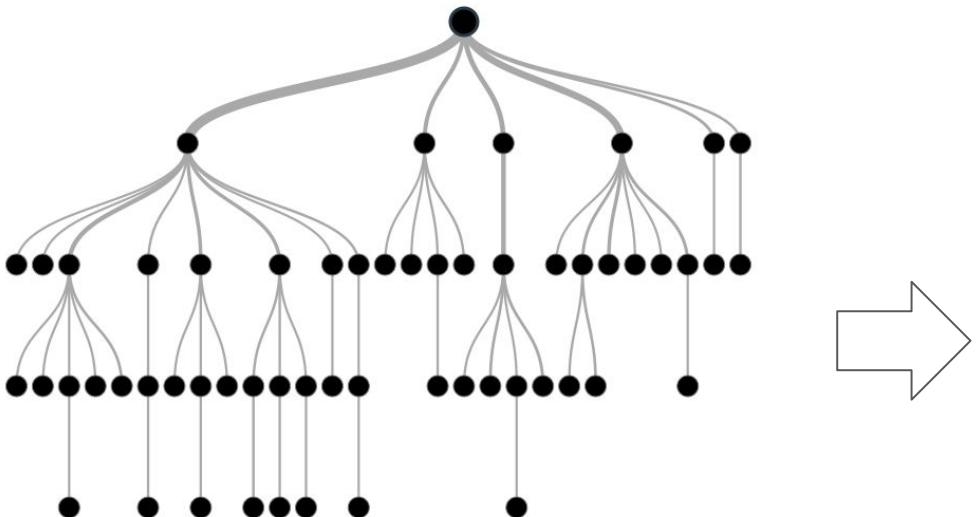


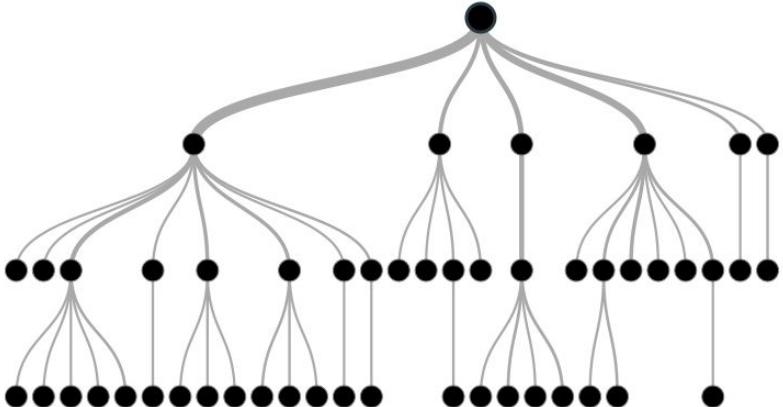
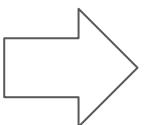
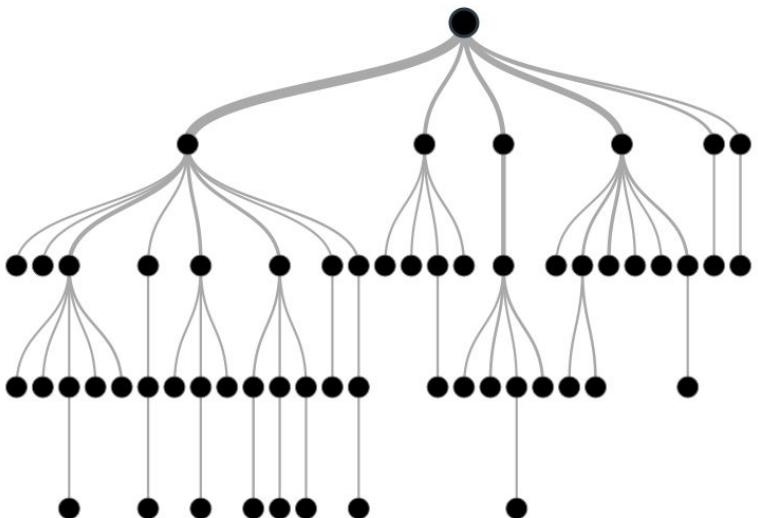




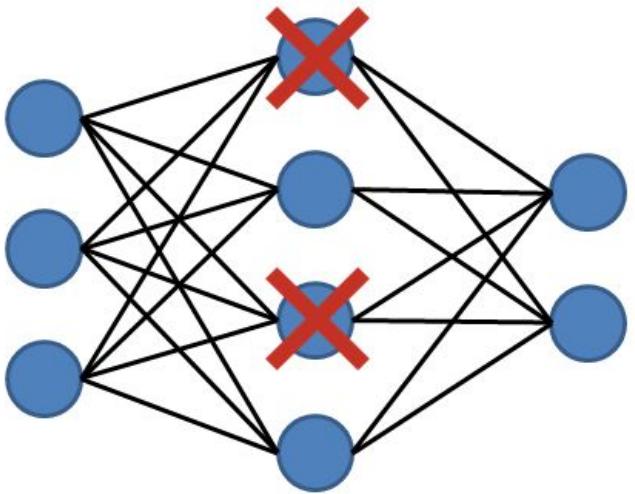








pruning



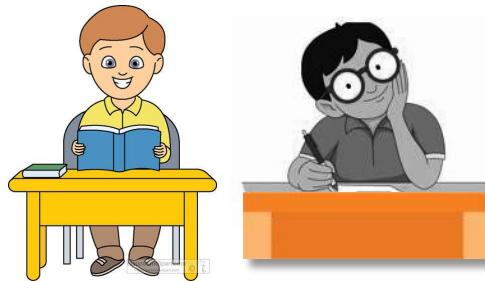
Drop-outs



student 1 student 2



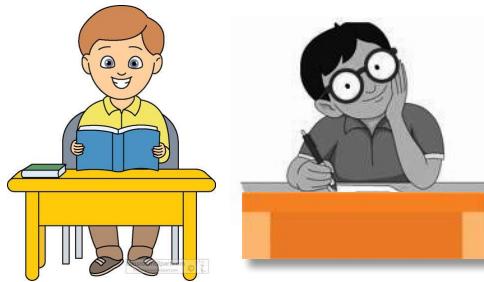
other students



student 1 student 2



other students



student 1 student 2

Q1



scenario 1

two students always
volunteer

student 1 student 2

Q1



Q2



scenario 1

two students always
volunteer

student 1 student 2

Q1



Q2



Q3



scenario 1

two students always
volunteer

student 1 student 2

Q1



Q2



Q3



Q4



scenario 1

two students always
volunteer

student 1 student 2

Q1



Q2



Q3



Q4



Q5



scenario 1

two students always
volunteer



scenario 2

the first two students
are avoided
sometimes

student 1 student 2

Q1



scenario 2

the first two students
are avoided
sometimes

student 1 student 2

Q1



Q2



scenario 2

the first two students
are avoided
sometimes

student 1 student 2

Q1



Q2



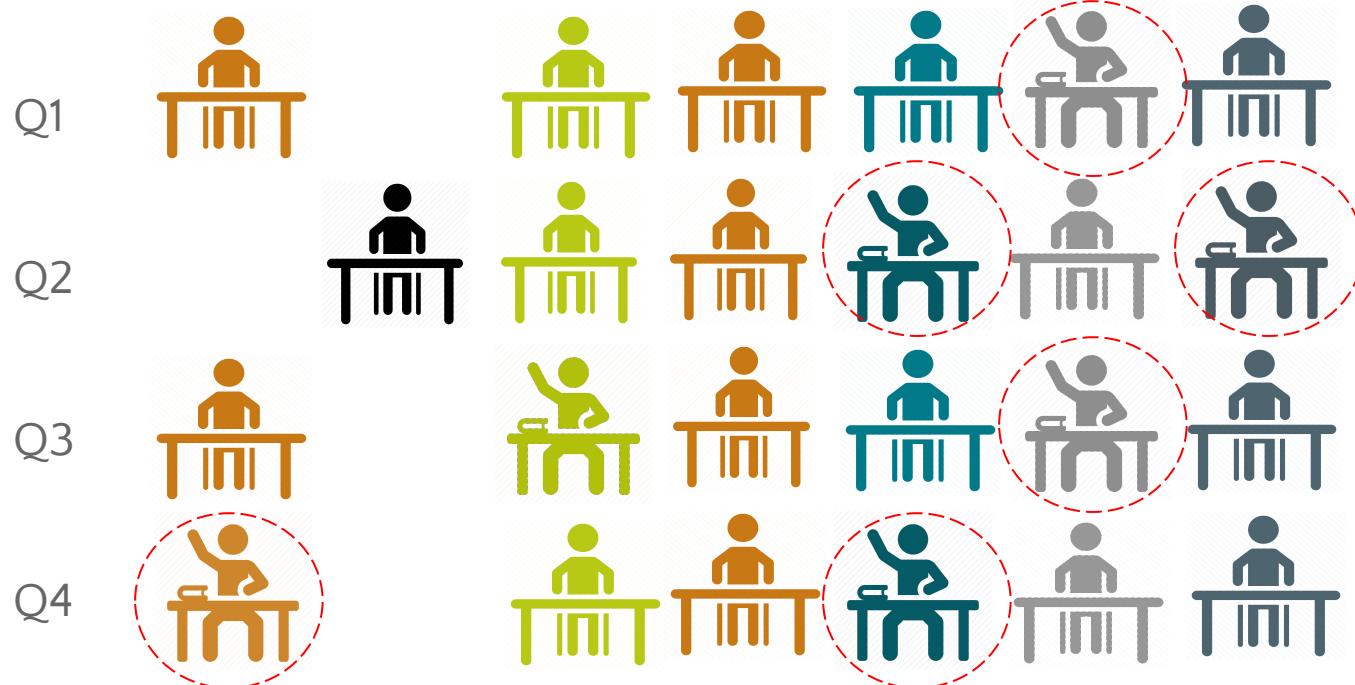
Q3



scenario 2

the first two students
are avoided
sometimes

student 1 student 2



scenario 2

the first two students
are avoided
sometimes

student 1 student 2



scenario 2

the first two students
are avoided
sometimes



Regularized Linear Models



Regularized Linear Models

- As we saw earlier, a good way to reduce overfitting is to
 - Regularize the model - Constraint it
- It is hard for a model to overfit data if it has less degree of freedom
- Example
 - To regularize a polynomial model, reduce the number of polynomial degrees



Regularized Linear Models

- For a linear model
 - Regularization is achieved by
 - Constraining the weights of the model



Regularized Linear Models

Let's understand L1 and L2 norms first



Regularized Linear Models

Different Performance Measures

- We've already looked into RMSE (Root Mean Square Error) as a performance measure
- RMSE is generally the preferred performance measure for regression tasks
- But in some cases we may want to use another performance measure
 - Like MAE – **Mean Absolute Error**
 - Also called the **Average Absolute Deviation**
- MAE is particularly useful when there are many outliers



Regularized Linear Models

$$\text{MAE}(\mathbf{X}, h) = \frac{1}{m} \sum_{i=1}^m |h(\mathbf{x}^{(i)}) - y^{(i)}|$$

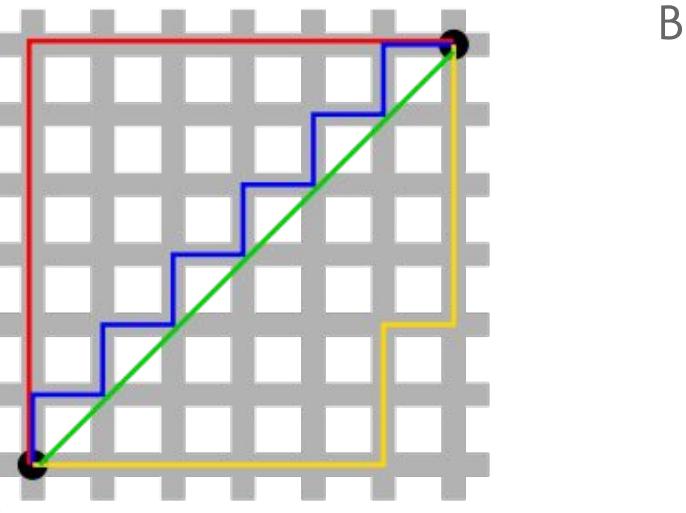
Mean Absolute Error



Regularized Linear Models

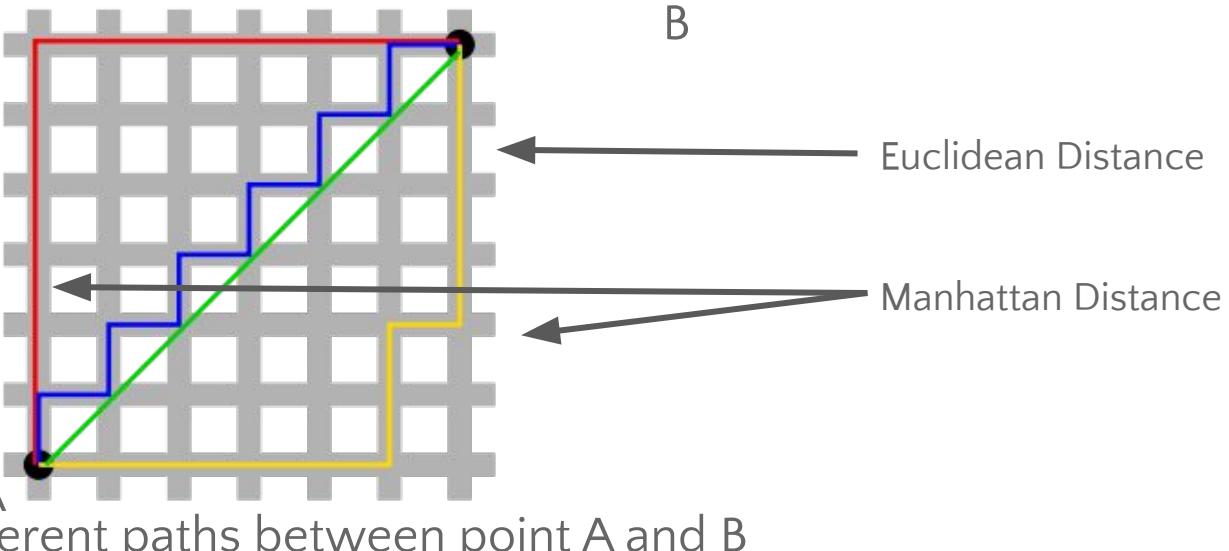
- Both RMSE and MAE are ways to measure distances
 - Between two vectors
 - Predictor values vector and target values vector
- There are various distance measures or norm to calculate distance between two vectors
 - Manhattan Norm
 - Euclidean Norm

Regularized Linear Models



Different paths between point A and B

Regularized Linear Models



- Red, Yellow and Blue paths (If we can only travel along orthogonal city blocks) – Manhattan Distance
- Green path – Euclidean Distance



Regularized Linear Models

L1 norm

- Computing the sum of absolutes (MAE) corresponds to $L1(\ell_1)$ norm
- Denoted as
 - $\| \cdot \|_1$
- Also called as Manhattan norm



Regularized Linear Models

L2 norm

- Computing the root of a sum of squares(RMSE) corresponds to L2(ℓ_2) norm
- Denoted as
 - $\| \cdot \|_2$ or just $\| \cdot \|$
- Also called as Euclidean norm



Regularized Linear Models

L1 vs L2 Regularization

- L1 can yield sparse models while L2 doesn't
- Sparse models help in feature selection
 - Which features are important and
 - Which features are redundant



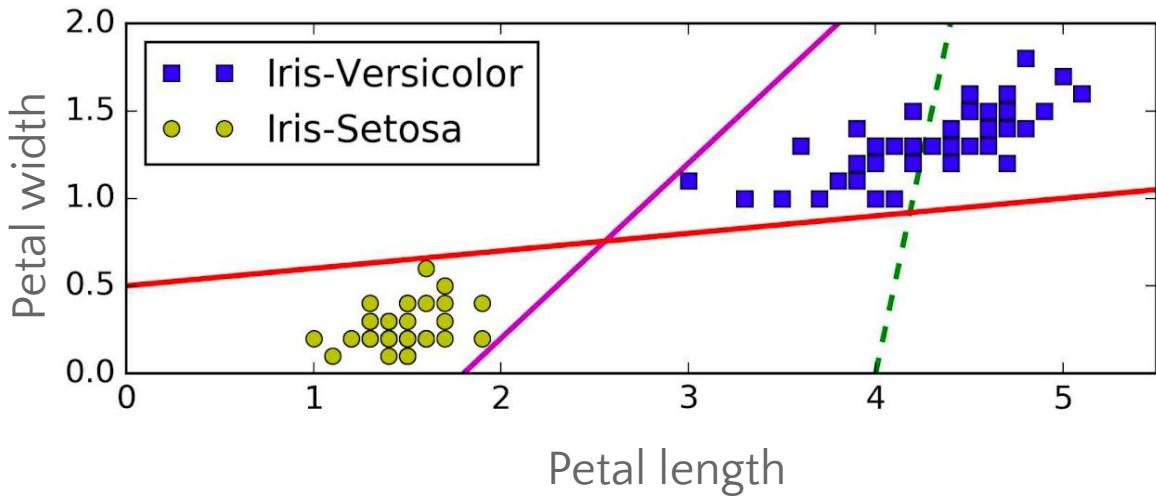
Let's Learn Some More Machine Learning Algorithm



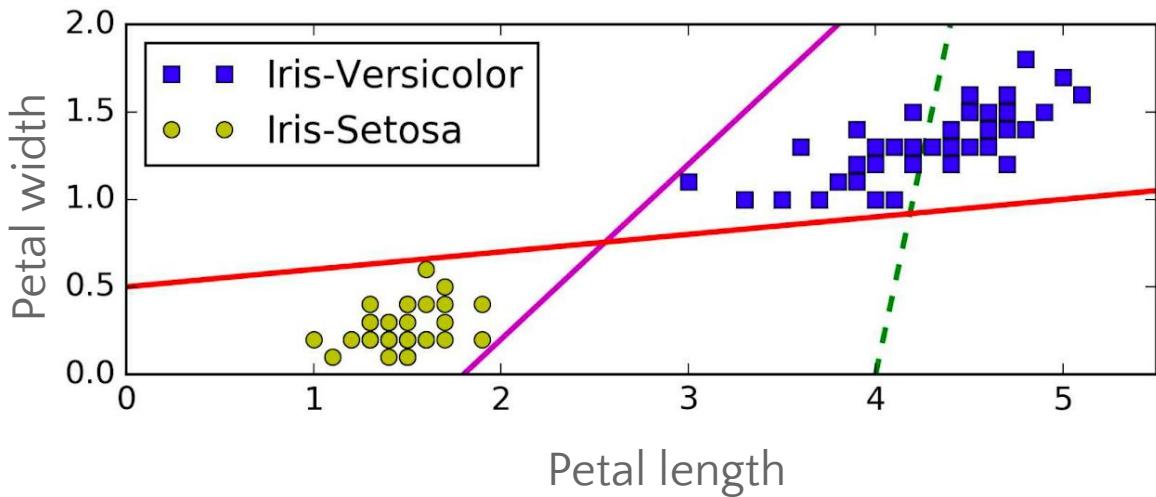
SVM – Support Vector Machines

What is Linear Classification?

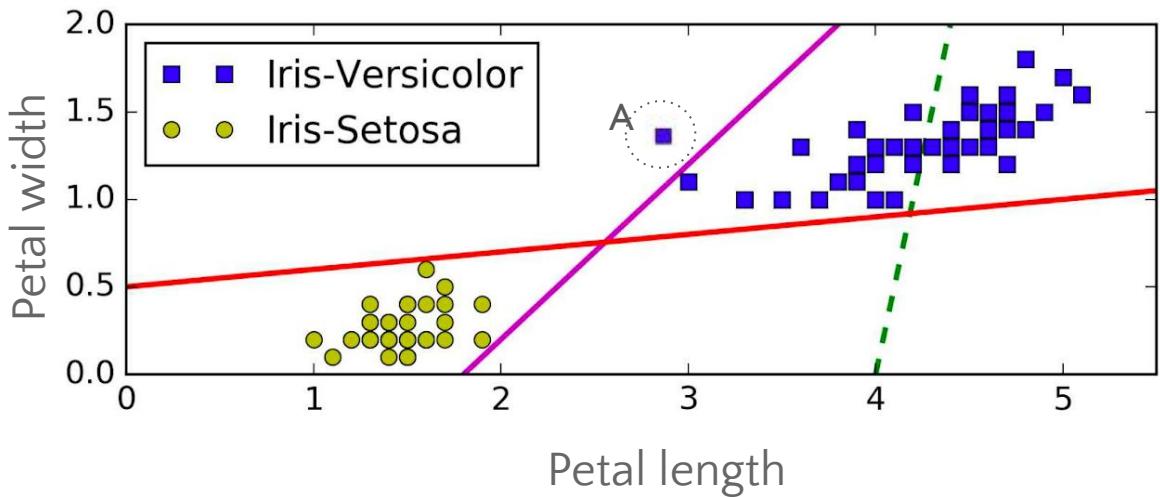
The two classes can be separated easily with a ‘straight’ line



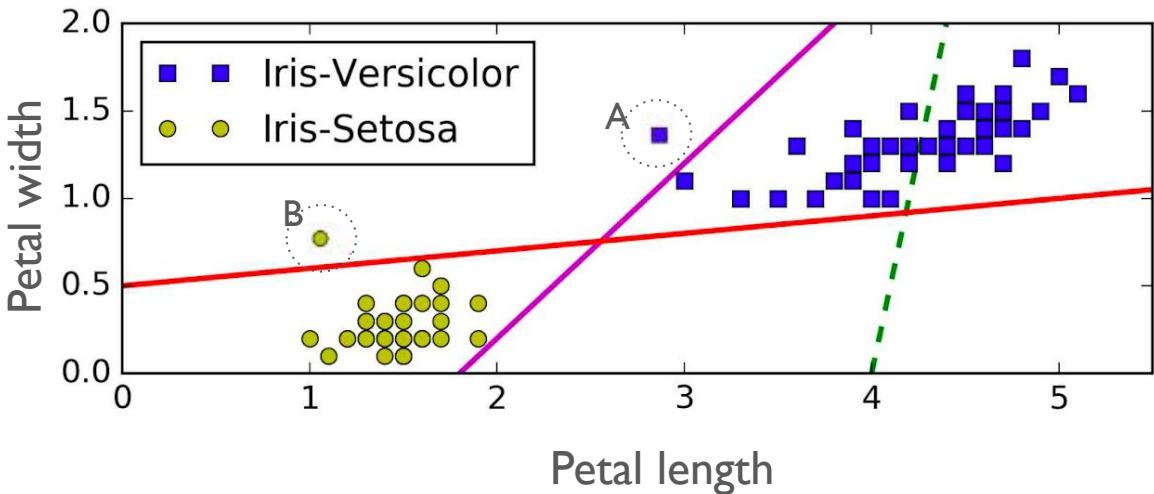
Is Linear Classification good enough?



Is Linear Classification good enough?

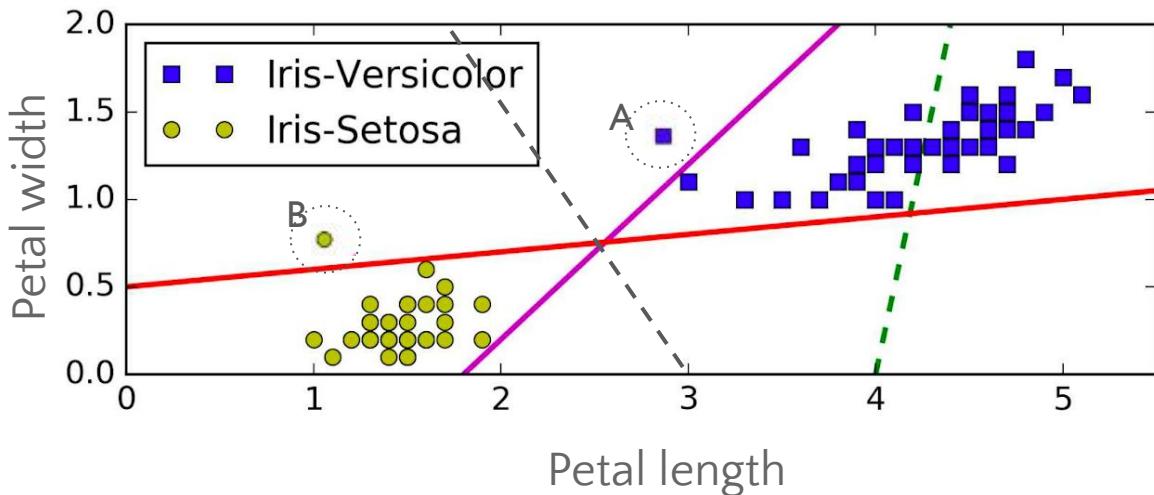


Is Linear Classification good enough?

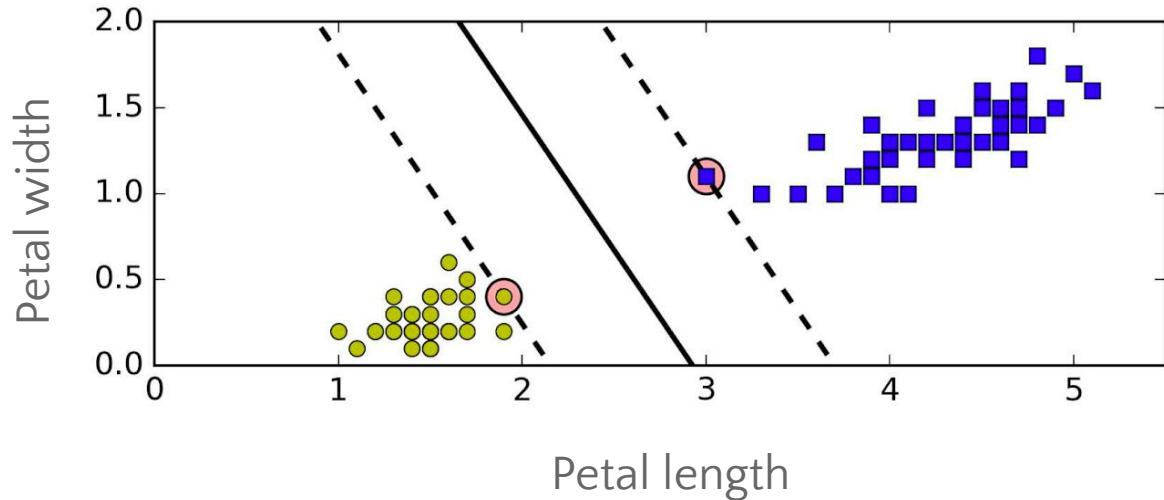


Is Linear Classification good enough?

How about this line?

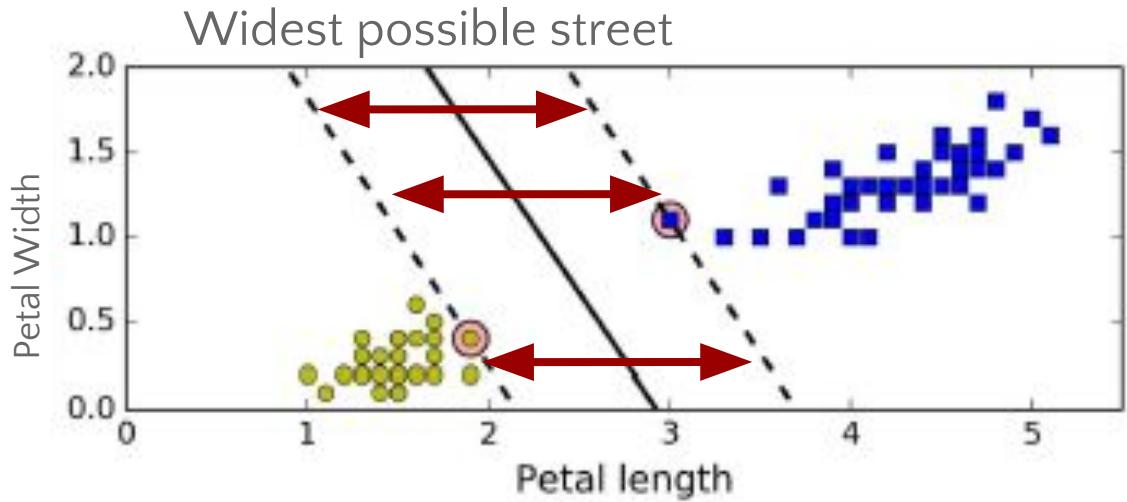


What is Linear Classification?

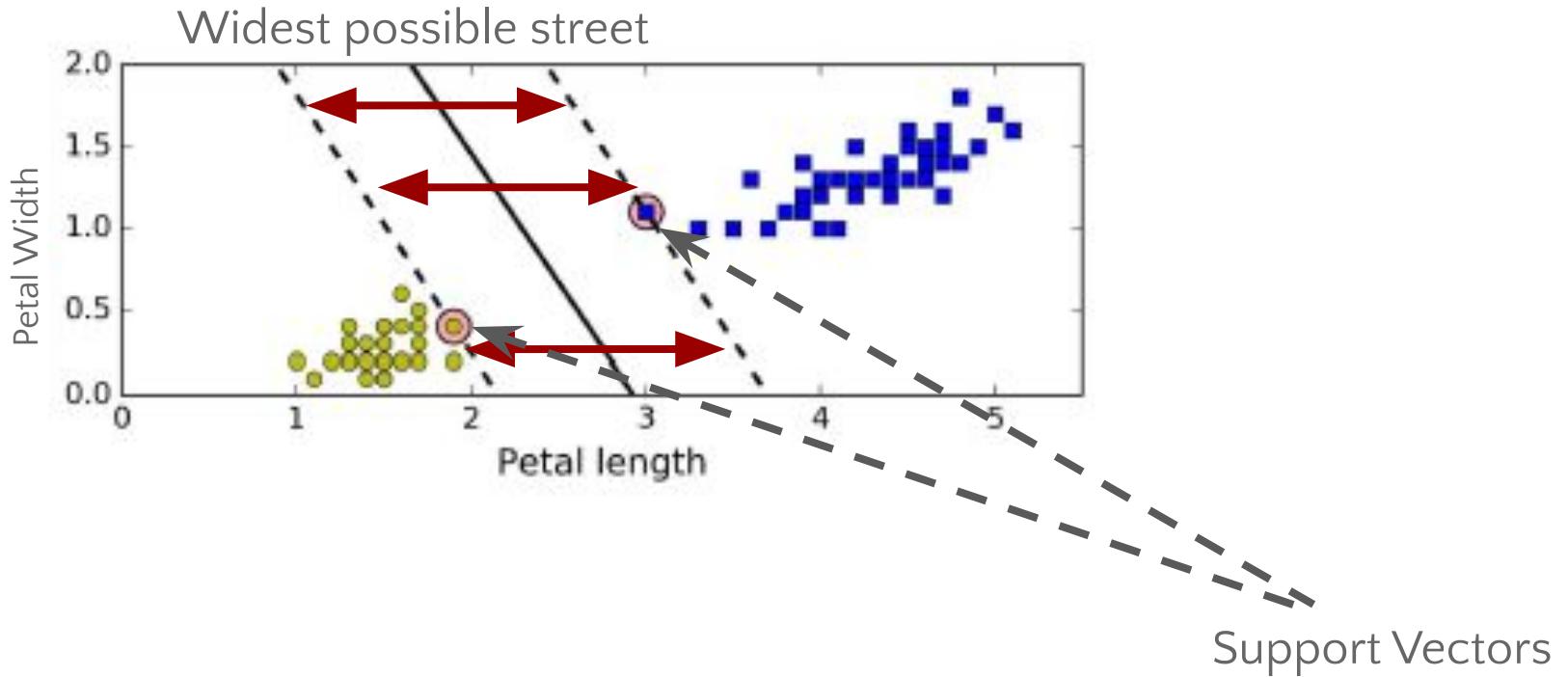


‘Straight’ is the keyword. It means linear classification.

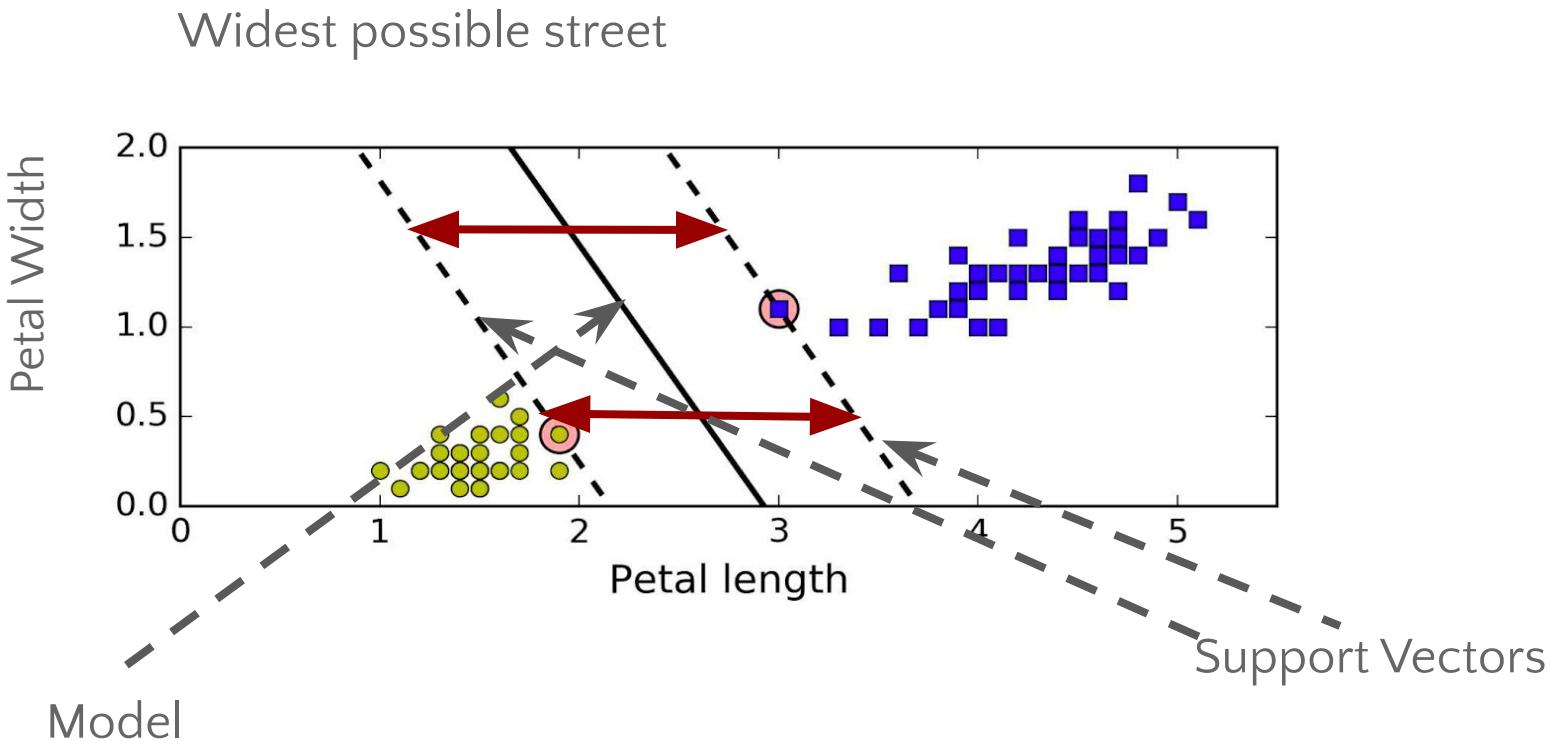
What is Linear Classification?



What is Linear Classification?



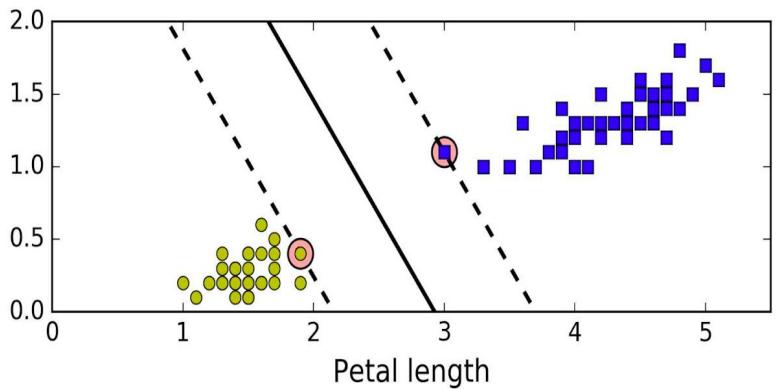
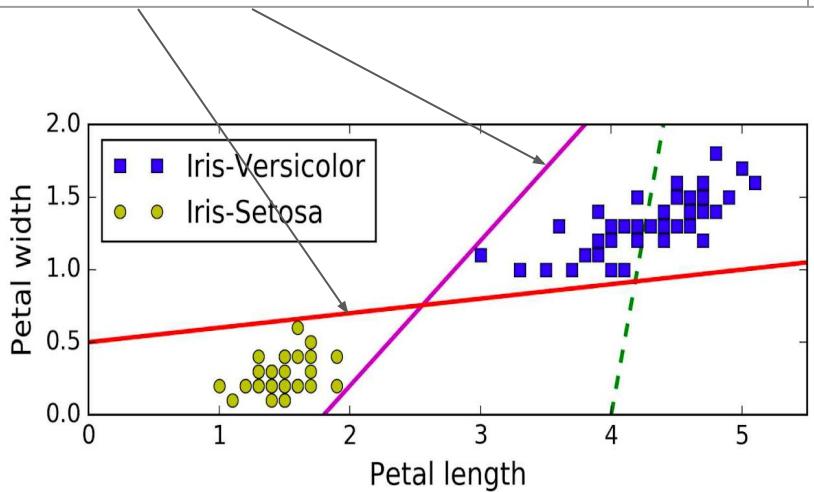
What is Linear Classification?



Linear SVM Classification - Large Margin

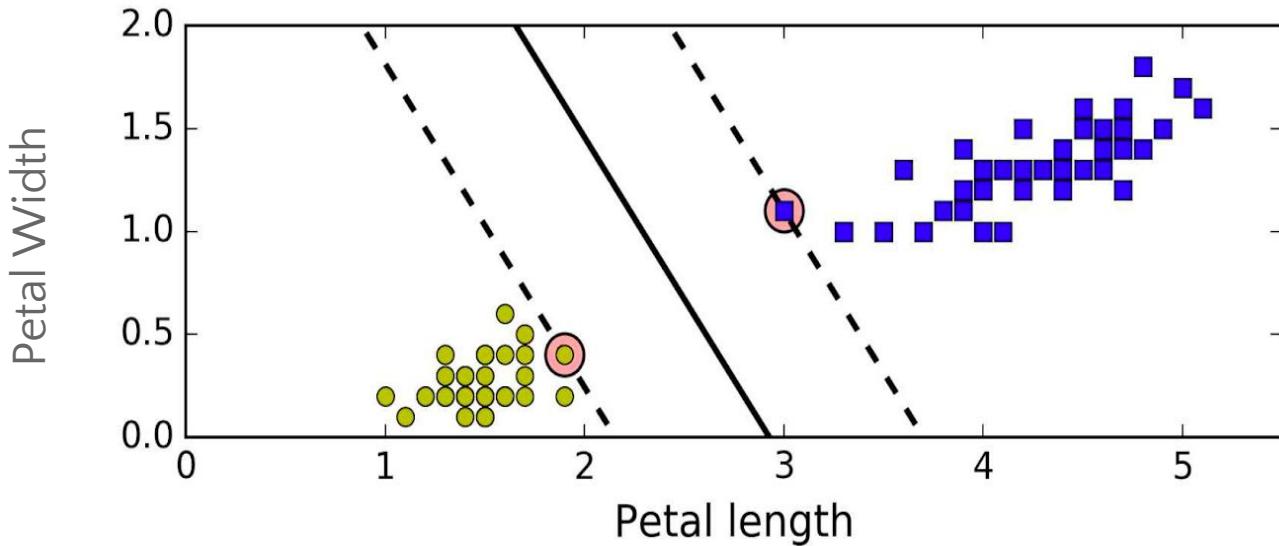
Violet and red decision boundaries are very close to the instances - bad model

Decision Boundary as far away from training instances - good model



Large Margin Classification

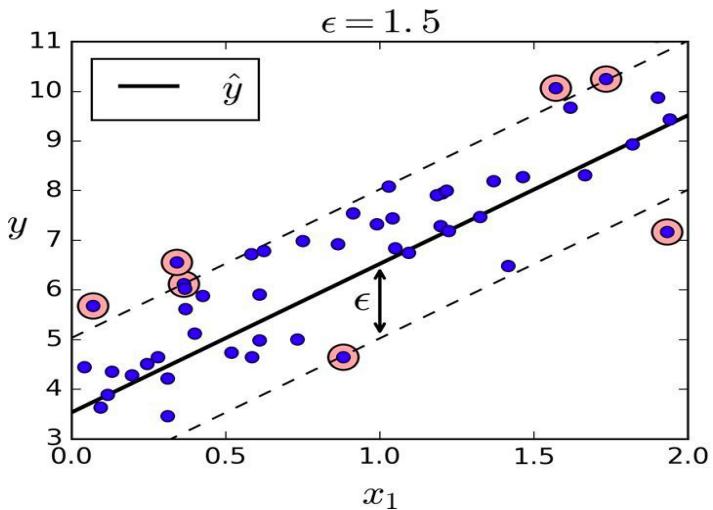
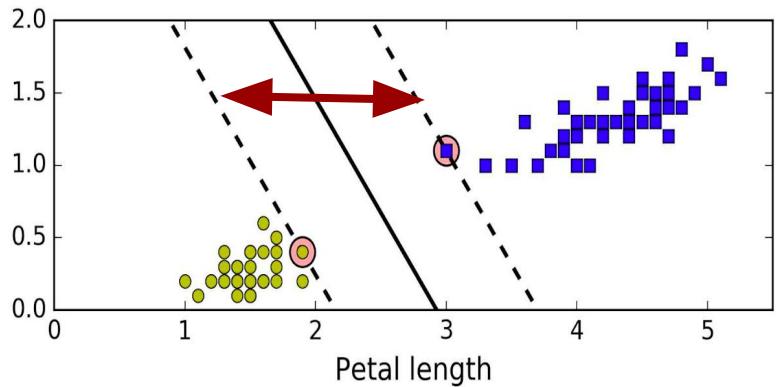
How does SVM work?



SVM Regression

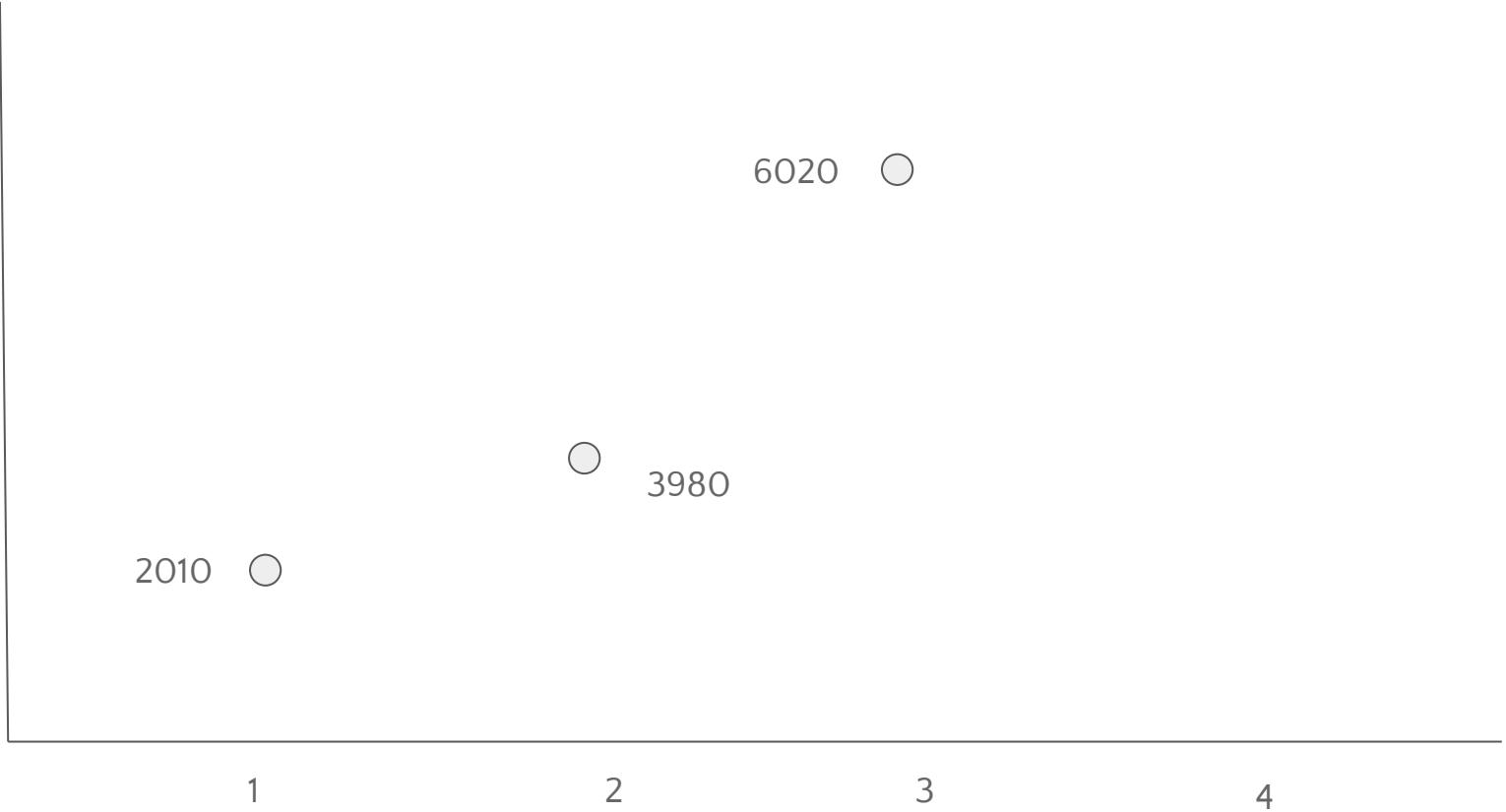
SVM Classifier	SVM Regression
Find the largest possible street between the two classes limiting margin violations	Fit as many instances as possible on the street while limiting margin violations

Widest possible street

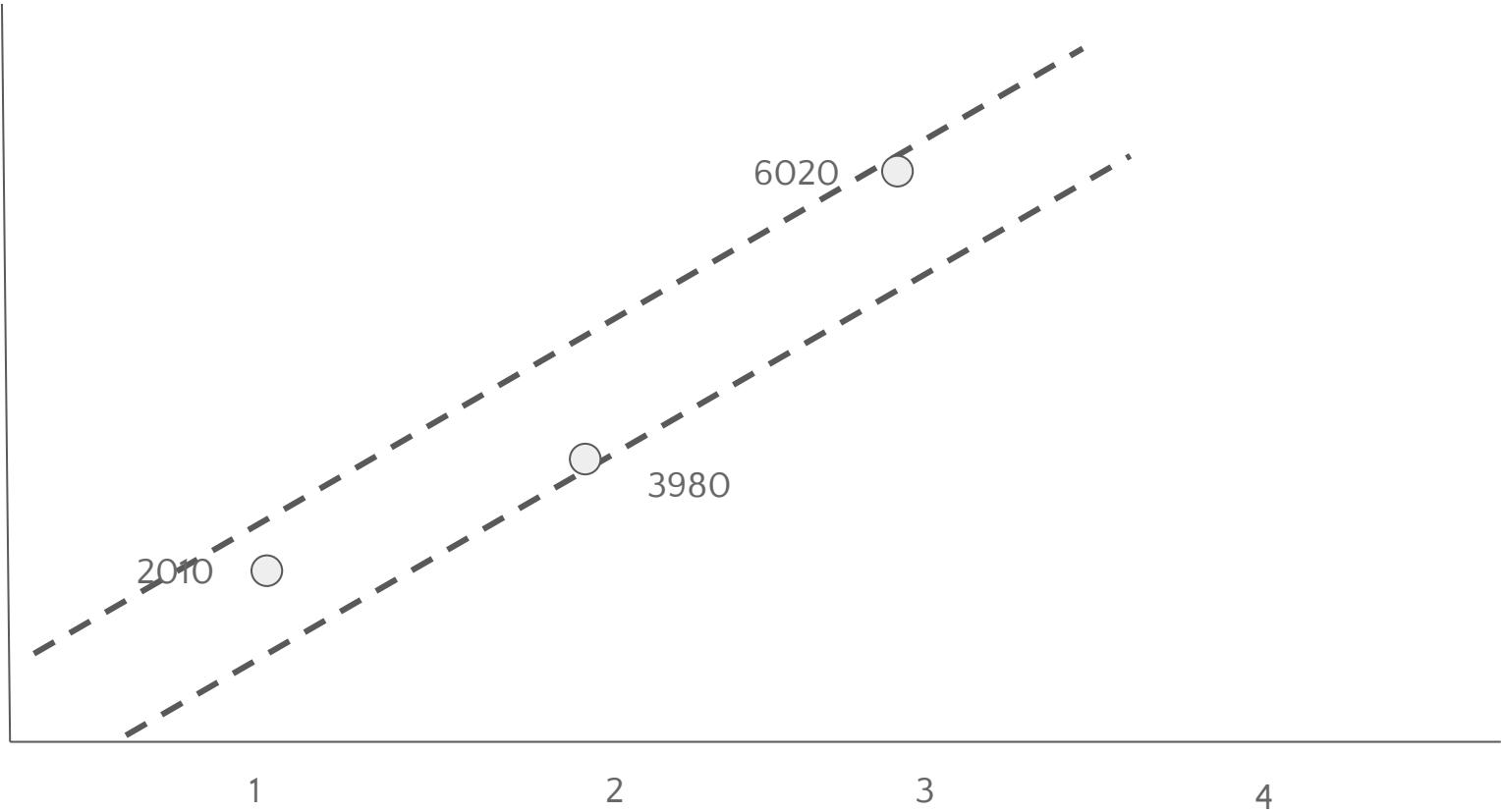




Can we solve it with SVM regression?

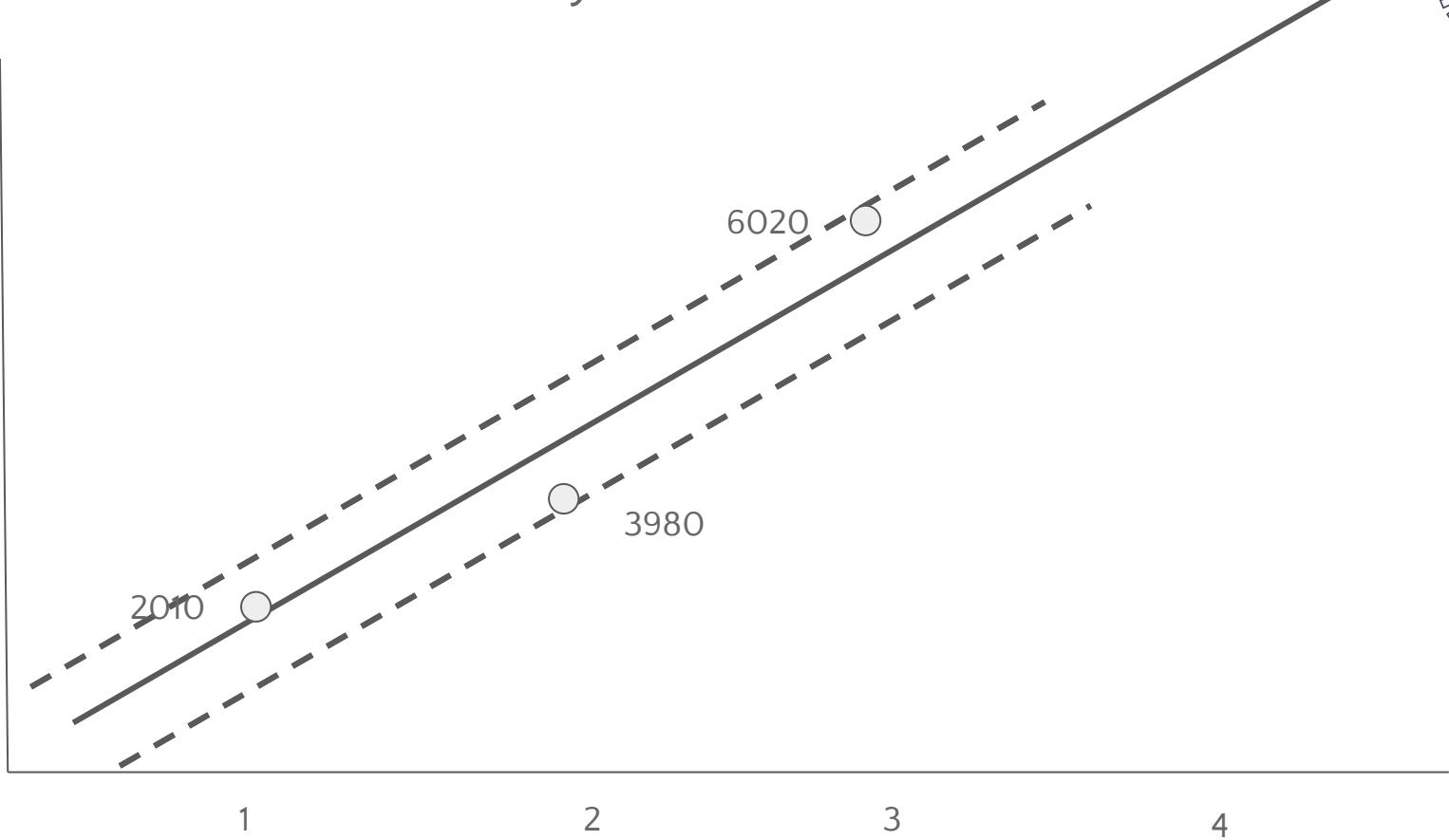


Find narrowest street that can fit the data points

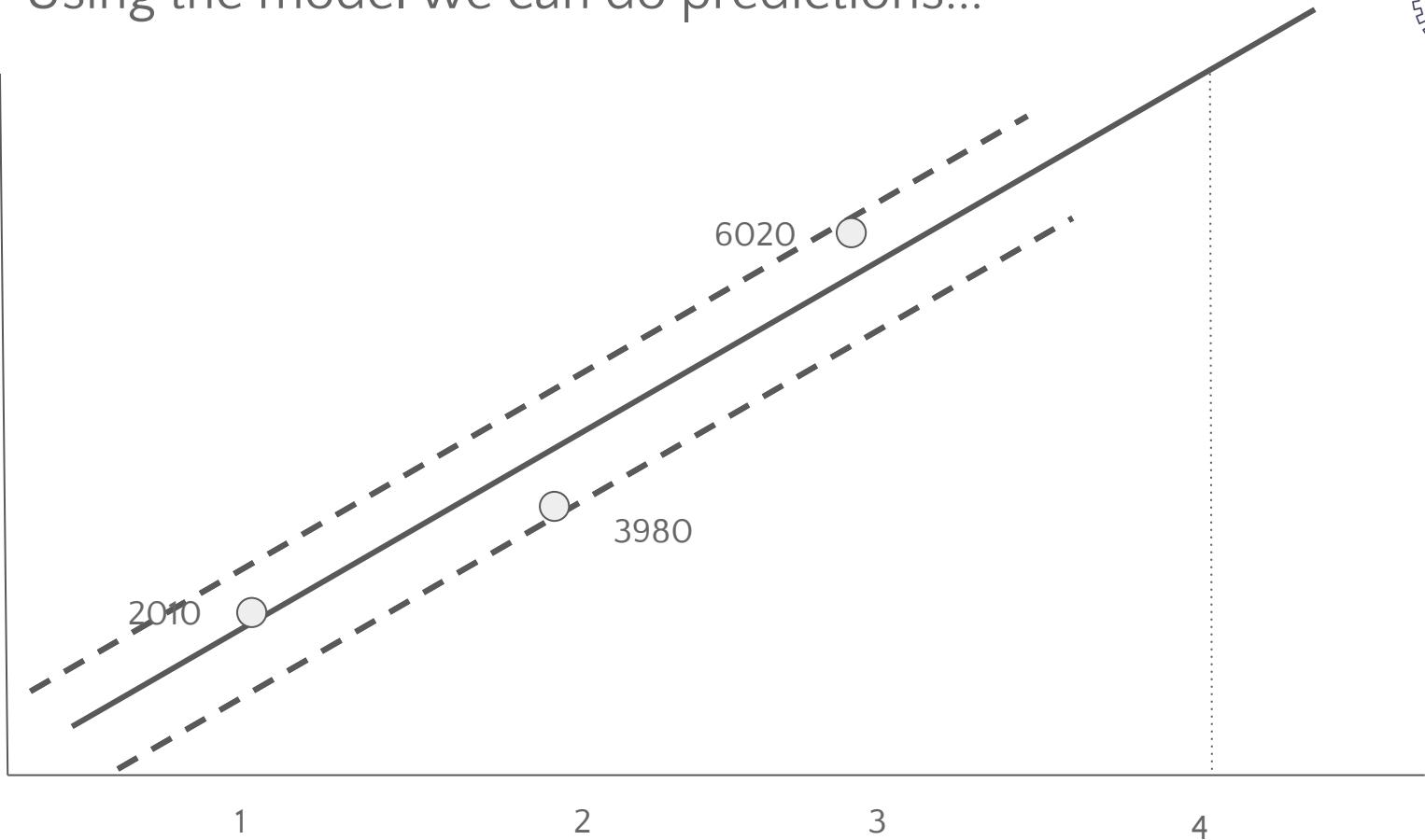




Central line of street is your model!

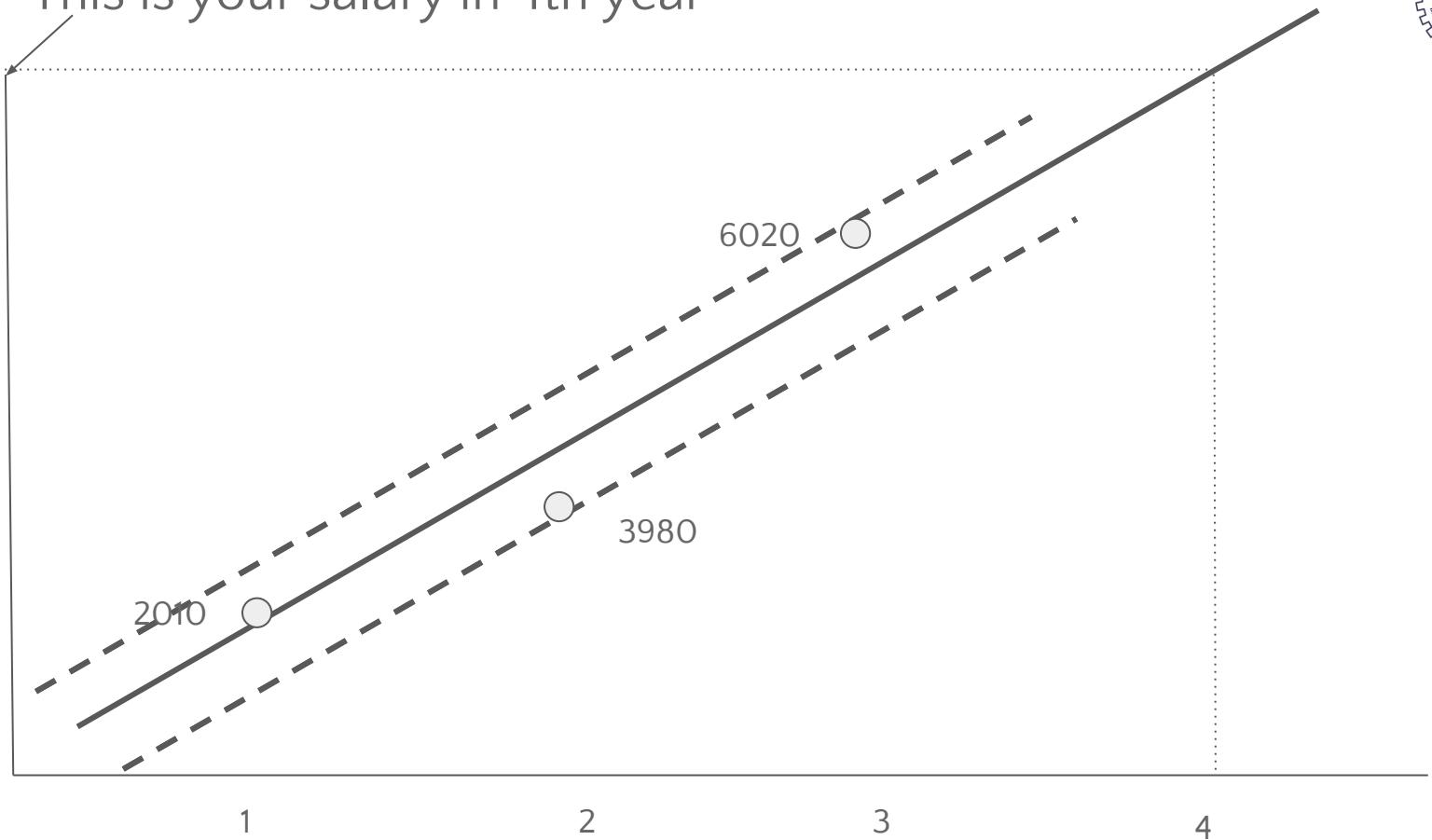


Using the model we can do predictions...





This is your salary in 4th year





SVM

Check the code of SVM in Notebook



SVM

- Very powerful and versatile model
- Capable of performing
 - Classification
 - Regression
 - Outlier detection
- Before neural networks, they were best performing
- Well suited for small or medium sized datasets



Ensemble Learning

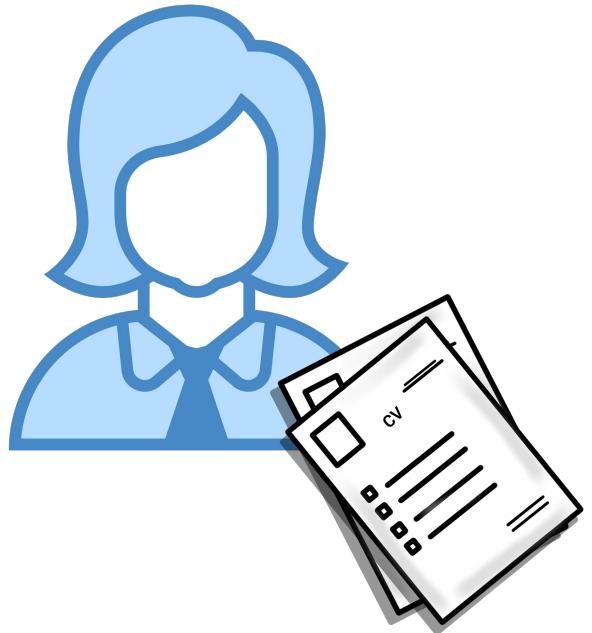


Question - We are all biased. What should we do?



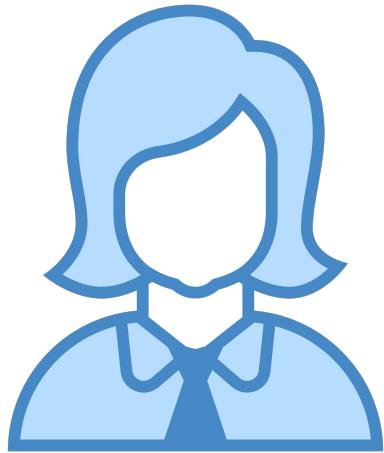
Question - We are all biased. What should we do?

Answer - Create an Ensemble





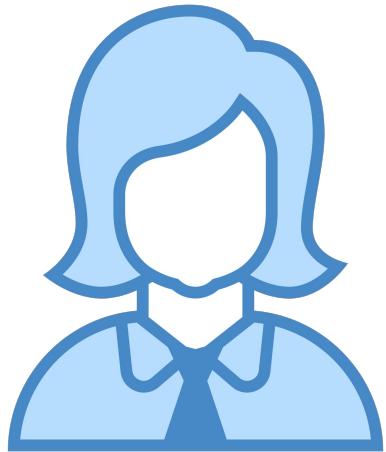
Expert Interviewer



Interviewee



Expert Interviewer

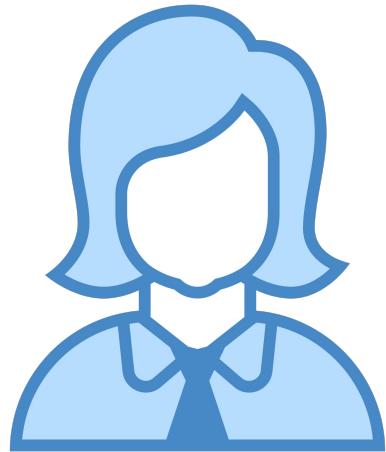


Interviewee





Expert Interviewer



Interviewee

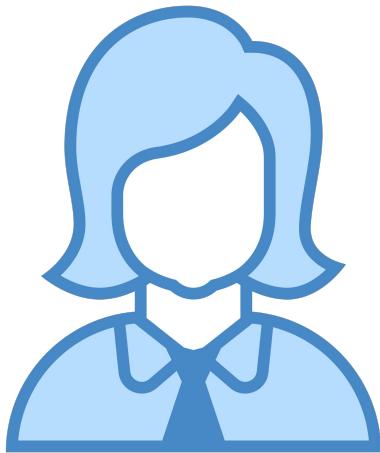




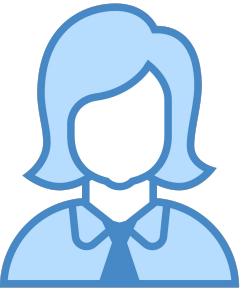
Biased Interviewer



Expert Interviewer



Interviewee



Interviewee

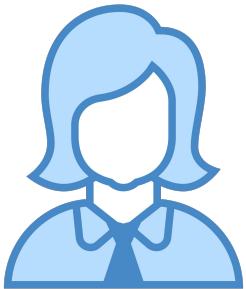


Panel of Interviewers





Panel of Interviewers

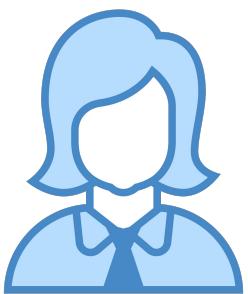


Interviewee





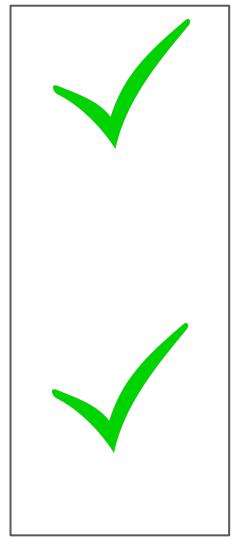
Panel of Interviewers



Interviewee



Total No - 1

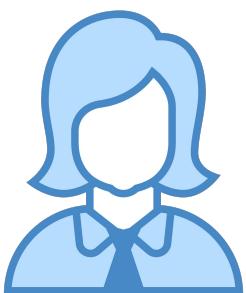


Total Yes - 2





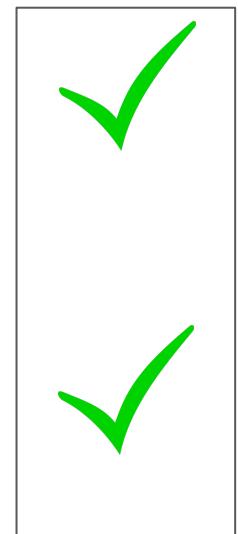
Panel of Interviewers



Interviewee



Total No - 1



Total Yes - 2

Final Decision
Based on
Majority



Hire





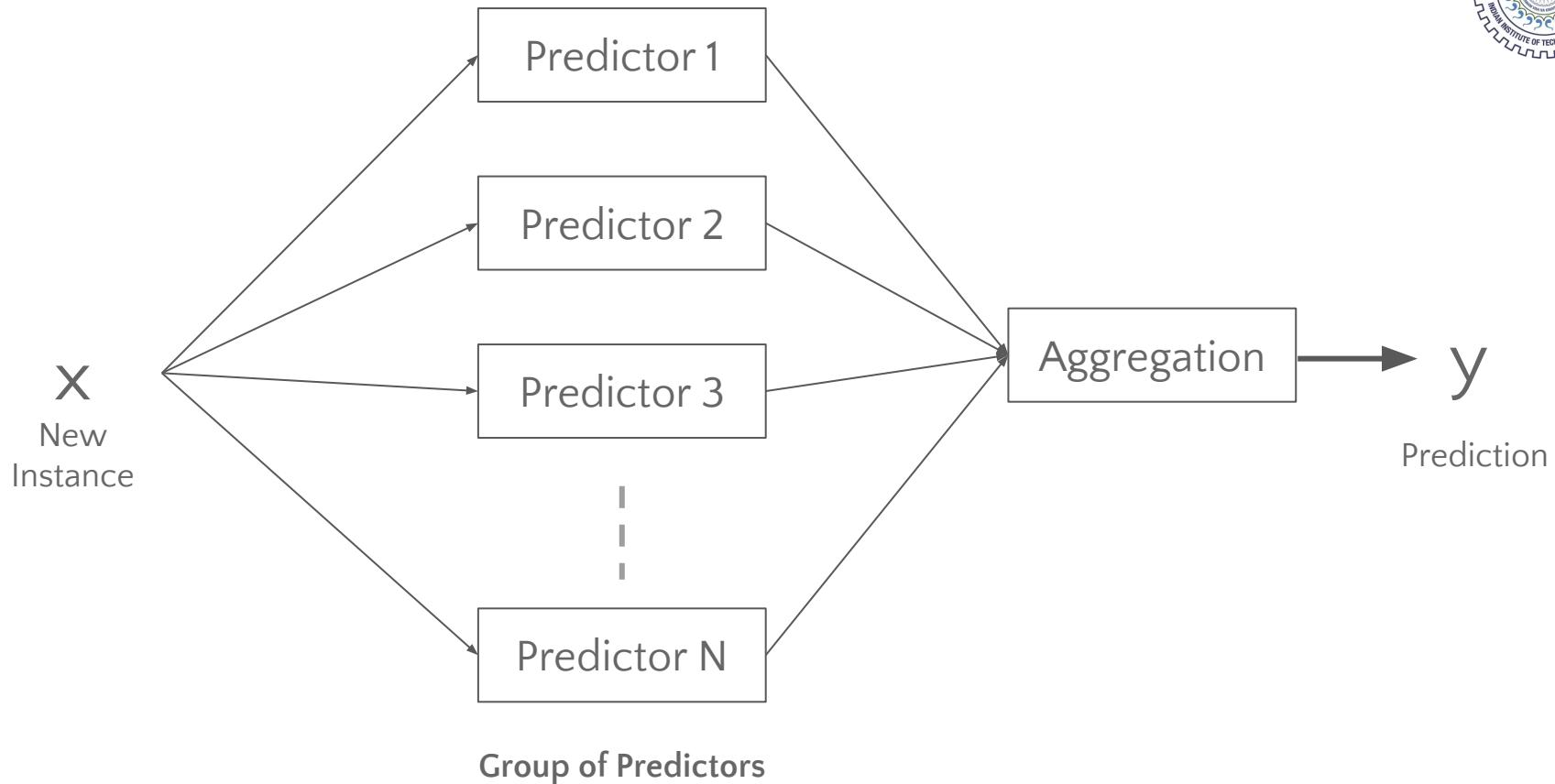
Panel of Interviewers

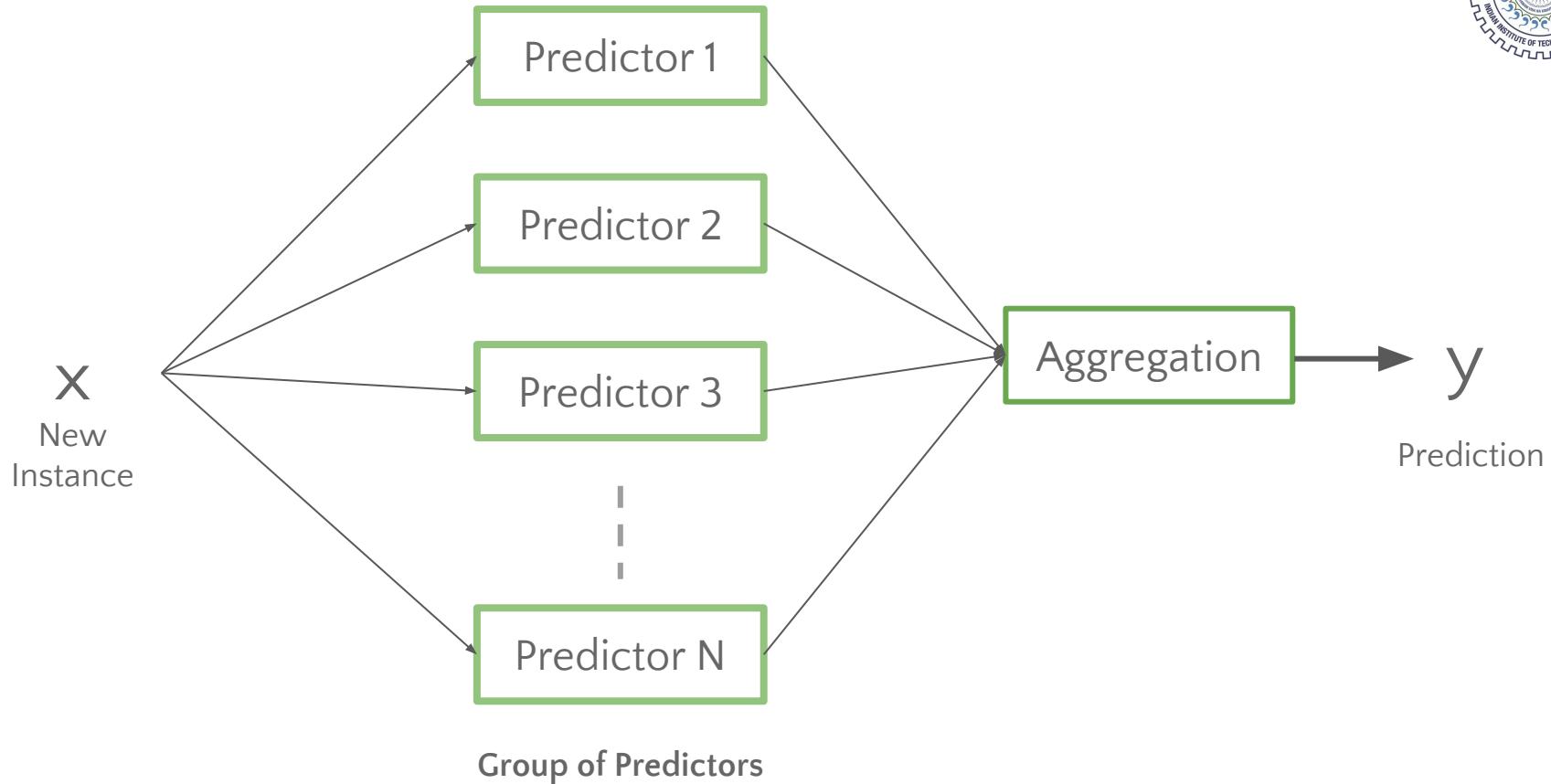


Better Decision



Expert Interviewer

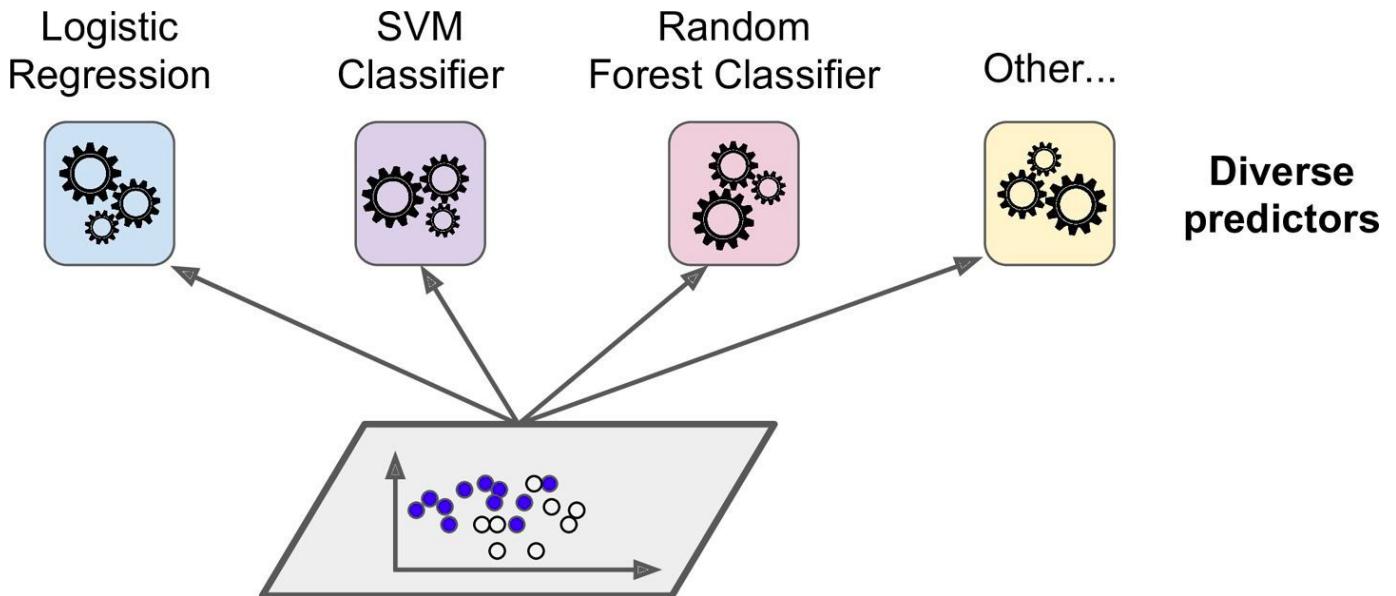


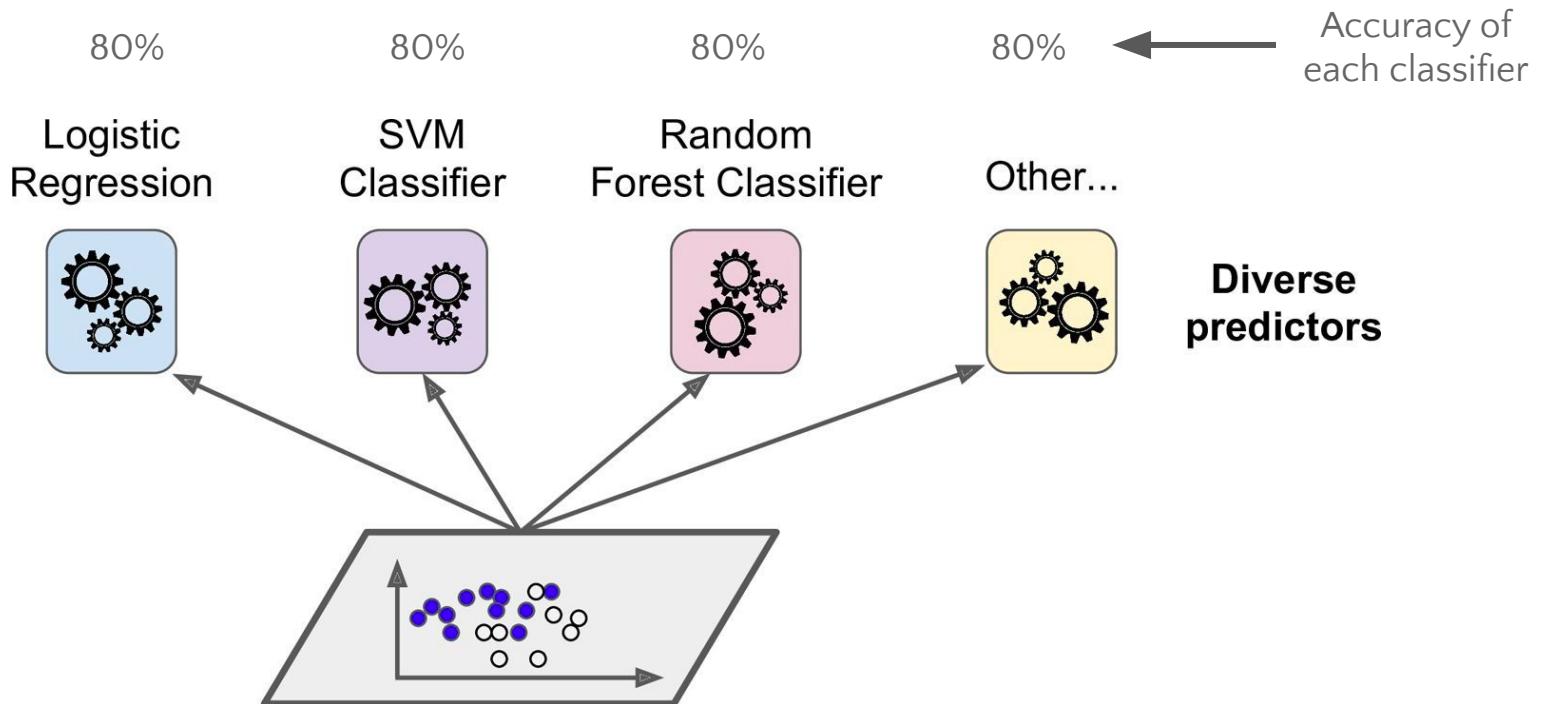


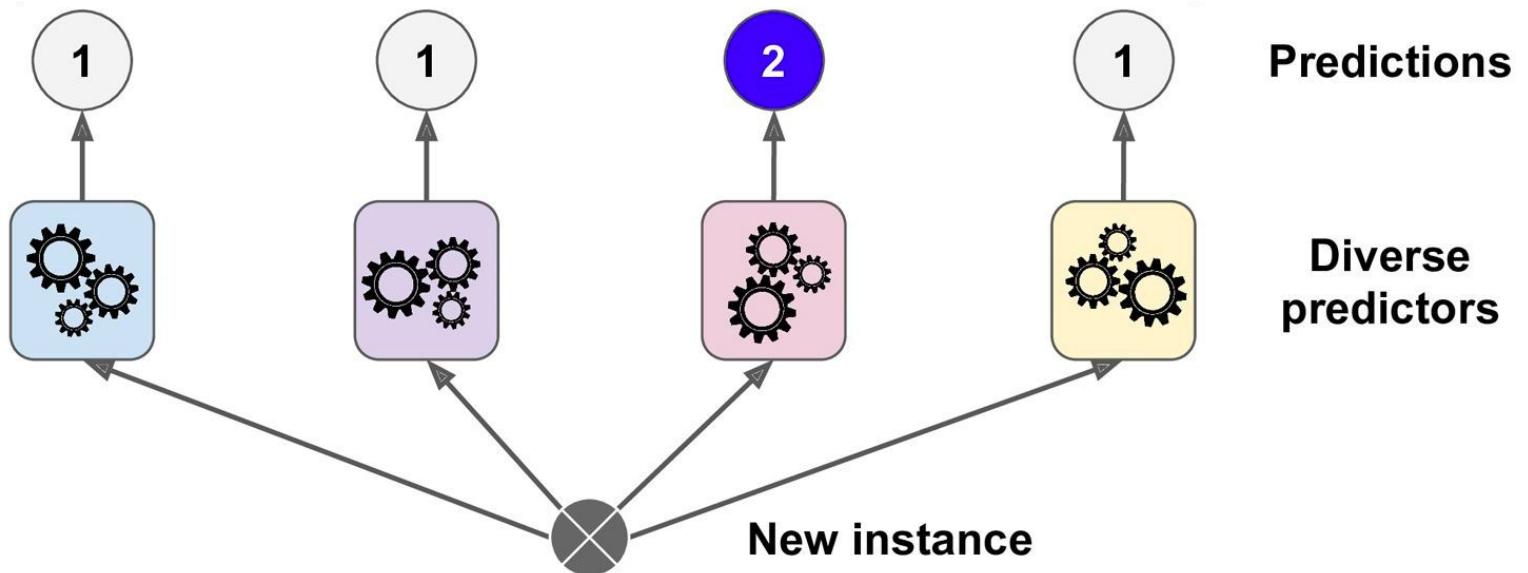


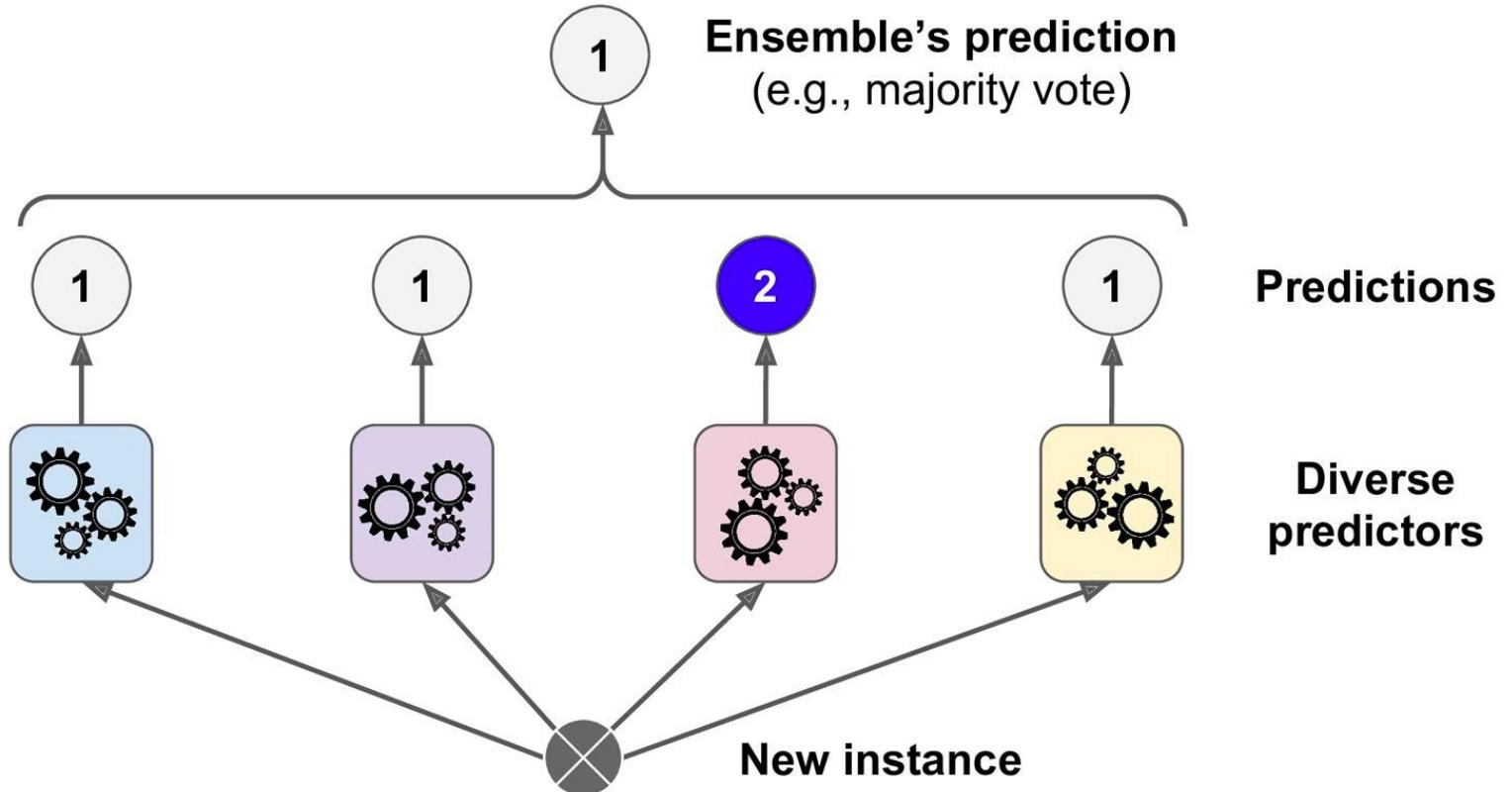
Checklist for Machine Learning Projects

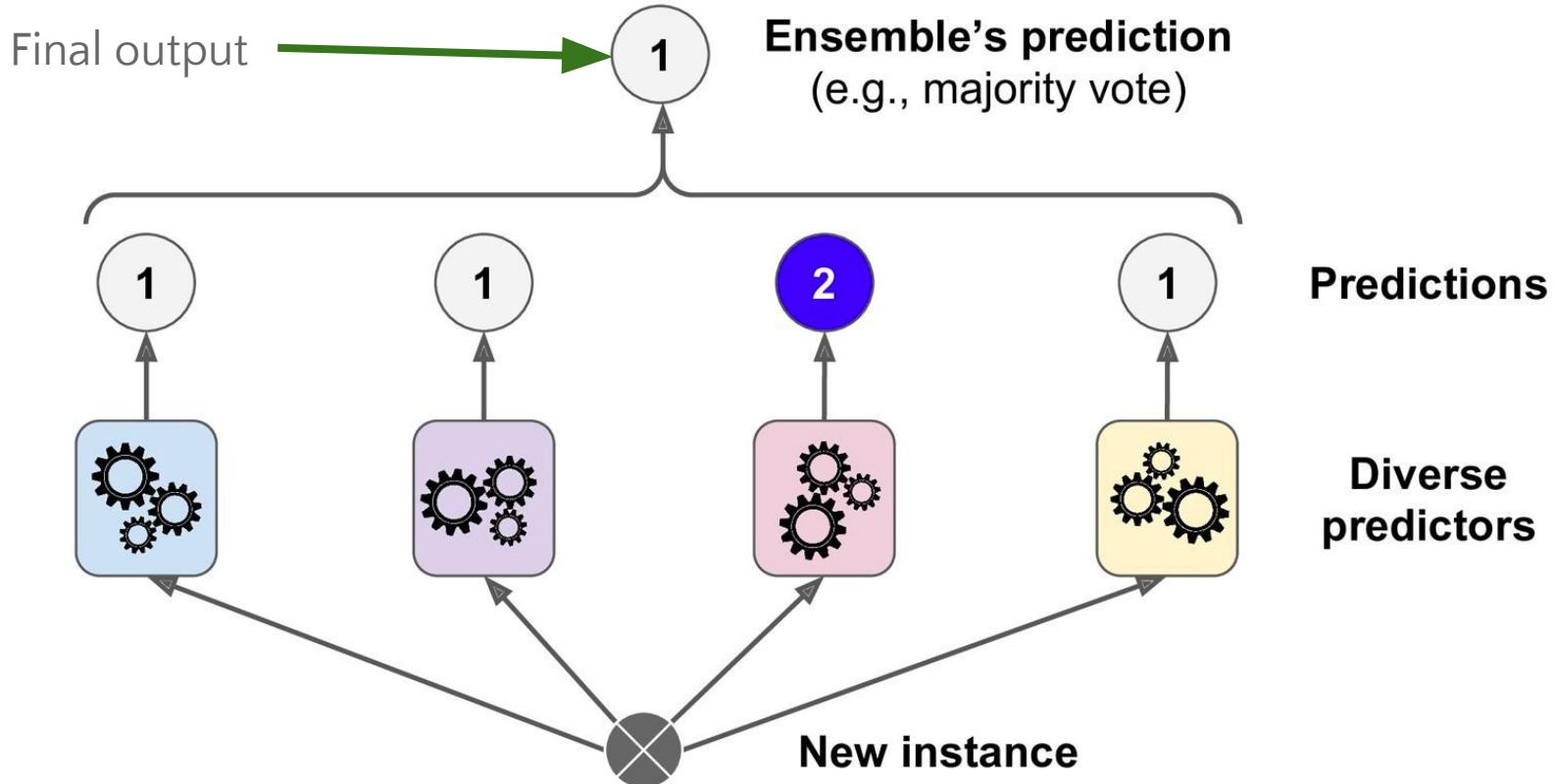
1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data for Machine Learning algorithms
5. Explore many different models and short-list the best ones
- 6. Fine-tune model**
7. Present the solution
8. Launch, monitor, and maintain the system

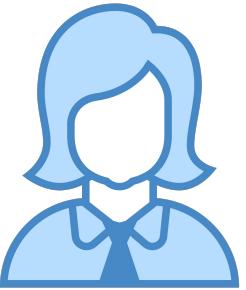












Interviewee

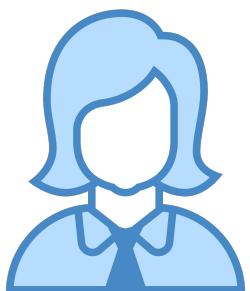


Panel of Interviewers

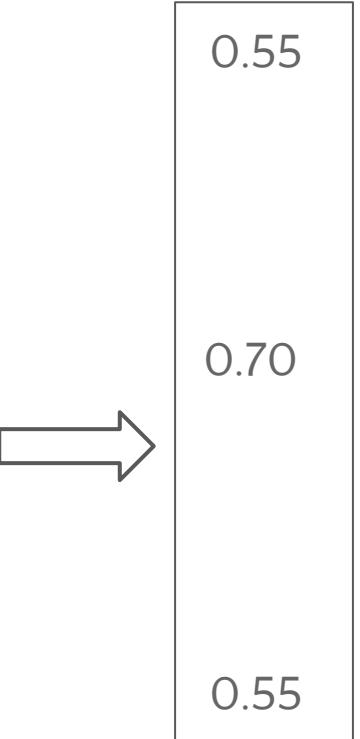




Panel of Interviewers



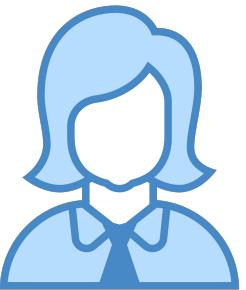
Interviewee



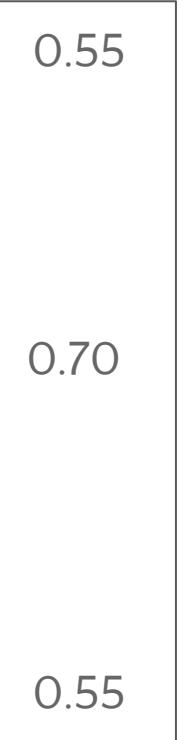
Probability of hiring by
each interviewer



Panel of Interviewers



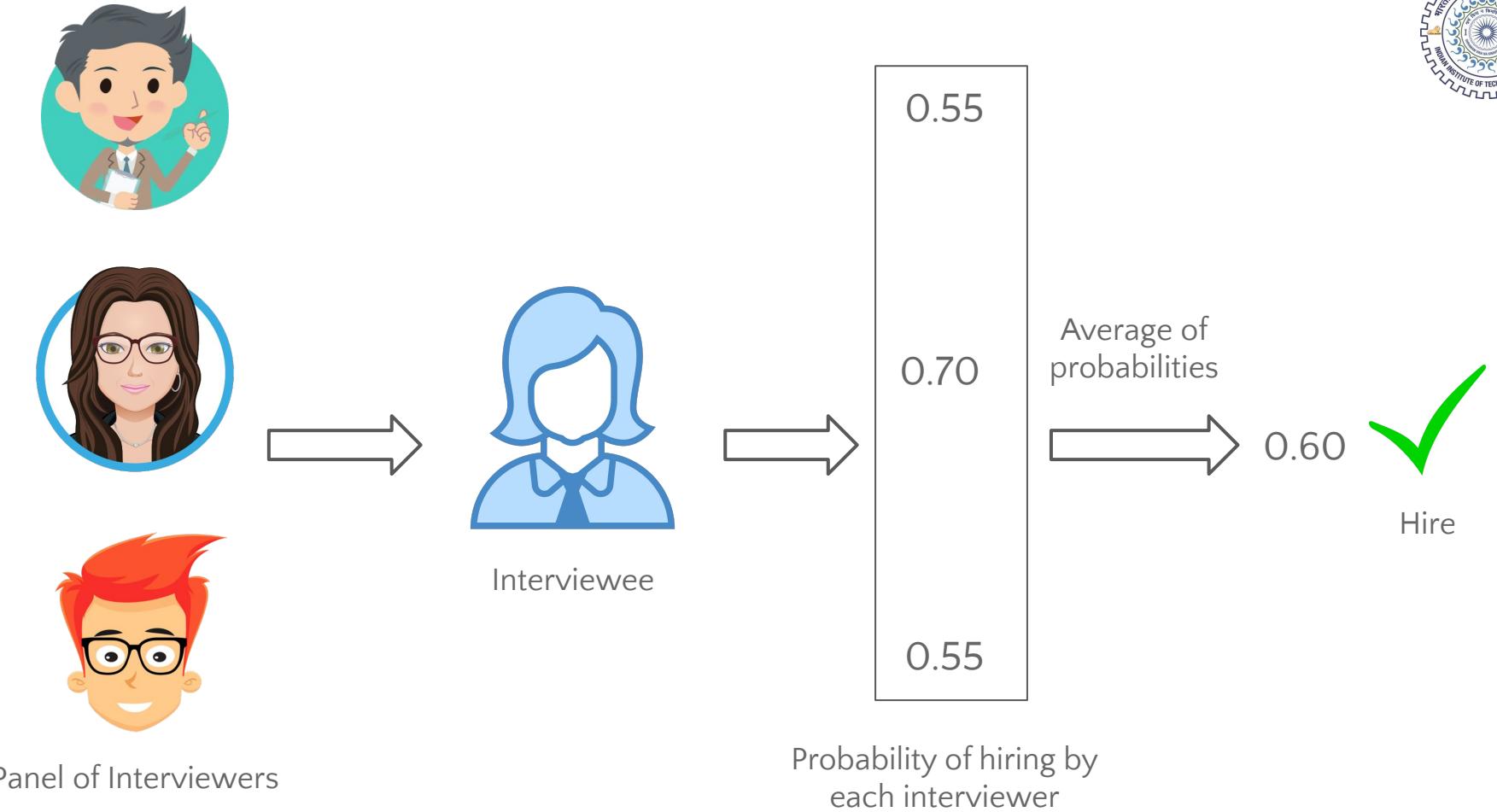
Interviewee

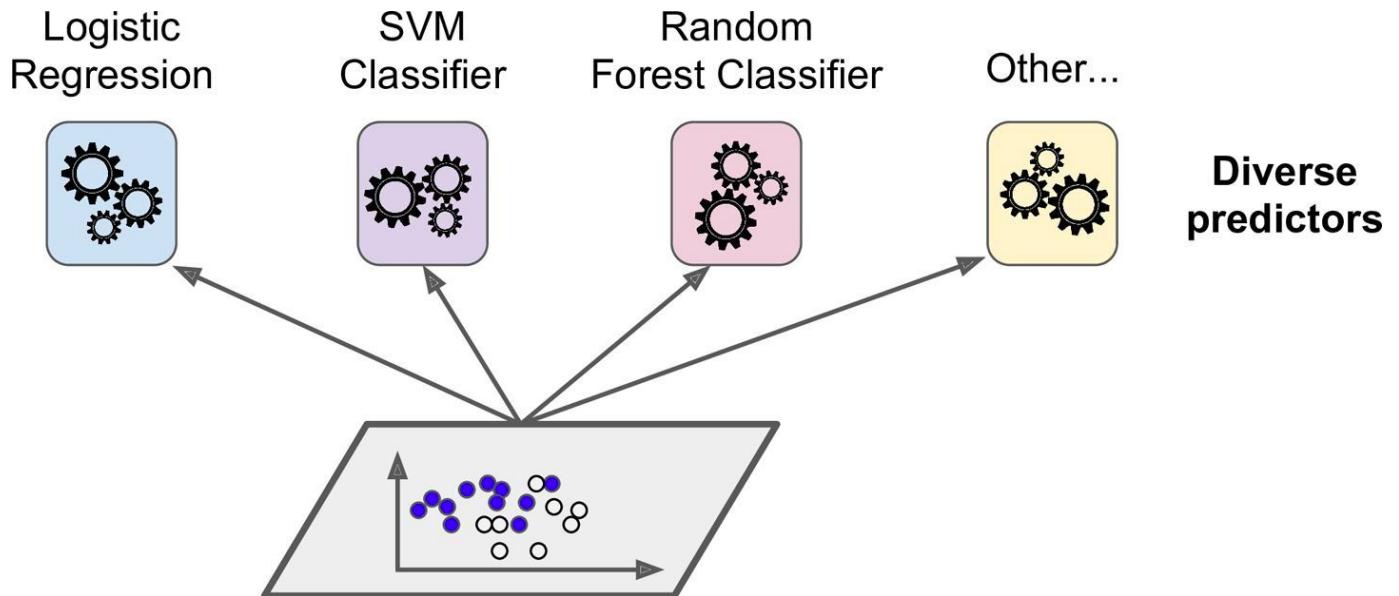


Probability of hiring by
each interviewer

Average of
probabilities

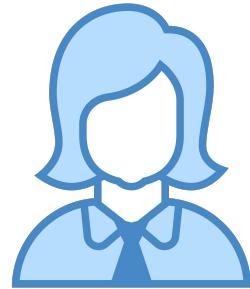
0.60







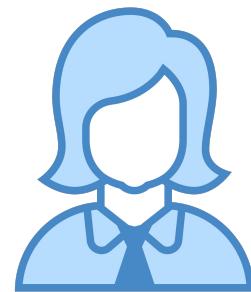
Panel of Interviewers



Interviewee



Communication Skills and
Attitude



Interviewee



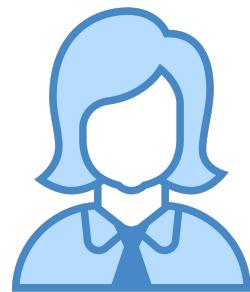
Panel of Interviewers



Communication Skills and
Attitude



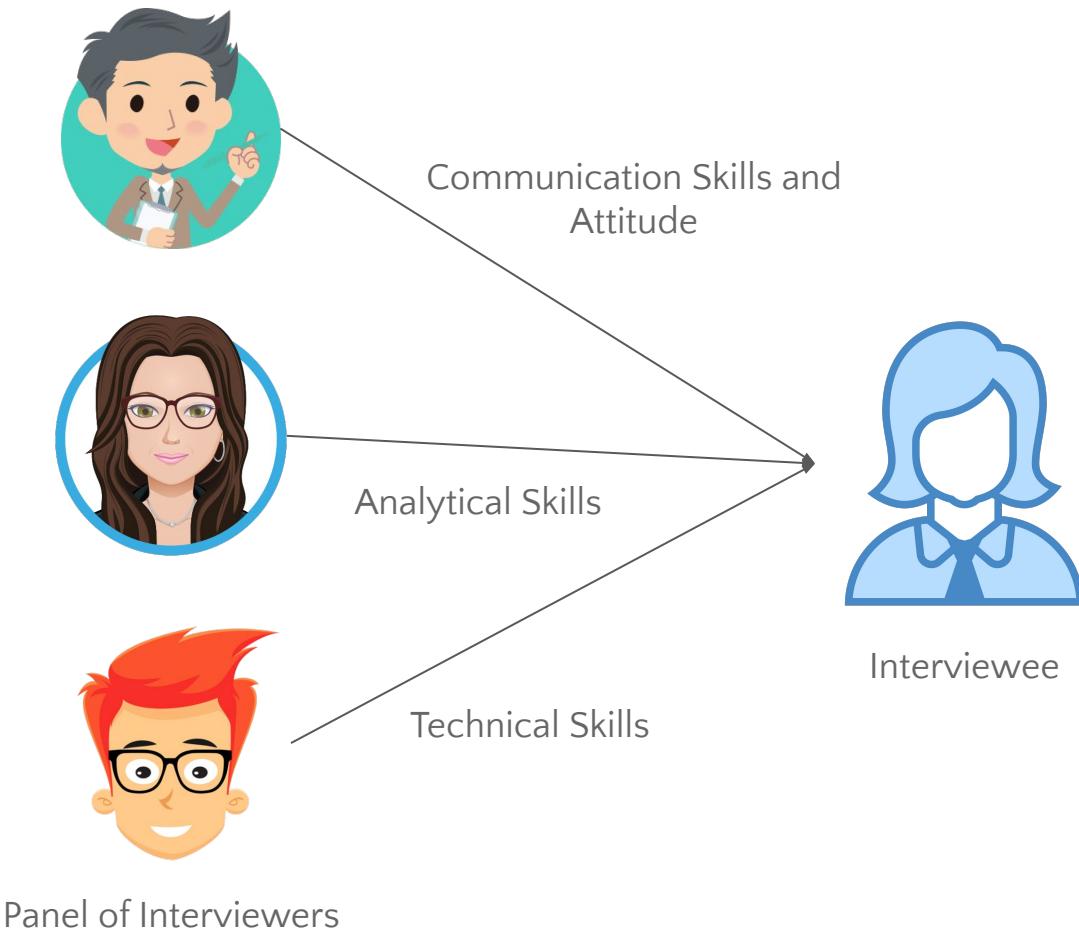
Analytical Skills

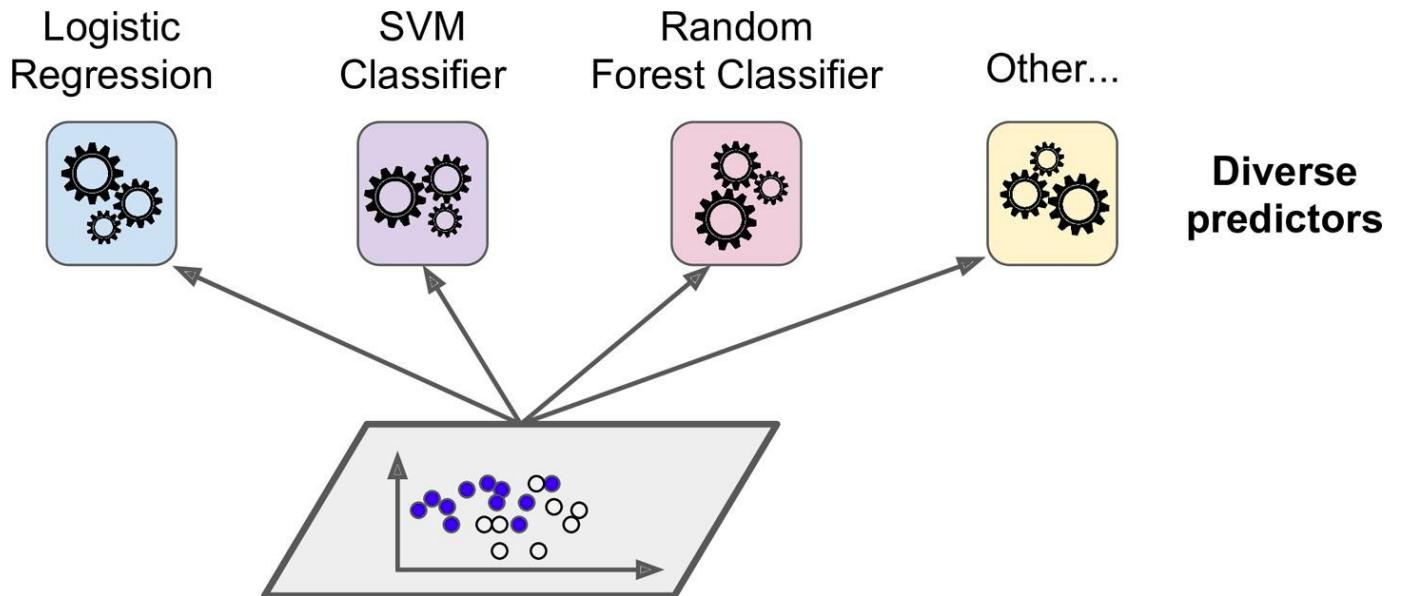


Interviewee



Panel of Interviewers







- In ensemble we use diverse models



- In ensemble we use diverse models
- These diverse models need to be better than random guessing



- In ensemble we use diverse models
- These diverse models need to be better than random guessing
- Ensembling can be used for both



- In ensemble we use diverse models
- These diverse models need to be better than random guessing
- Ensembling can be used for both
 - Classification

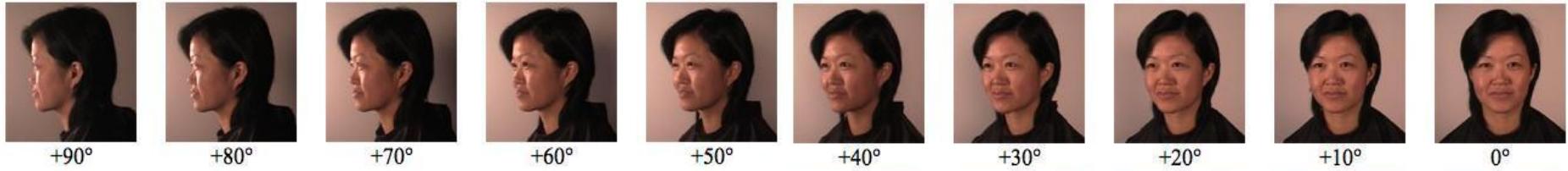


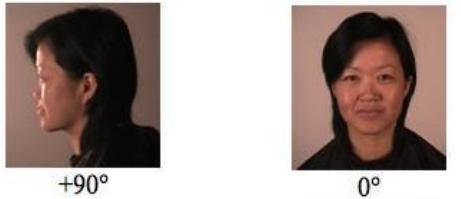
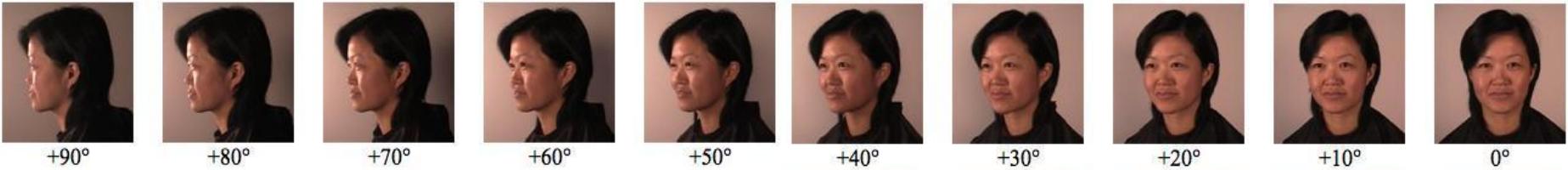
- In ensemble we use diverse models
- These diverse models need to be better than random guessing
- Ensembling can be used for both
 - Classification
 - Regression



Dimensionality reduction

















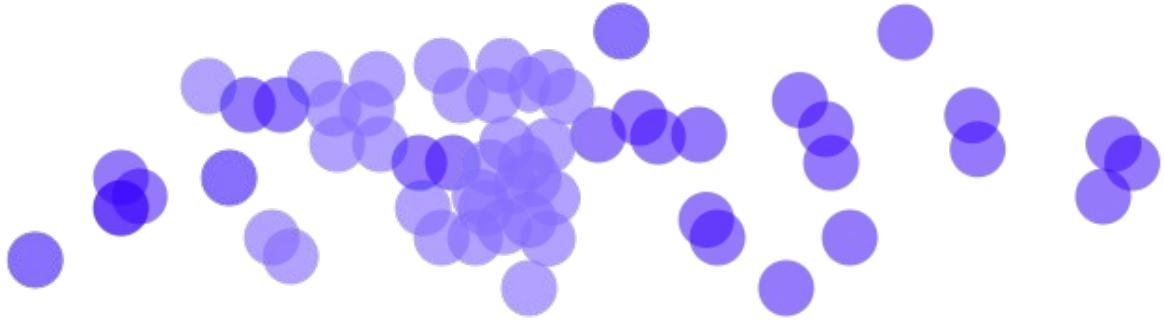


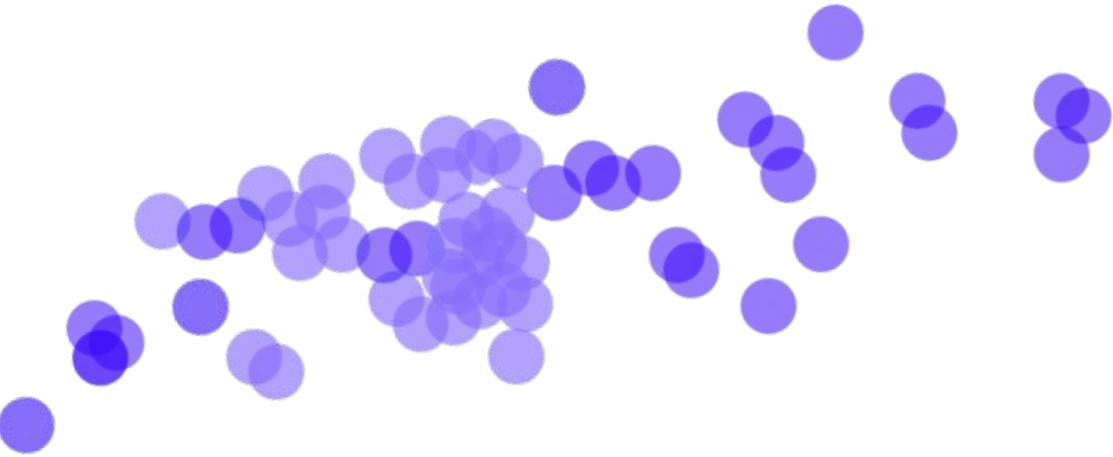


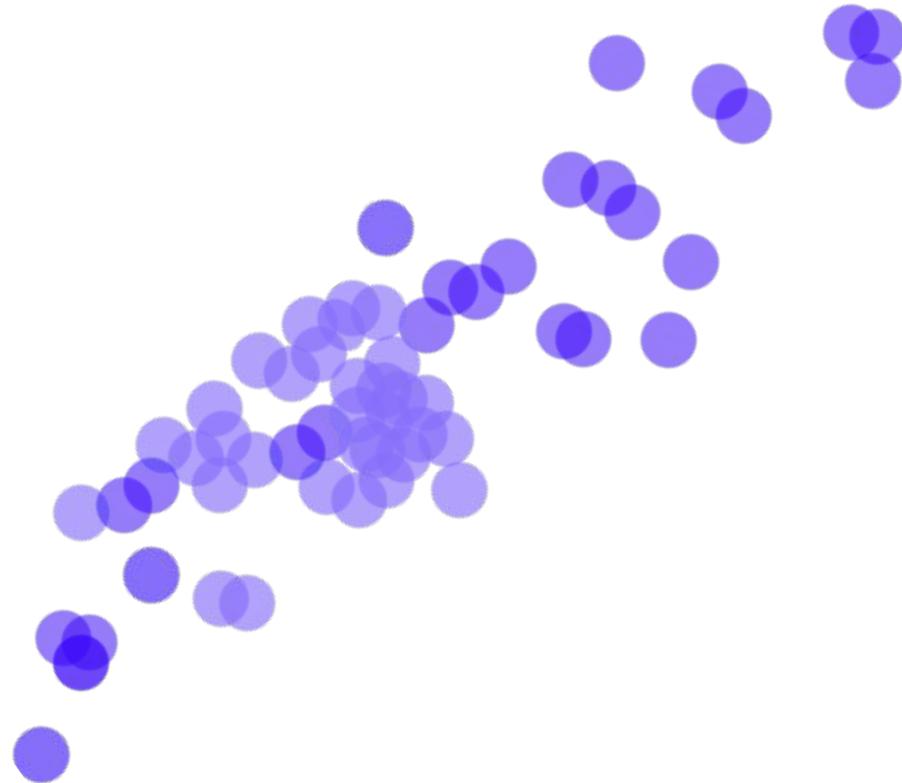
Principal Component Analysis

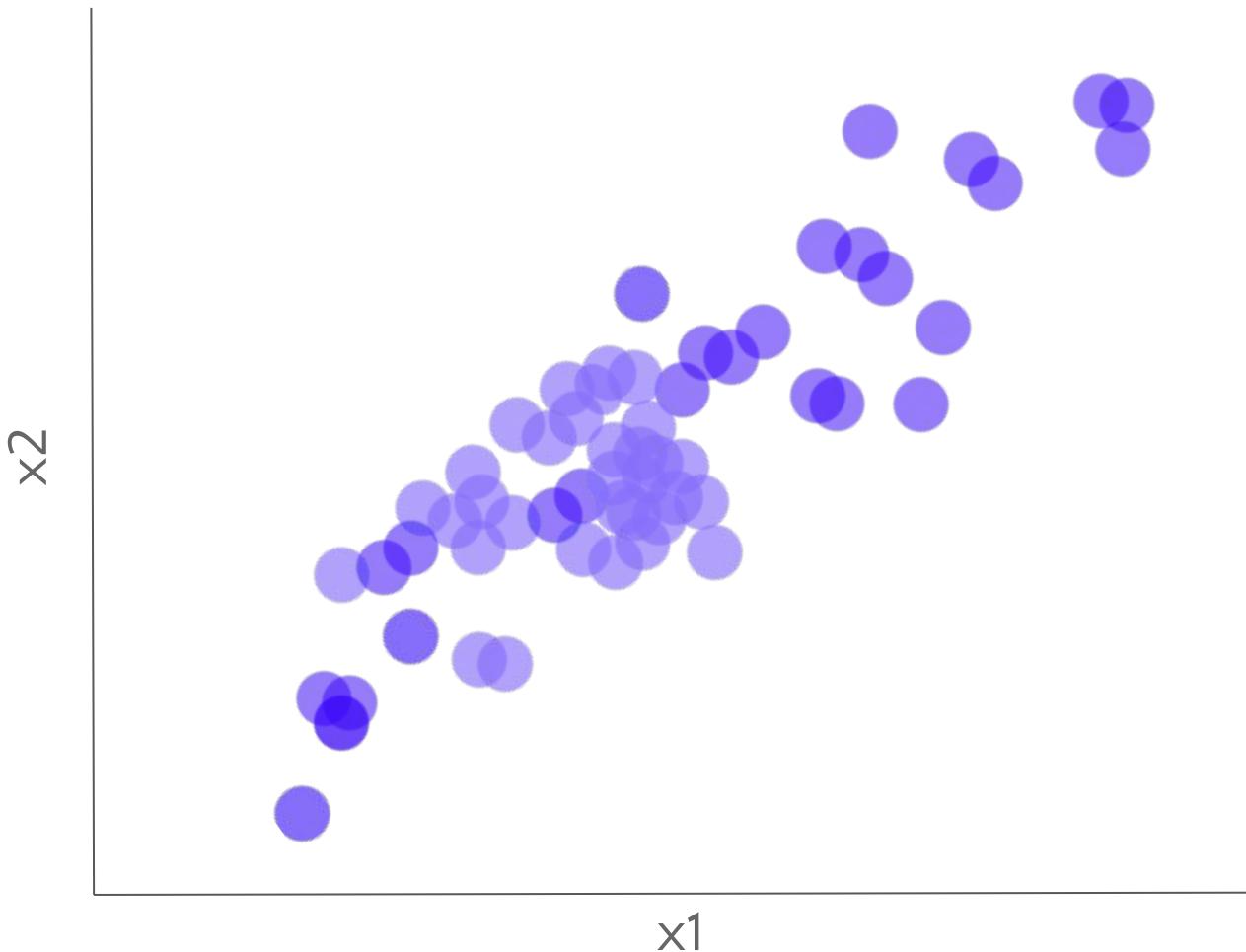


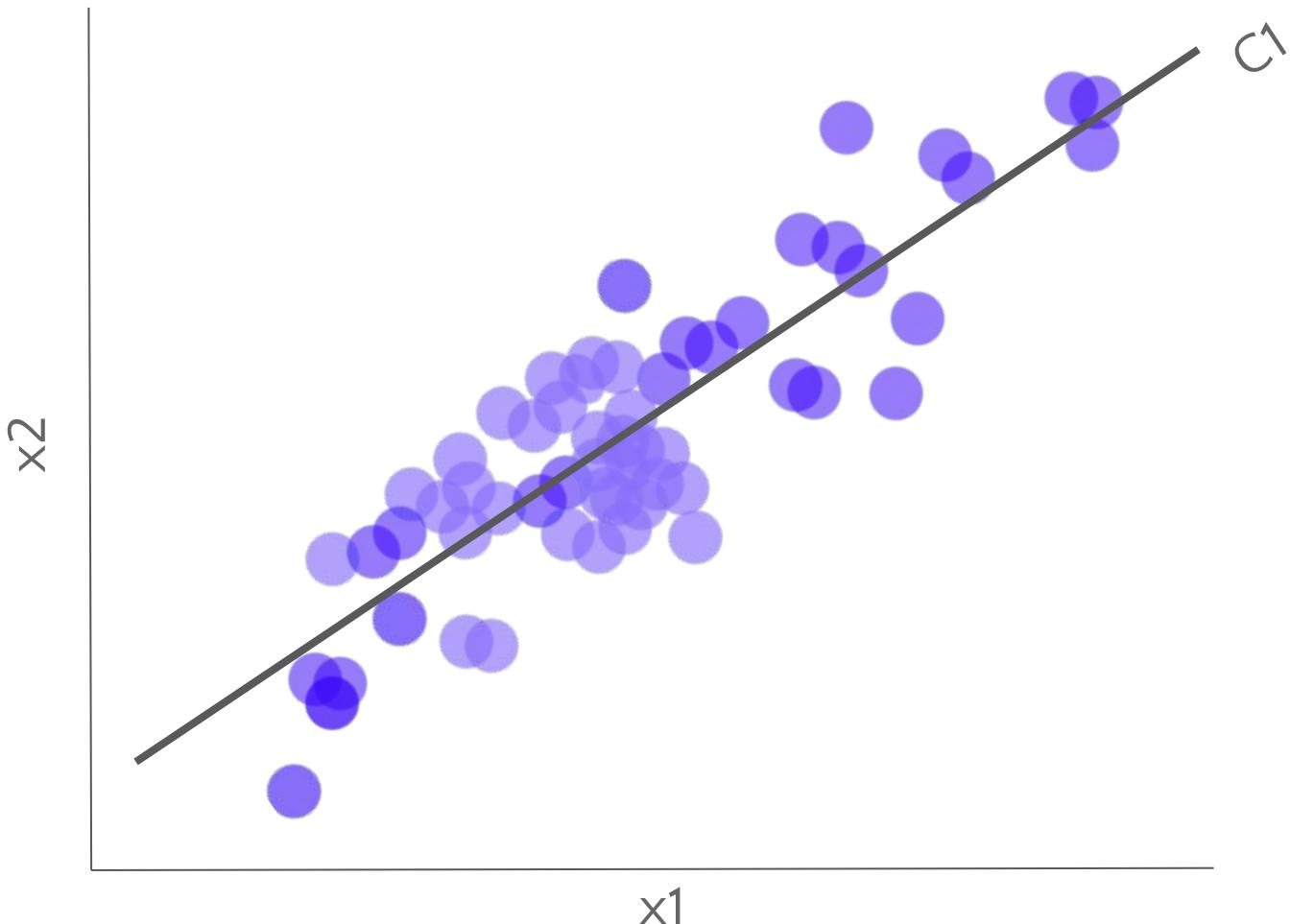


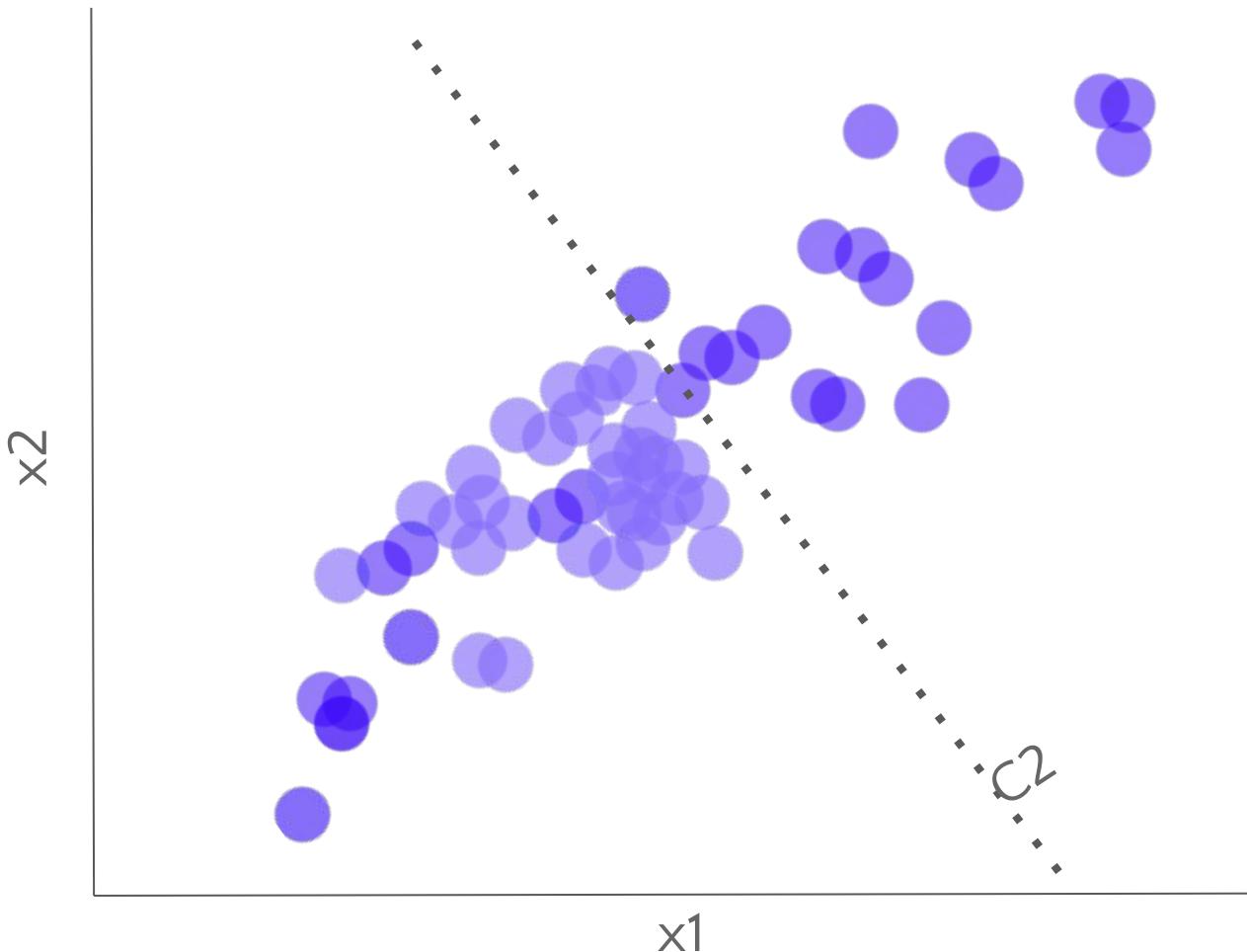


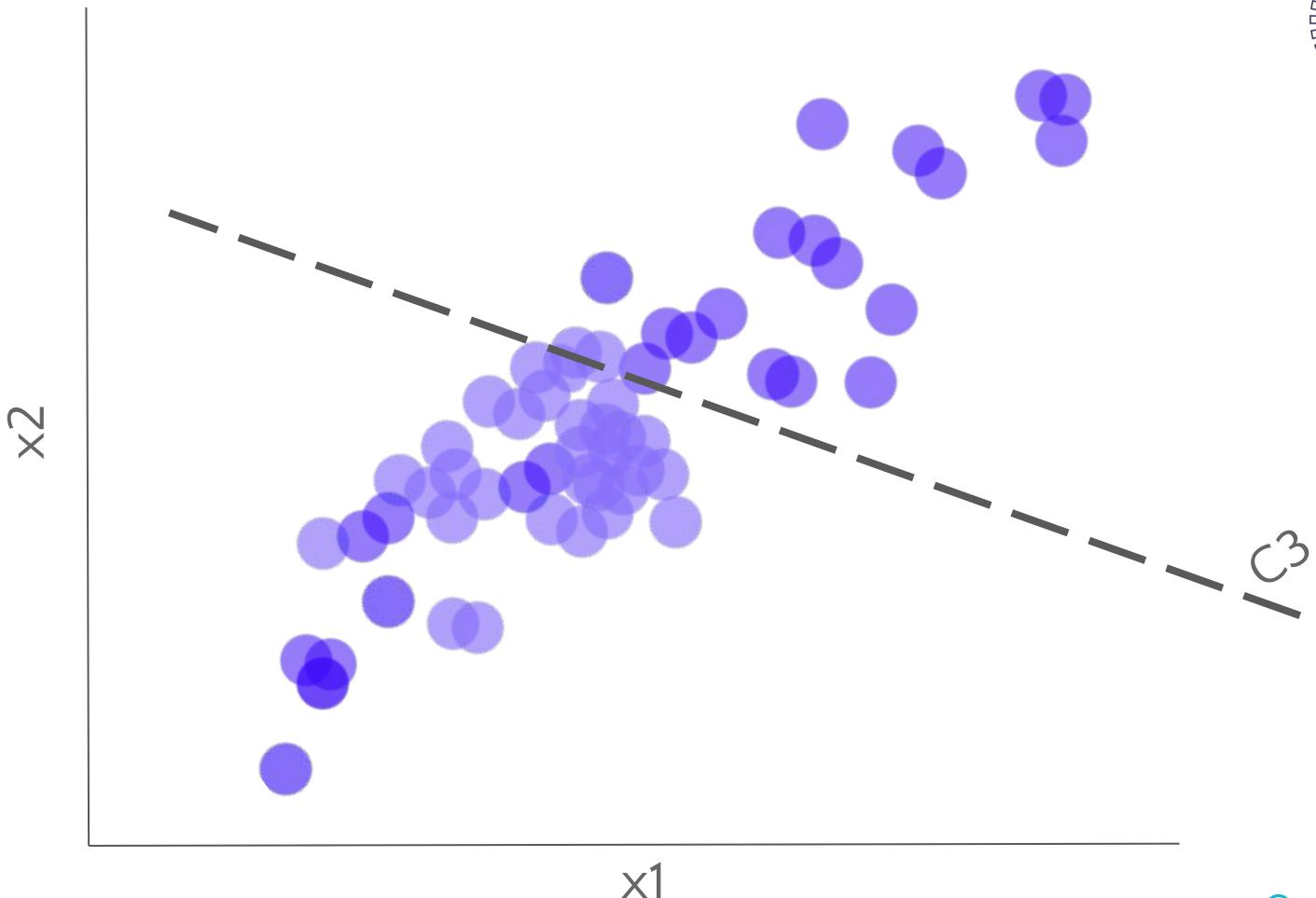


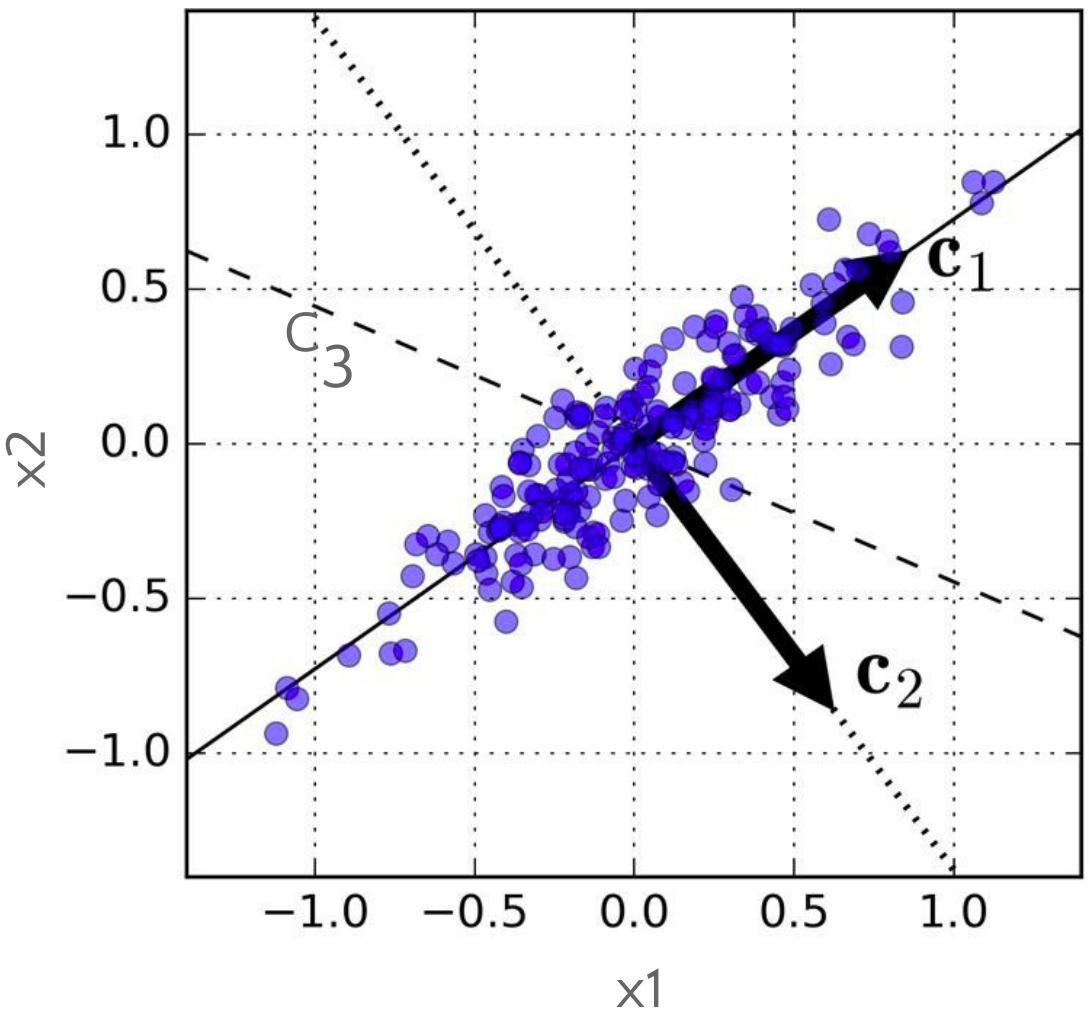


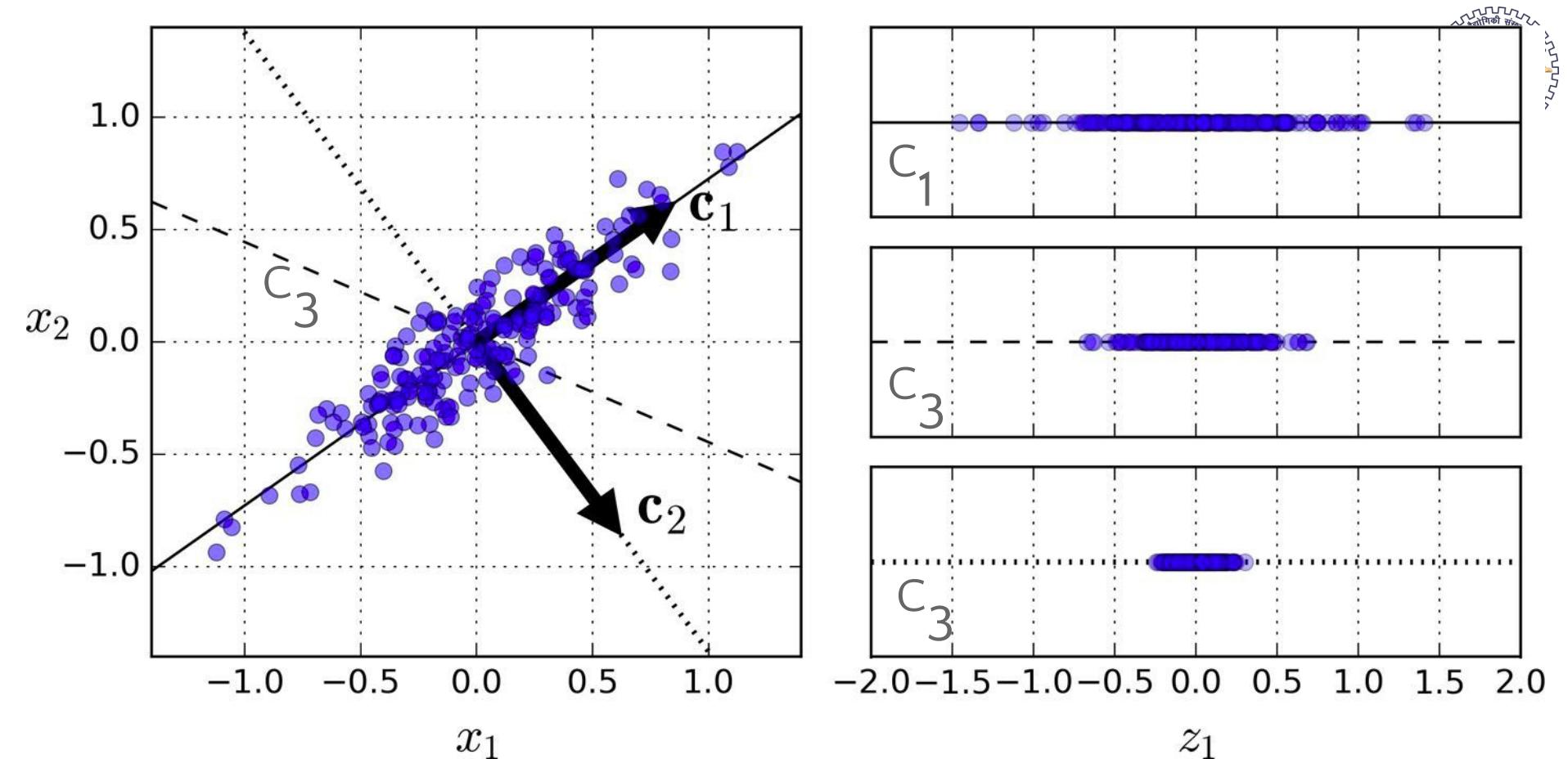


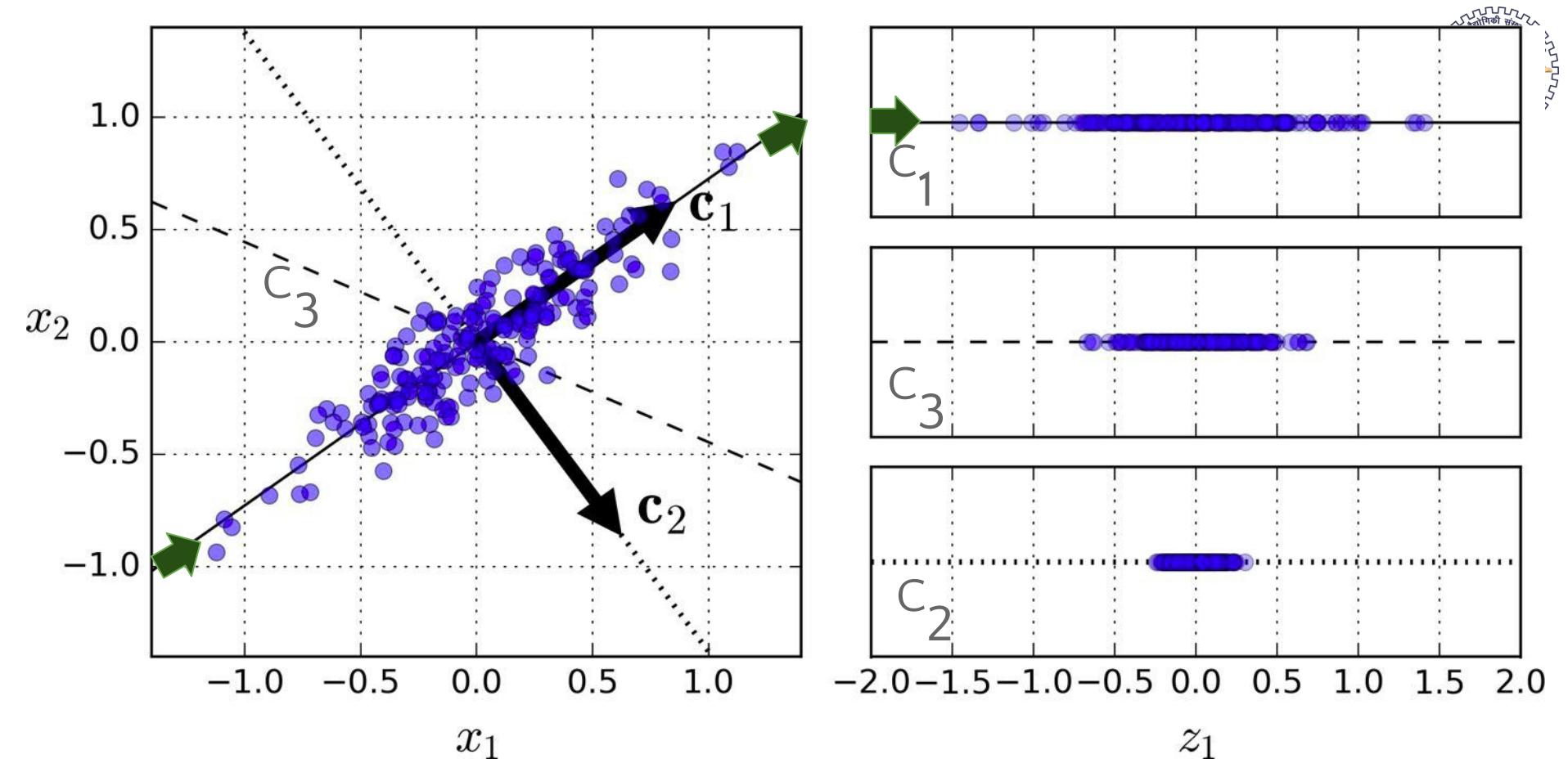


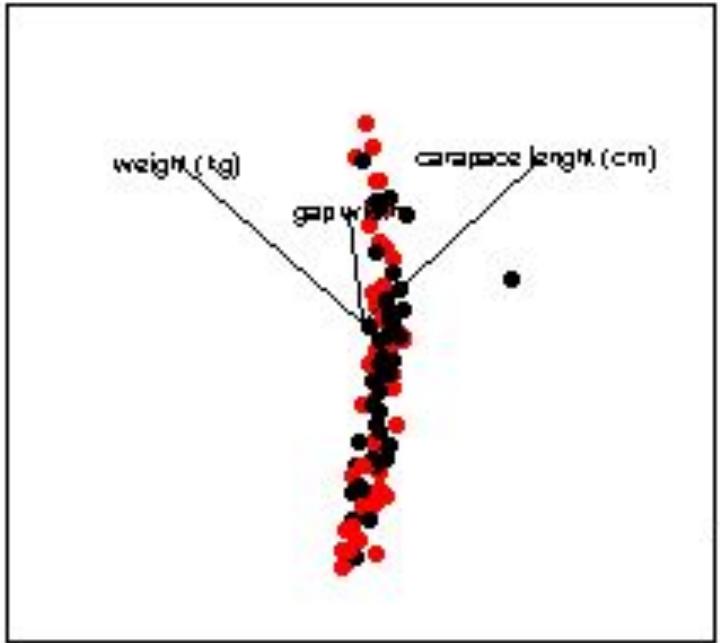








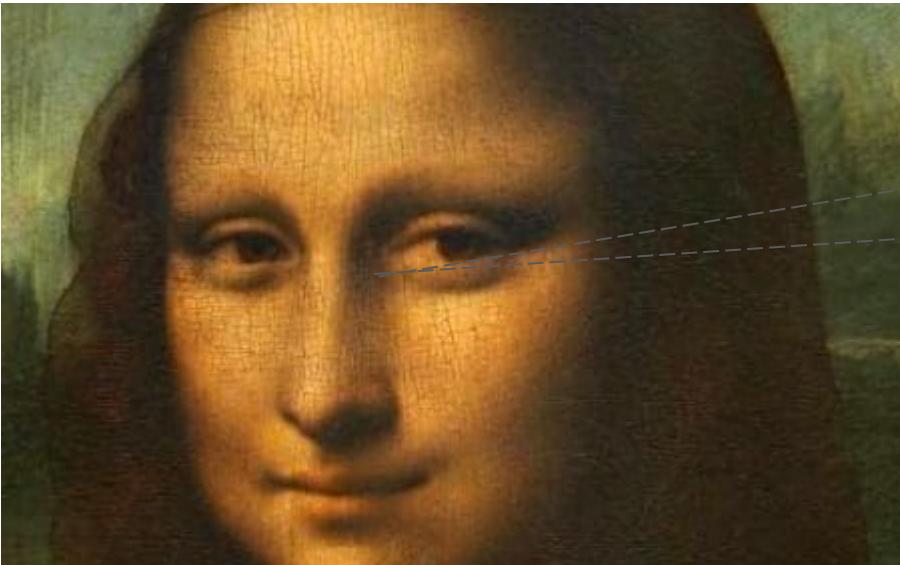








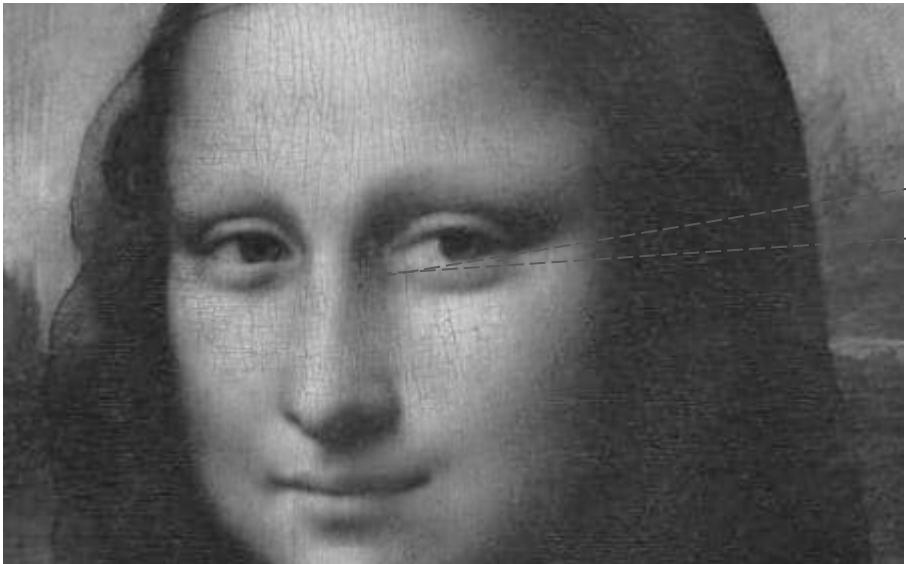
RED	GREEN	BLUE
231	34	122



(231, 34, 122)



RED	GREEN	BLUE
231	34	122



(129)

$$\frac{231 + 34 + 122}{3} = 129$$



Dimensionality reduction - summing up





Dimensionality reduction – summing up

Reduced Data





Dimensionality reduction – summing up

Reduced Data

Lesser Resources





Dimensionality reduction - summing up

Reduced Data

Lesser Resources

Shorter Time



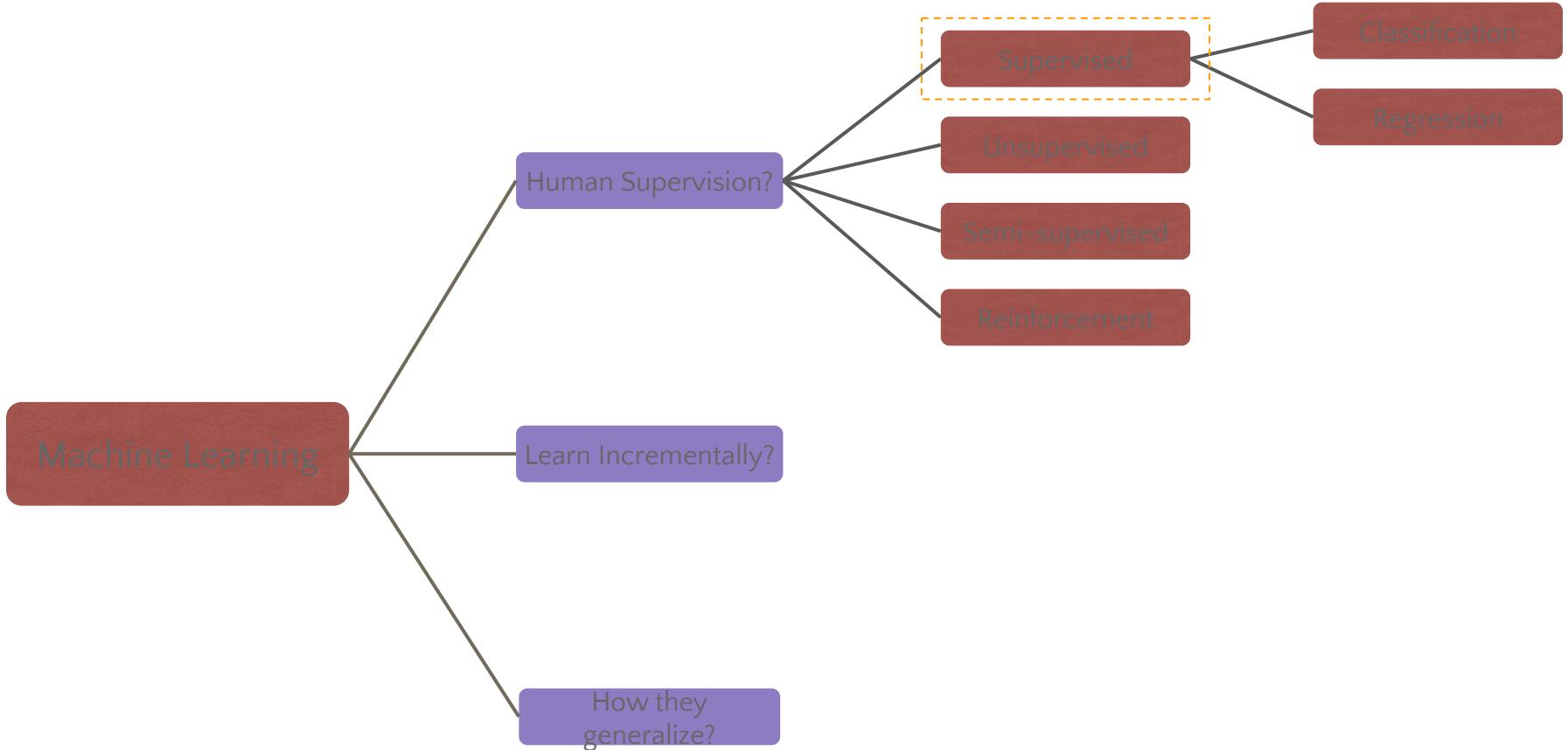


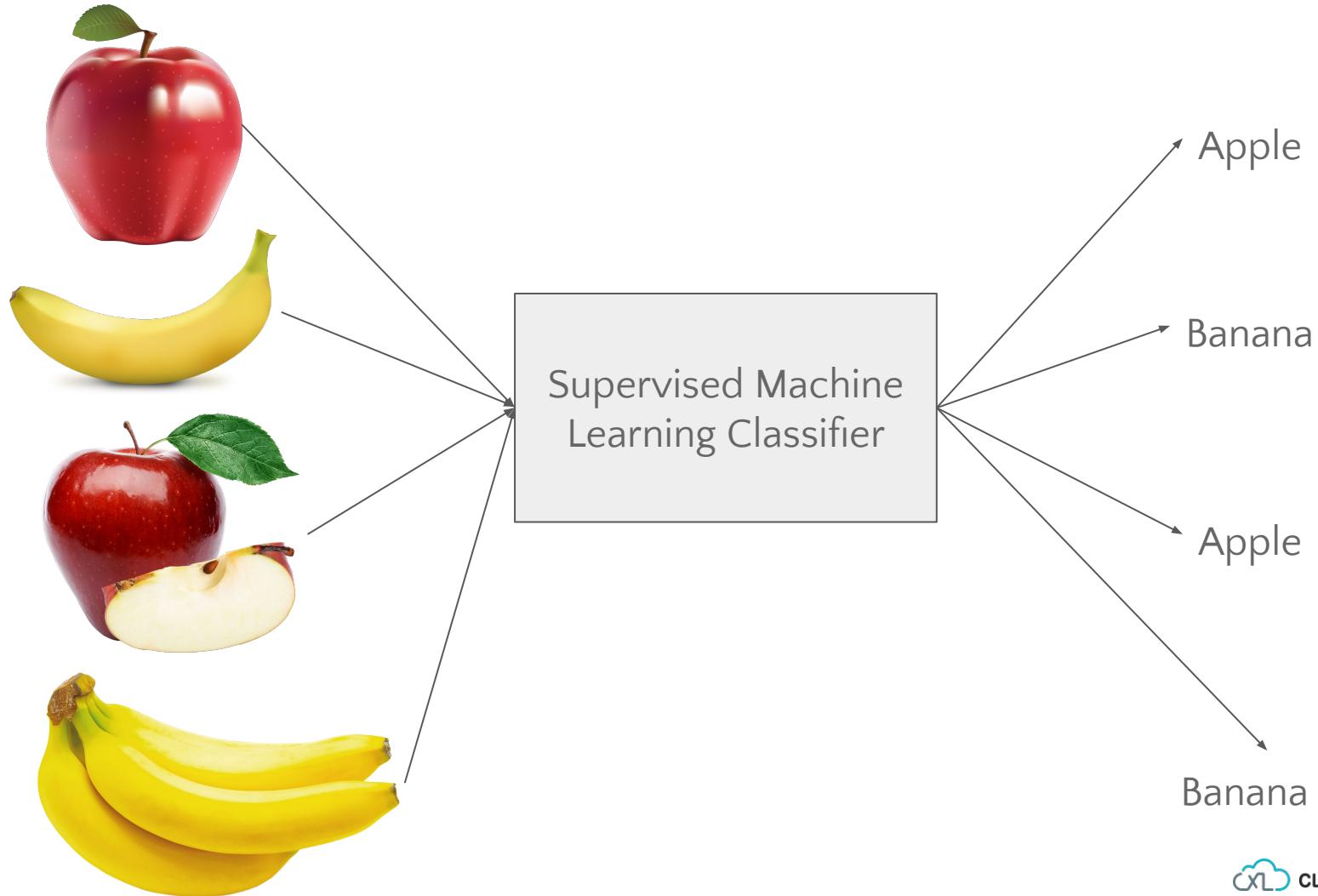
Unsupervised Machine Learning



What is Supervised Machine Learning?

- Recap from what we have learnt





What are unsupervised Algorithms?





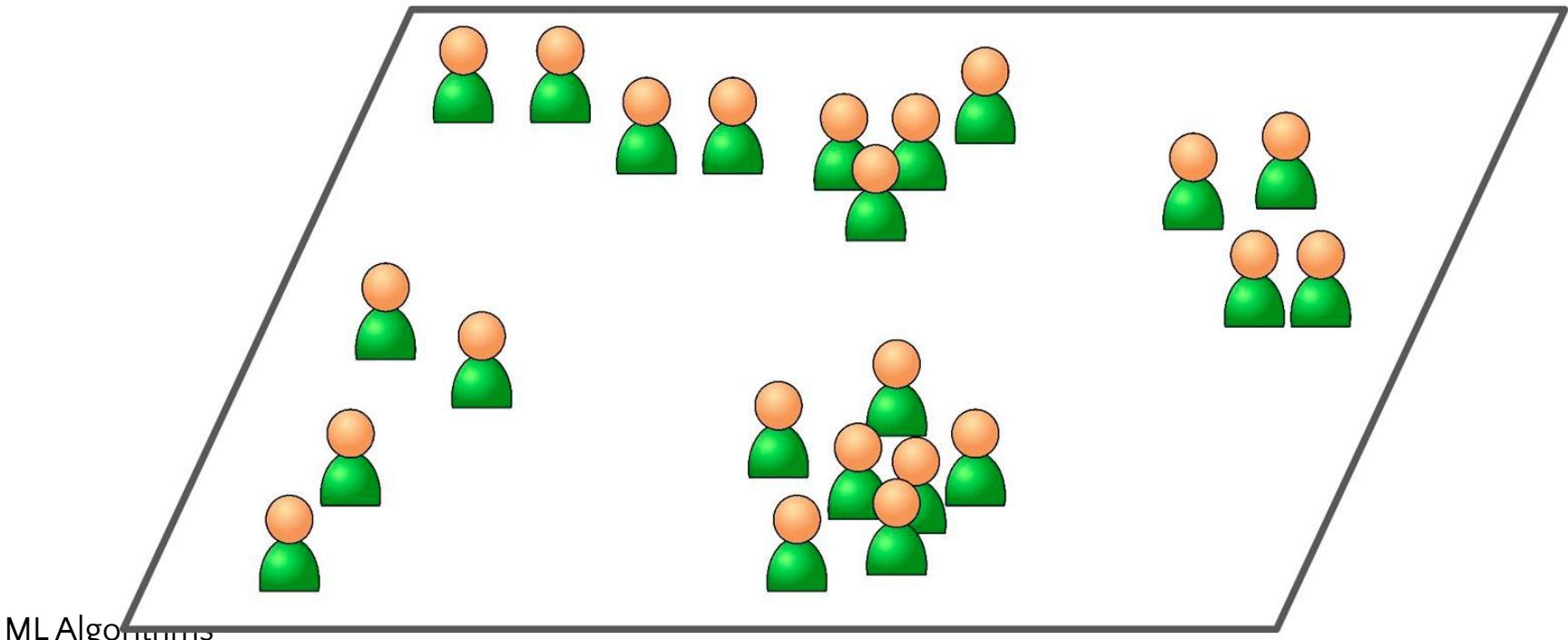
Where are unsupervised Algorithms used?

1. Finding Patterns in User behaviour
2. Anomaly Detection
3. Reducing Data - dimensionality reduction.

Where are unsupervised Algorithms used ?

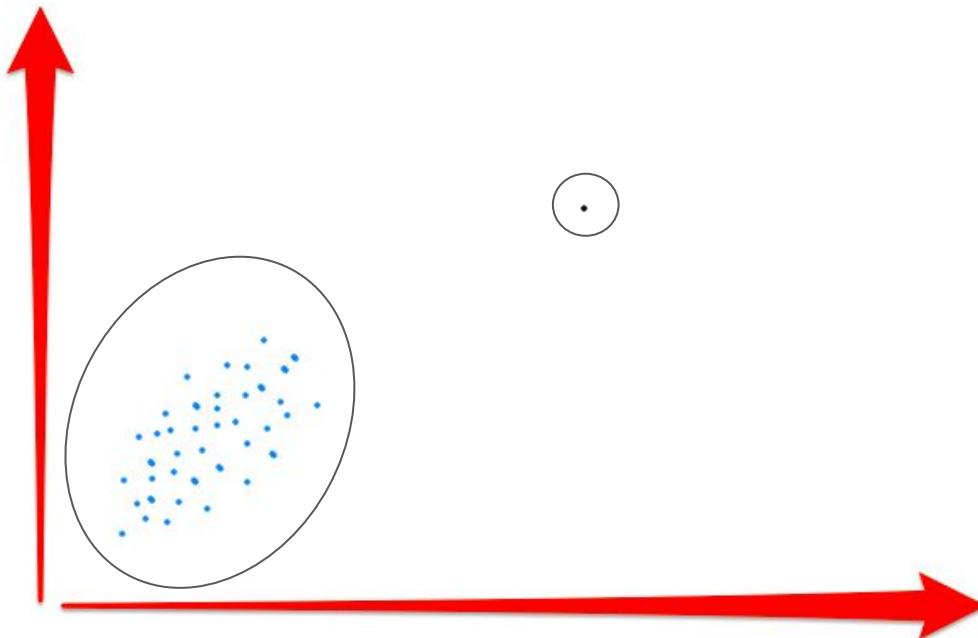
1. Finding Patterns in User behaviour

Training set



Where are unsupervised Algorithms used ?

2. Anomaly Detection



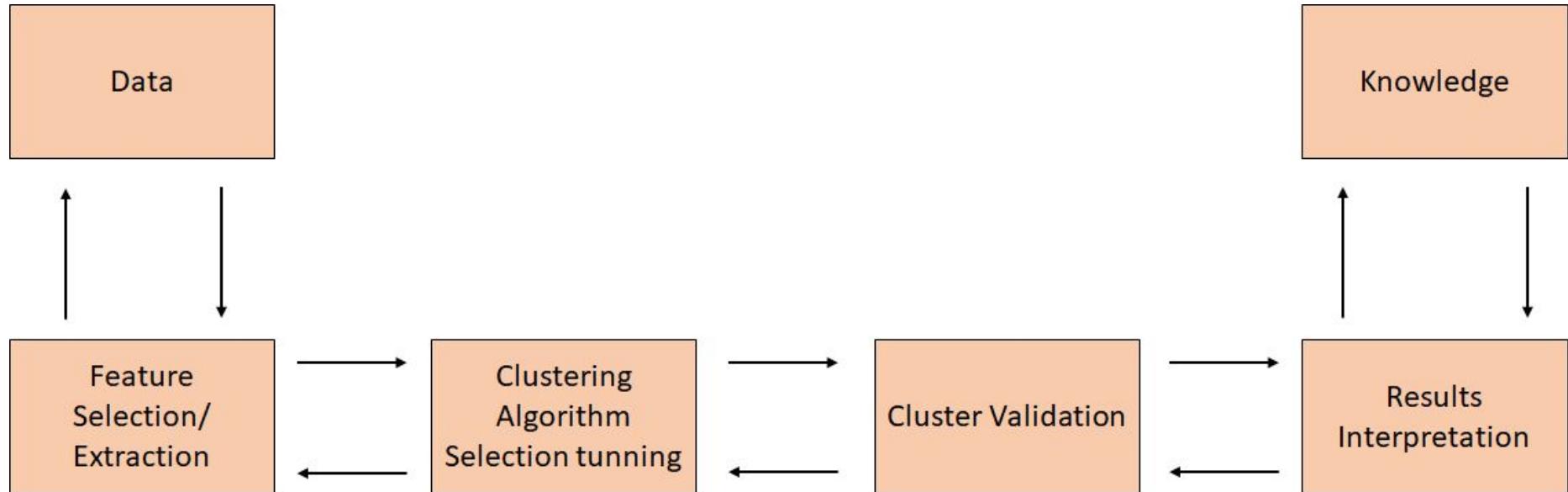
Where are unsupervised Algorithms used ?

3. Reducing Data - dimensionality reduction.





Unsupervised Learning Analysis Process





Main types of unsupervised learning problems:

- Clustering
- Dimensionality Reduction



How to find patterns in user behaviour?

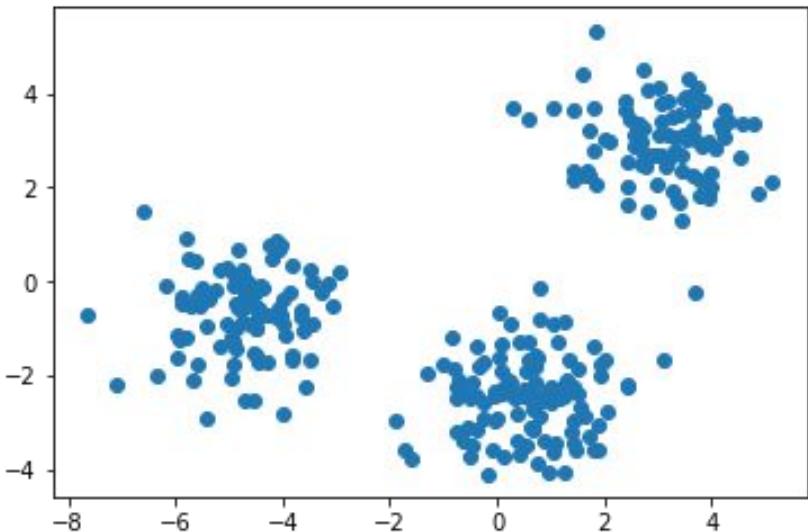
Use clustering algorithm such as KMeans



How does KMeans Work?

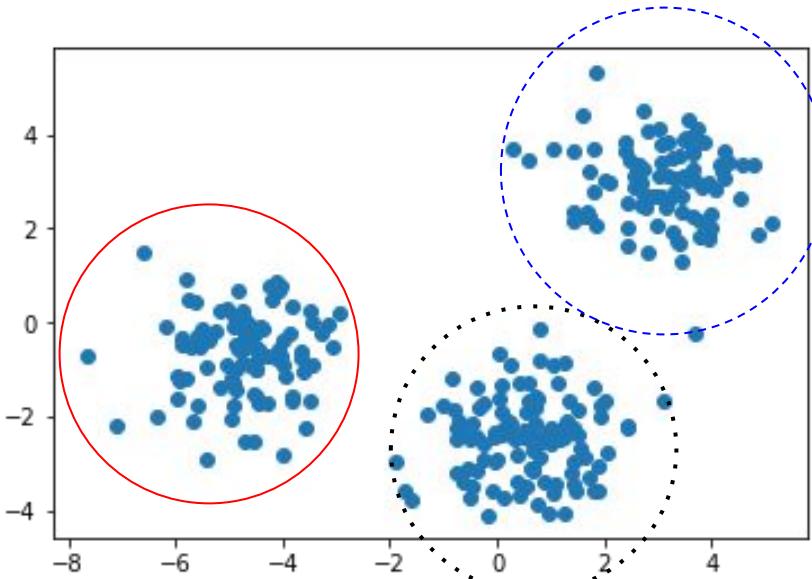
Clustering - KMeans

Let see clusters



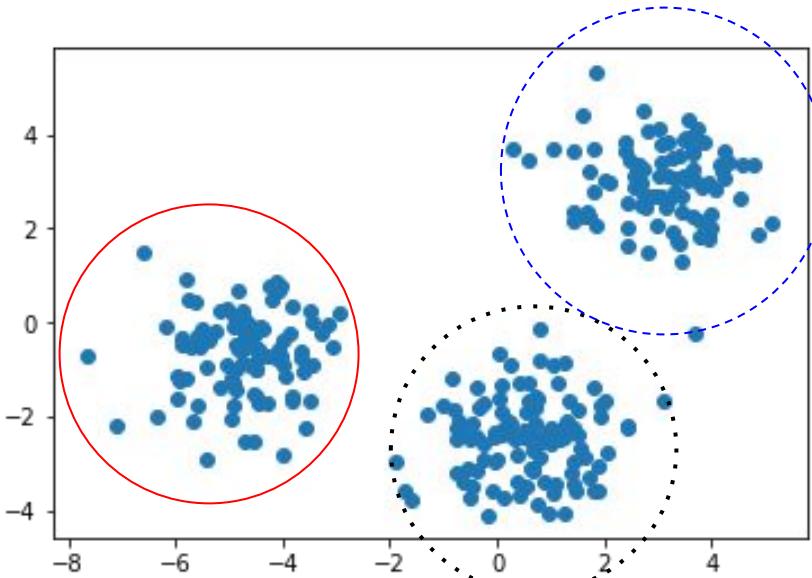
Clustering - KMeans

Let see clusters



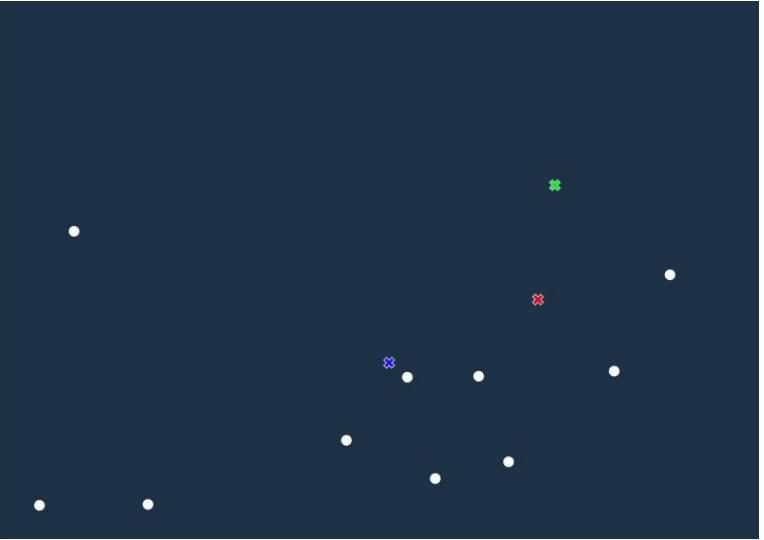
Clustering - KMeans

Let see clusters



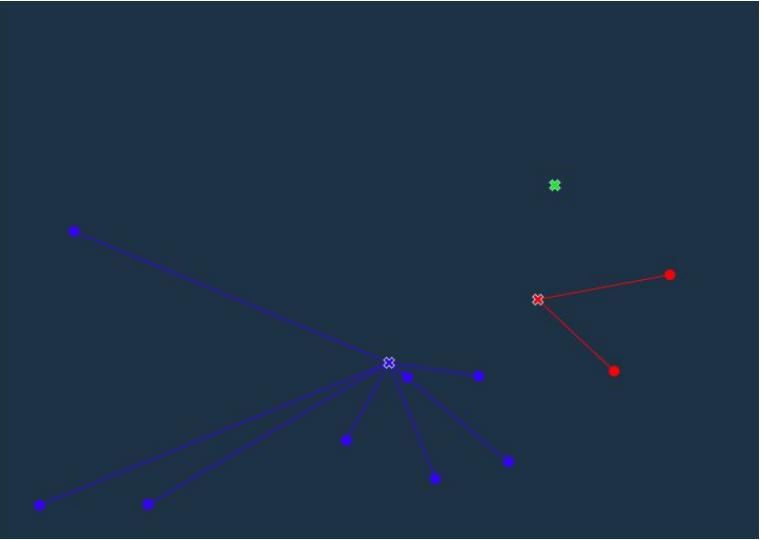
Clustering - KMeans

Initialize - 10 samples and 3 centroids

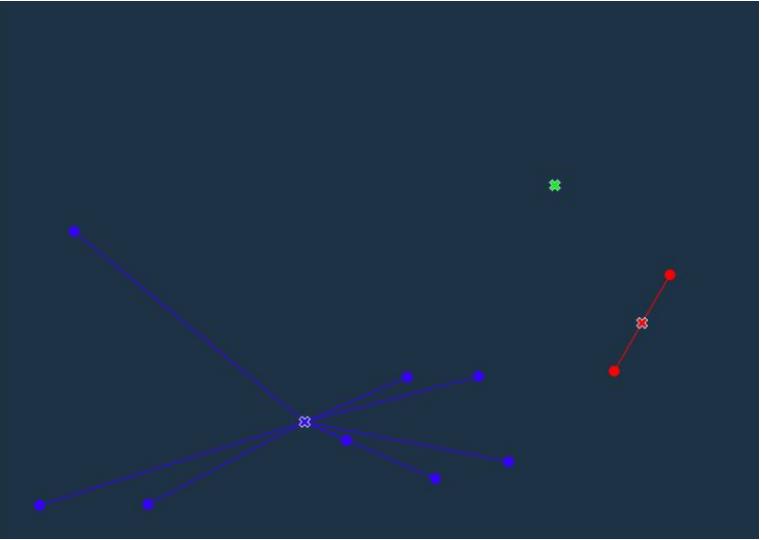


<http://tech.nitoyon.com/en/blog/2013/11/07/k-means/>

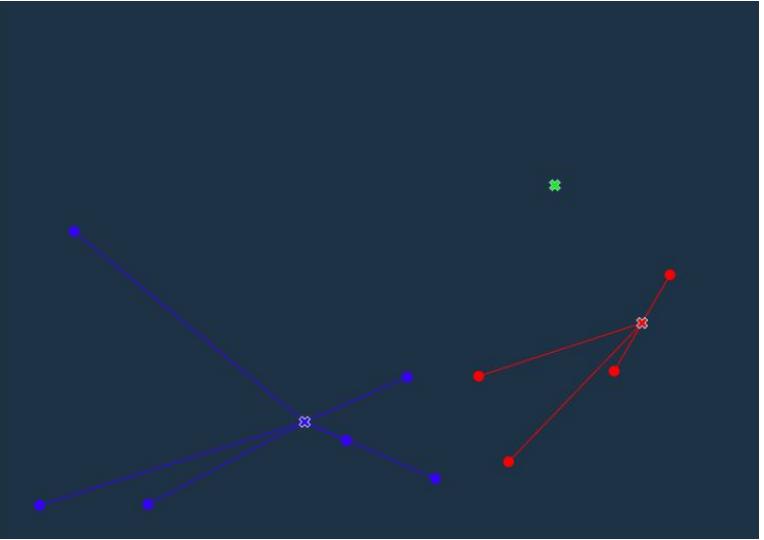
Clustering - KMeans



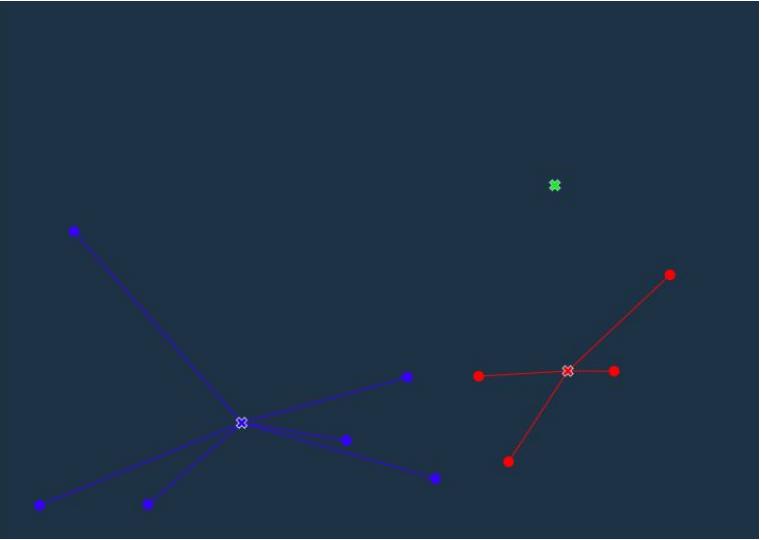
Clustering - KMeans



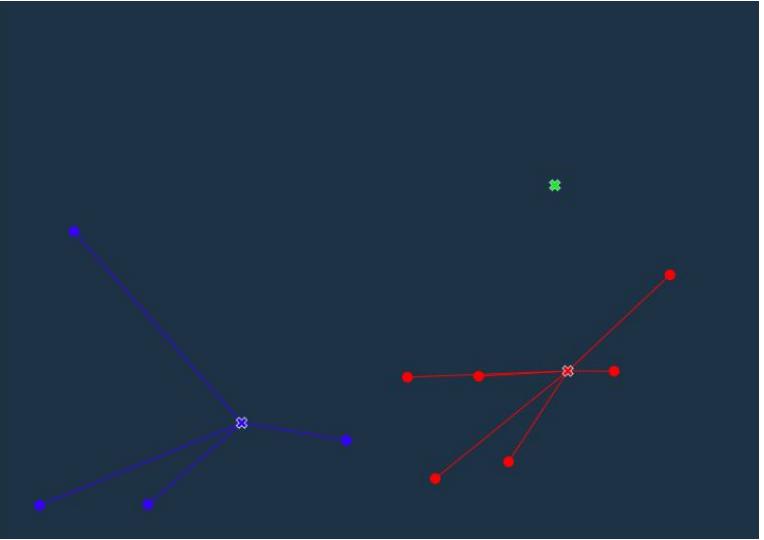
Clustering - KMeans



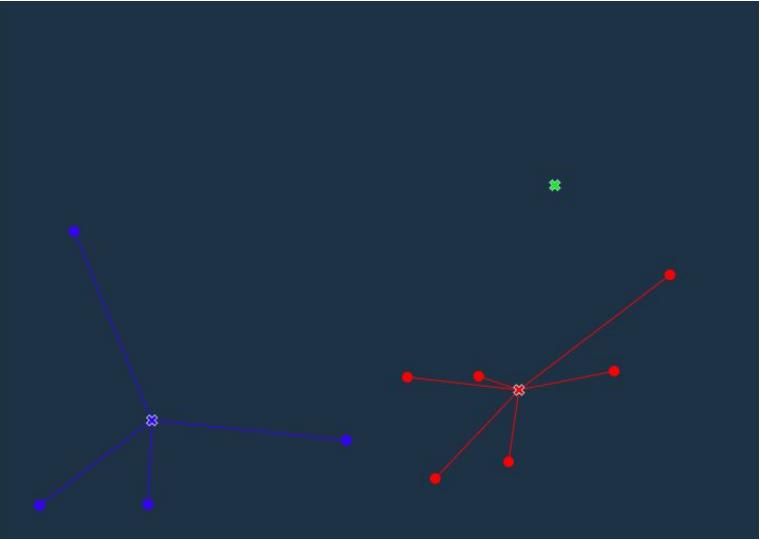
Clustering - KMeans



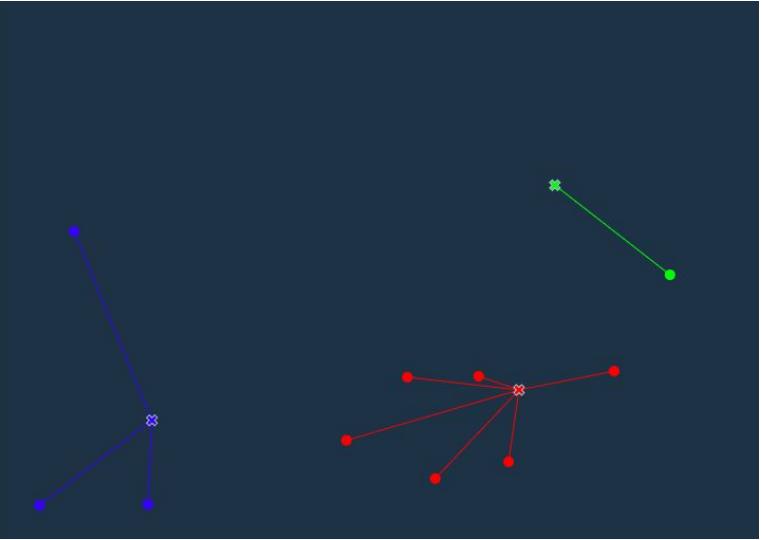
Clustering - KMeans



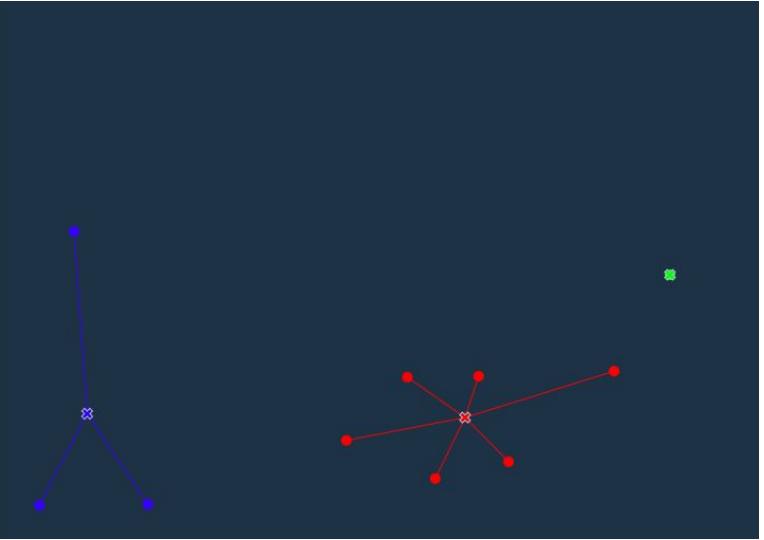
Clustering - KMeans



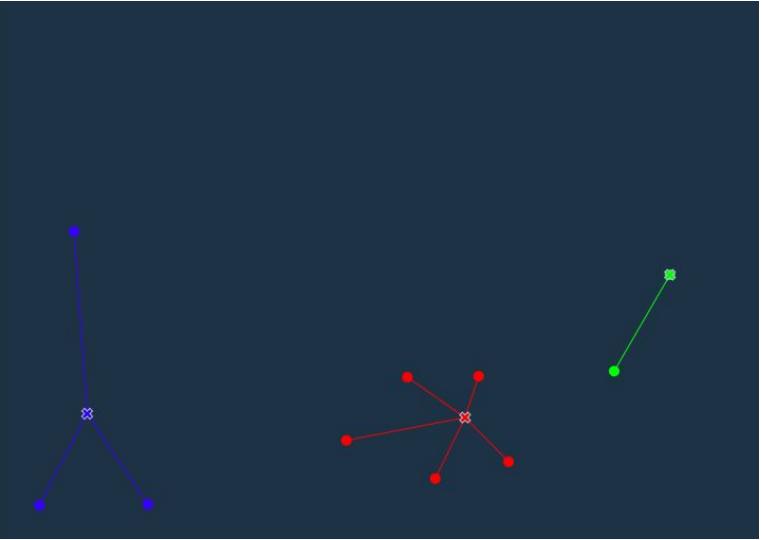
Clustering - KMeans



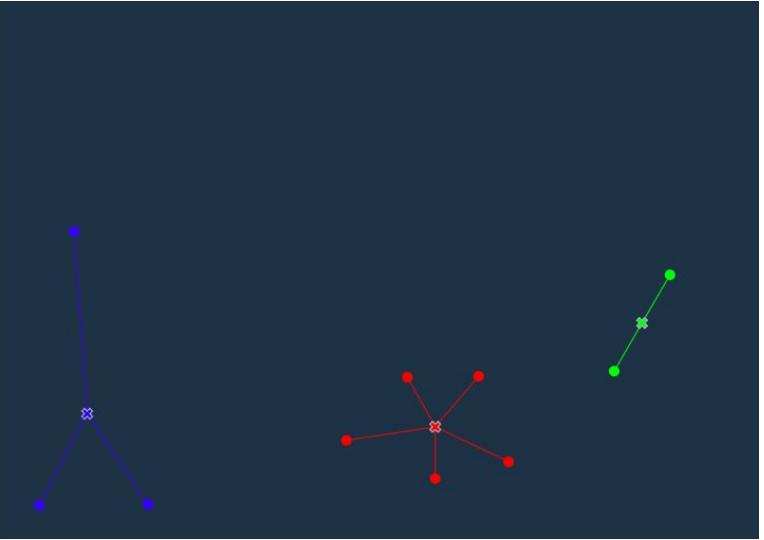
Clustering - KMeans



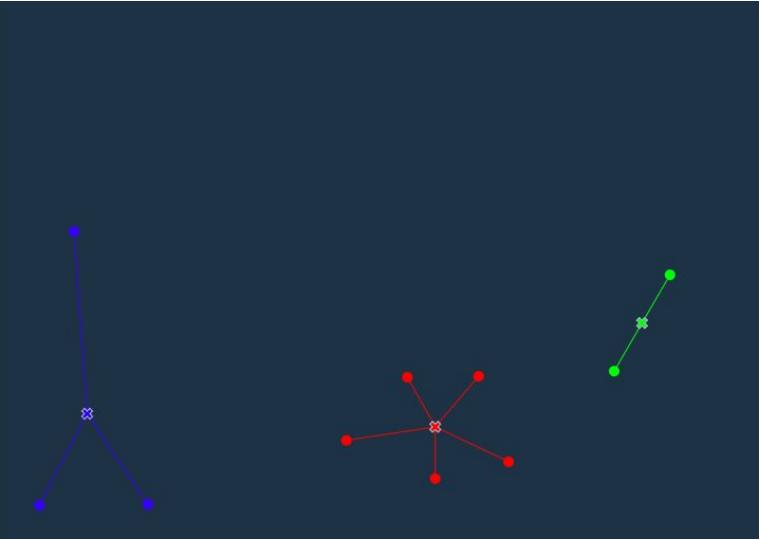
Clustering - KMeans



Clustering - KMeans



Clustering - KMeans





KMeans

How many groups should we have?



KMeans

How many groups should we have?

- Field knowledge
- Business decision
- Elbow Method



KMeans

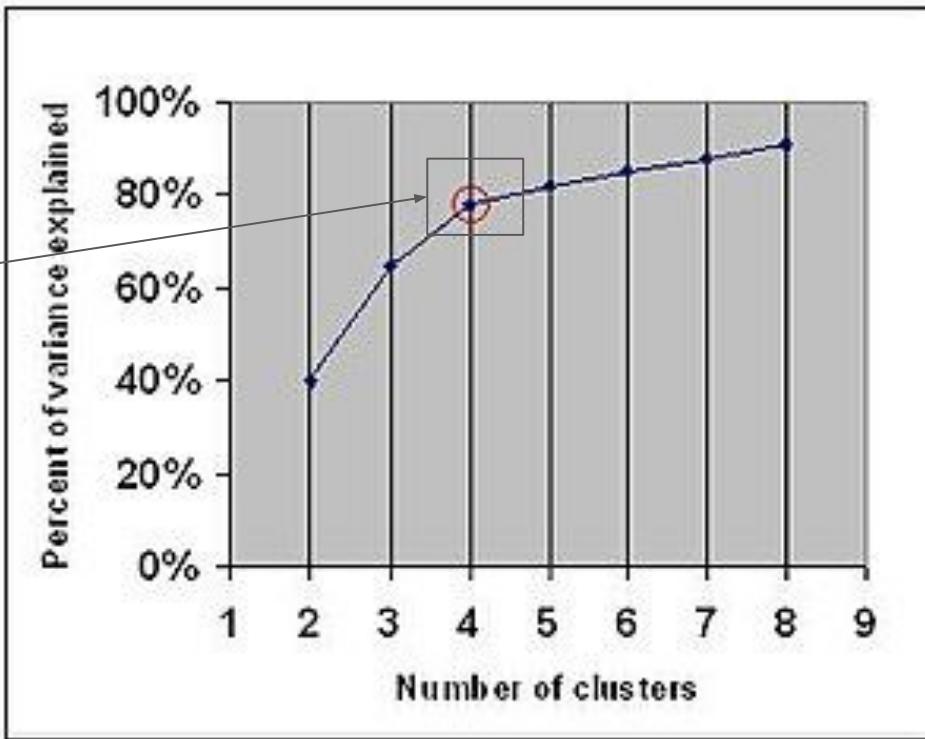
Percentage of variance explained?

$$\% \text{ Variance} = \frac{\text{Variance between groups}}{\text{Total variance}}$$

Clustering - KMeans

Evaluating a clustering

Elbow.





Clustering - KMeans

Check the code of KMeans in Notebook



Questions?

<https://discuss.cloudxlab.com>

reachus@cloudxlab.com