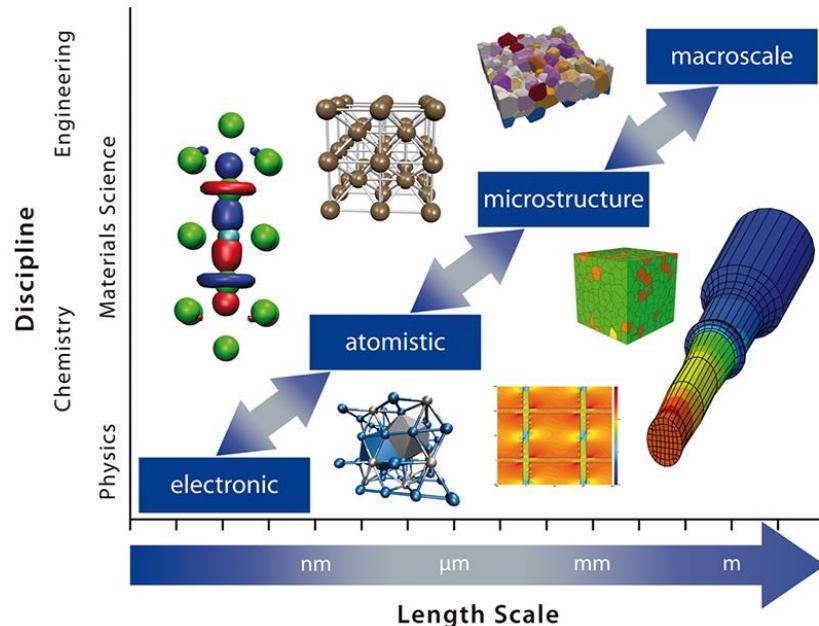


MLL213: Materials Modelling

Prof N S Harsha Gunda

Department of Materials Science and Engineering
IIT Delhi



Machine Learning Algorithms

Office: 306H, Academic Complex West

Email: gnsharsha@iitd.ac.in

Cost Function

- A cost function is a mathematical function that quantifies the error between the predicted outcomes of a machine learning model and the actual outcomes.
- ML models try to reduce the cost function during training.

Types of cost function

1. Mean Squared Error

- Typically used for regression tasks
- Penalizes large errors

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

2. Mean Absolute Error

- Does not square the values
- Does not penalize large errors.

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

Types of cost function

3. Cross Entropy or Log loss

- Used for classification, like logistic regression
- Log Loss penalizes confident and incorrect predictions

$$\text{Log Loss} = -\frac{1}{n} \sum_{i=1}^n [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

4. Hinge loss

- Used in SVM regression
- Works well with margin based algorithms

$$\text{Hinge Loss} = \frac{1}{n} \sum_{i=1}^n \max(0, 1 - y_i \cdot \hat{y}_i)$$

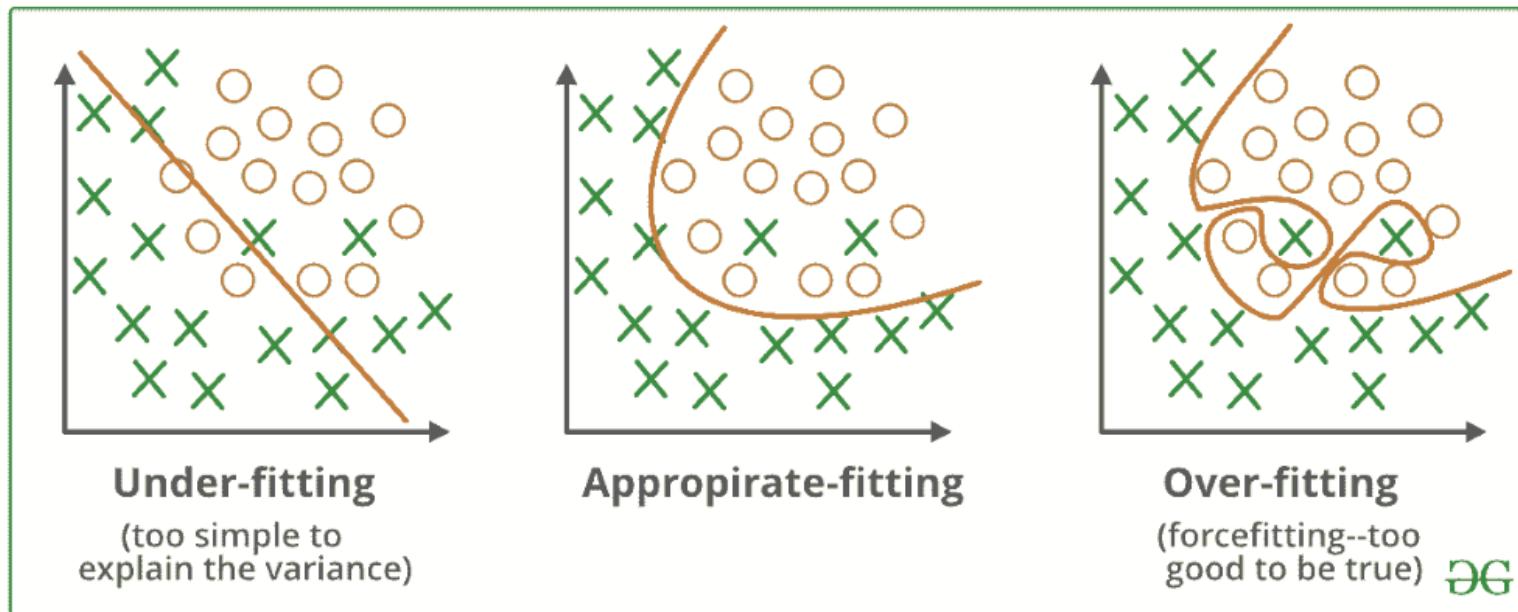
4. Huber loss

- Used for a more robust regression
- Compromise between MAE and MSE

$$L_\delta(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{for } |y - \hat{y}| \leq \delta \\ \delta \cdot (|y - \hat{y}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

Regularization

- It is very important to control the complexity of the model to avoid under- or overfitting the data
- Regularization is typically used to find the sweet spot where model is perfectly optimized
- Regularization can simplify models, reduces multicollinearity, allows fine tuning and provides consistency



How do we regularize linear regression?

Ridge vs Lasso Regression

λ is the regularization parameter

Regression Algorithms

- Ridge regression (L2)

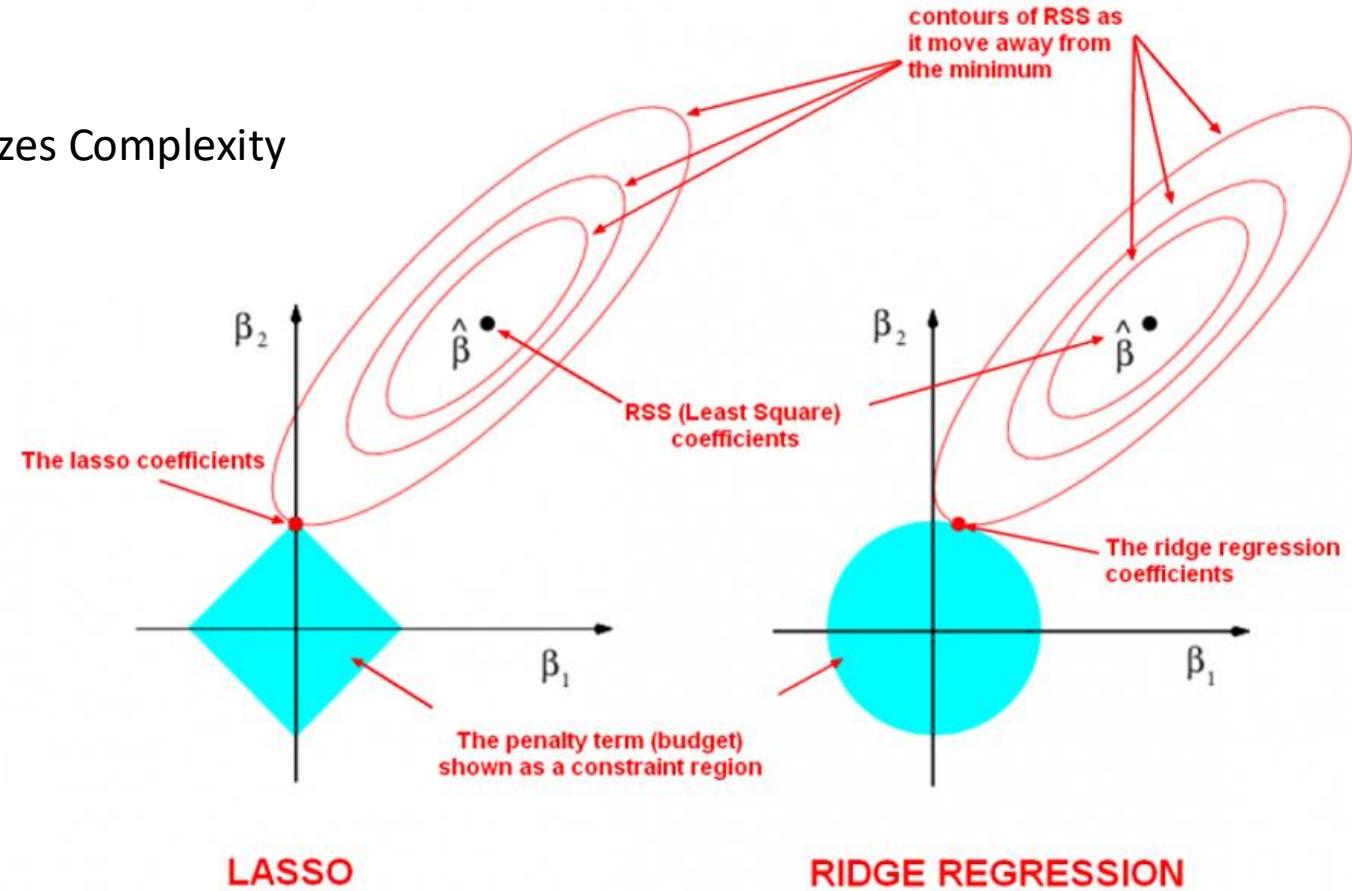
$$\sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij})^2 + \lambda \sum_{j=1}^p \beta_j^2$$

Minimizes Complexity

- Lasso regression (L1)

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

Automatic feature selection



Elastic Net methods use both L1 and L2 regularization to get optimized models

How does ML optimize the cost function?

- Typically, gradient descent is used to get the weights of the model

Cost Function

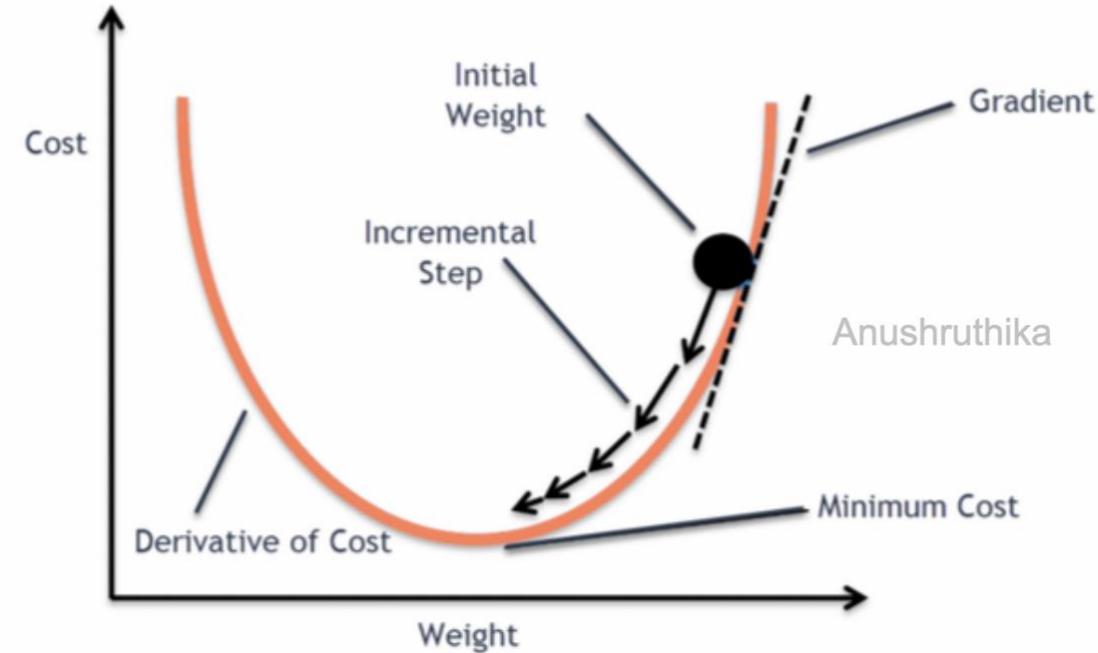
$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^m [h_\Theta(x_i) - y_i]^2$$

↑
Predicted Value ↑
True Value

Gradient Descent

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$$

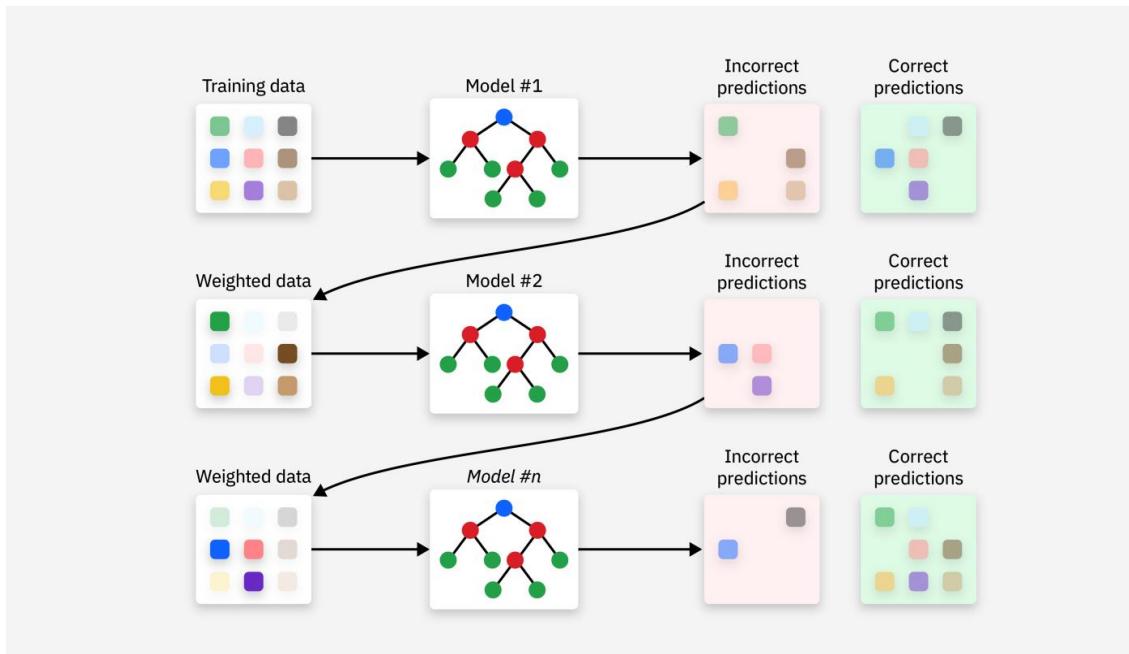
↑
Learning Rate



Learning rate(α) is another hyperparameter that needs to be tuned

Boosting

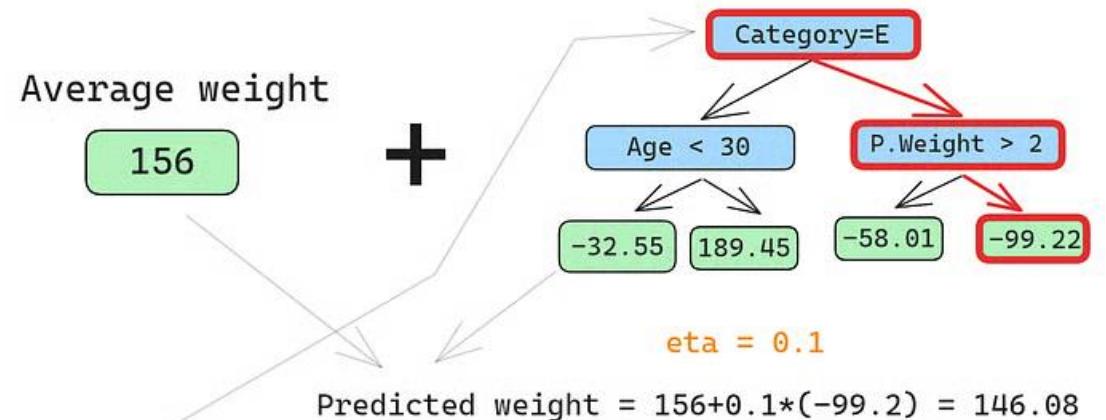
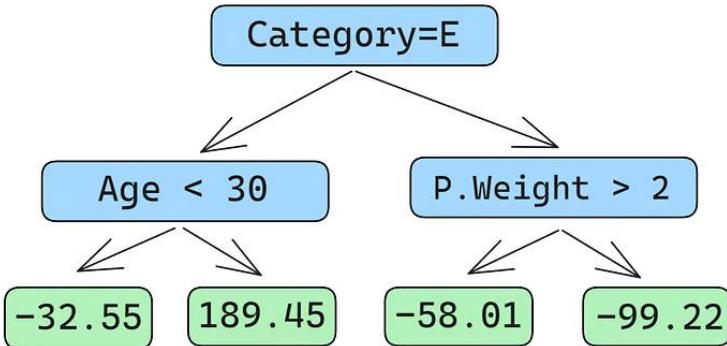
- This is a technique used in ML to improve the accuracy of the training
- By categorically biasing the incorrect prediction in the training data more accurate models are obtained



There is also “bagging” approach where models are trained on a subset of data and combined together stochastically

Gradient Boost algorithm

- Can be used for both regression and classification problems
- XGBoost is a common method used, which is implemented in the sklearn package.
- First makes an initial prediction and calculates the pseudo-residuals
- Builds a “weak learner” decision tree to estimate the residuals.
- Updates the weights based on the learning rate on each row



```
from sklearn import ensemble
```

```
gbr = ensemble.GradientBoostingRegressor(loss='lad', max_depth  
= 10, learning_rate = 0.015, min_samples_split = 50,  
min_samples_leaf = 1, max_features = len(cols), subsample =  
0.9, n_estimators = 300)
```

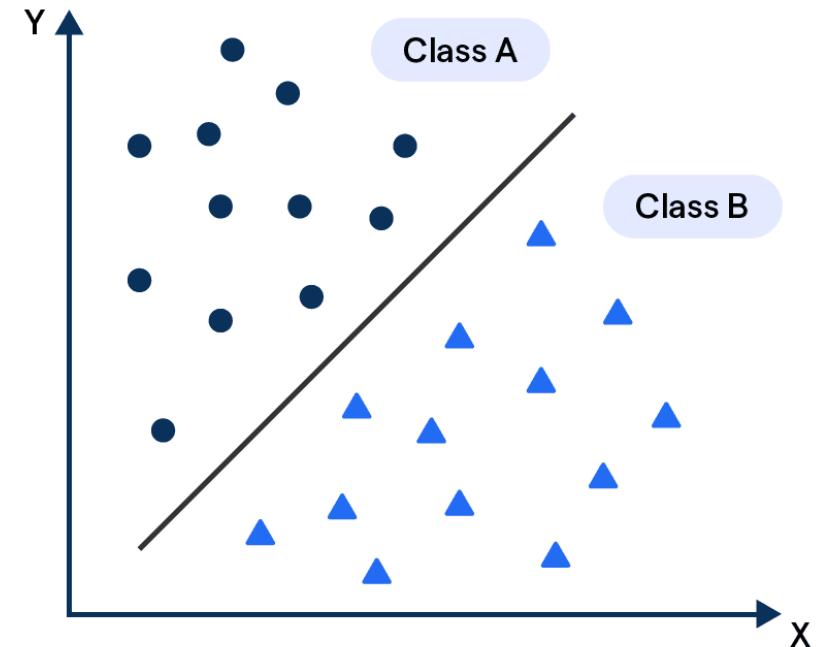
```
gbr.fit(X, y)
```

Other ML algorithms typically used for modeling

Classification Algorithms :

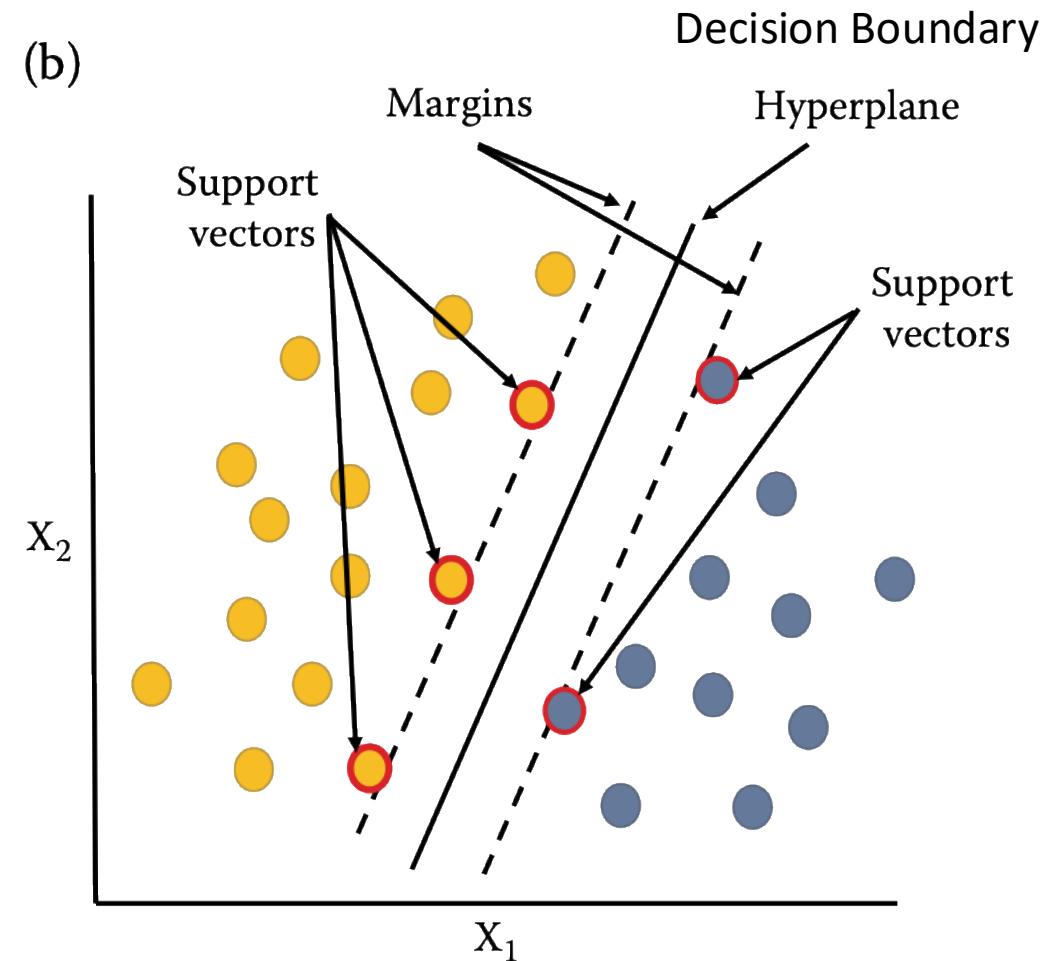
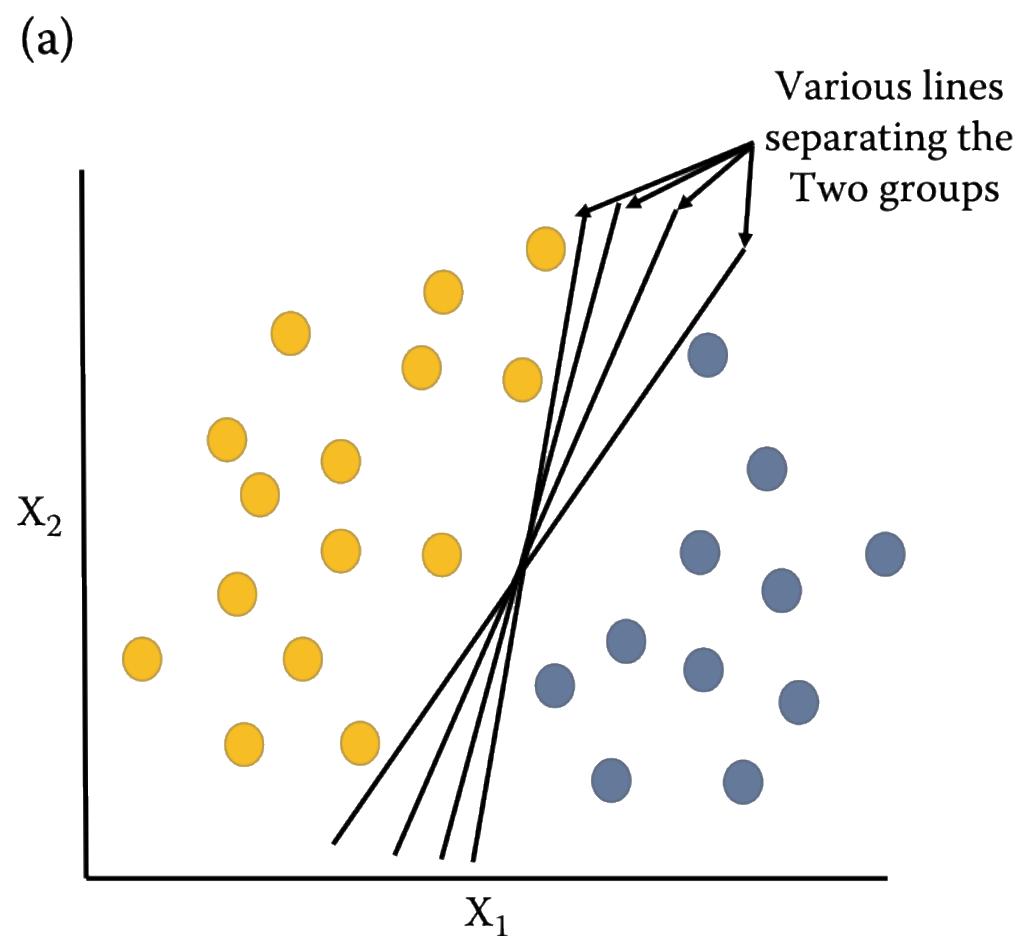
- Support Vector Machines
- kNN
- Random Forest
- Artificial Neural Networks

Classification Algorithm

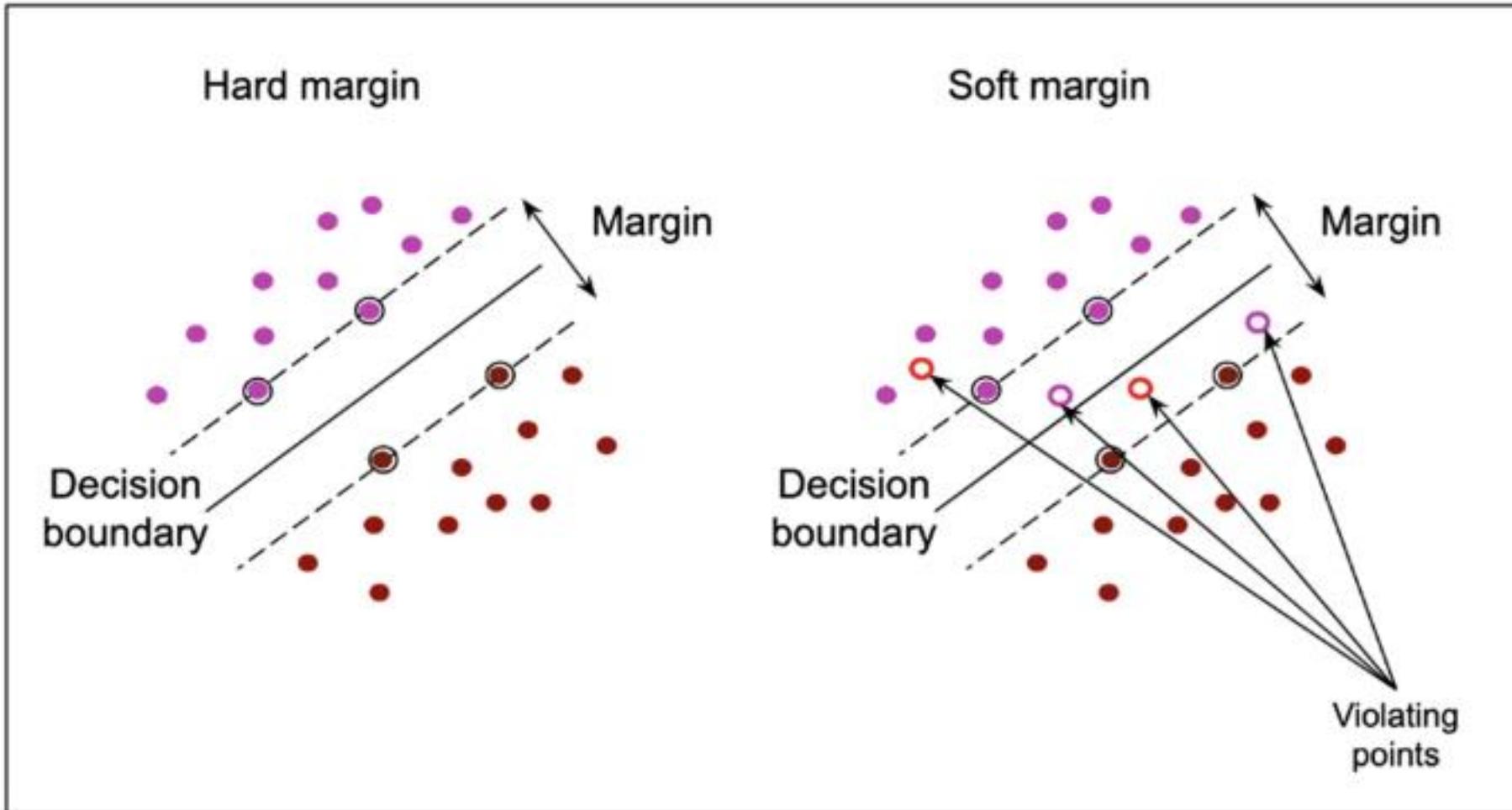


They can be used for Regression as well

Support vector machines



Soft vs Hard Margin SVM



Hyper parameter tuning

- The default setting is usually not that great for the diverse data that we want to model.
- We need to tune the parameters that the model relies on to get more predictive modeling.

Hyper parameters in SVM

Important parameters

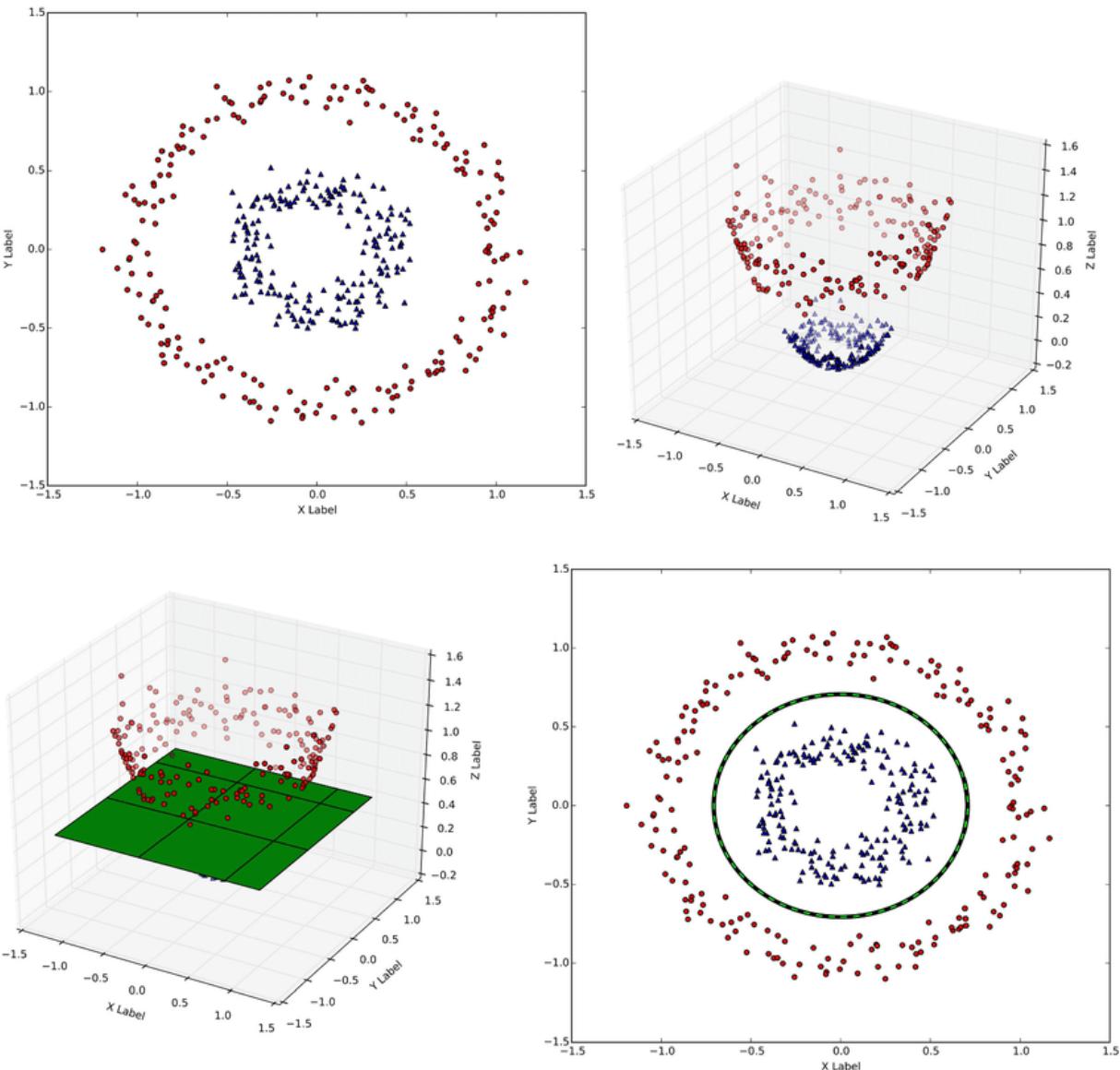
We can use Grid Search or Random search to optimize parameters

```
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

SVR._get_param_names()

['C',
 'cache_size',
 'coef0',
 'degree',
 'epsilon',
 'gamma',
 'kernel',
 'max_iter',
 'shrinking',
 'tol',
 'verbose']
```

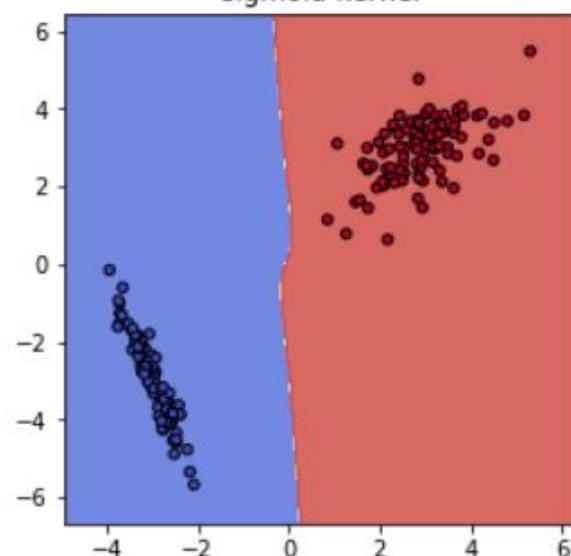
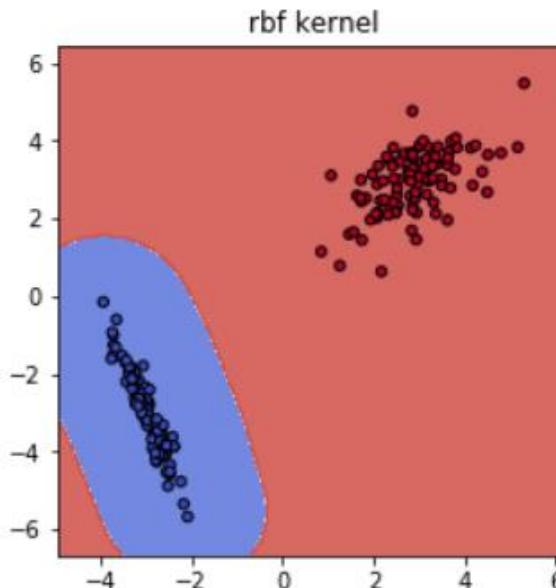
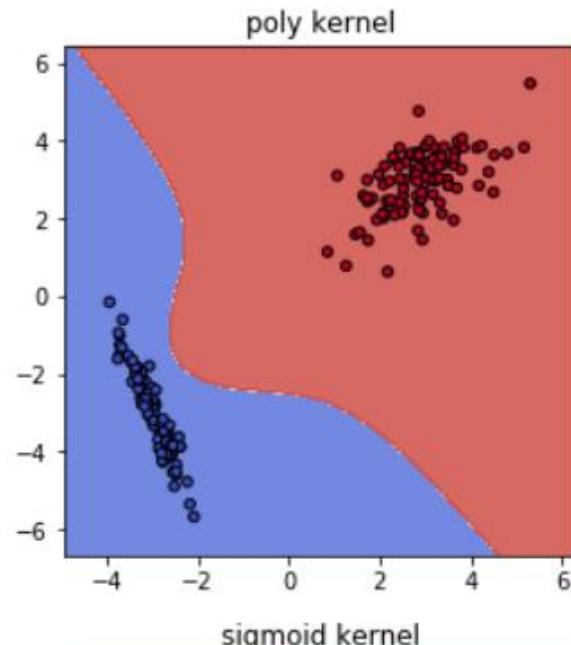
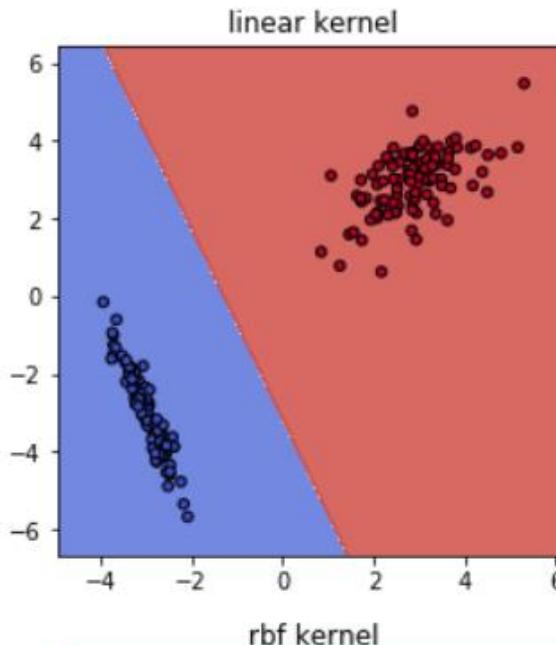
The Kernel Trick



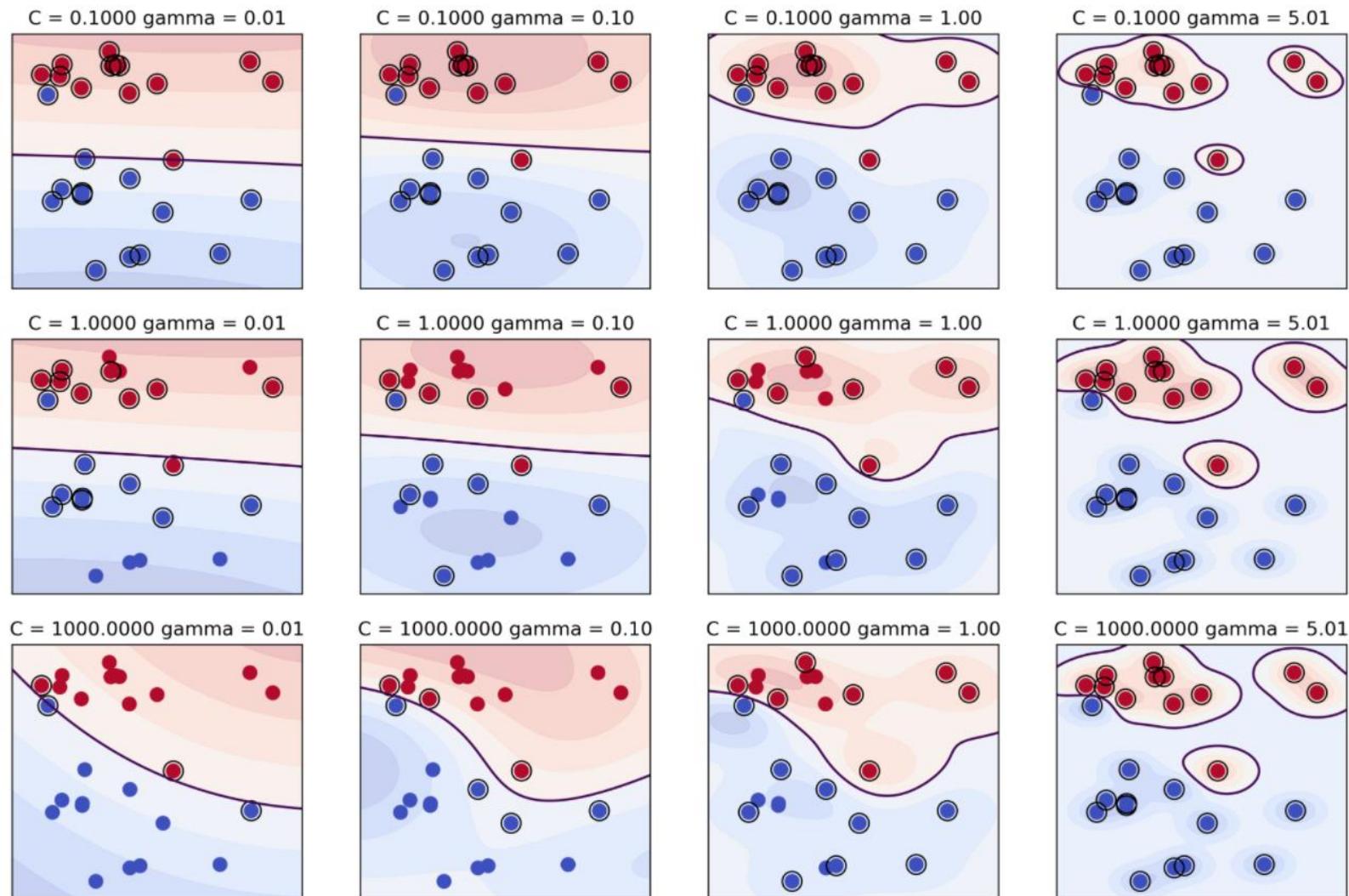
Kernels in SVM

Kernel function	Mathematical formula
Linear kernel	$k(X_i, X) = X_i \cdot X$
Polynomial kernel	$k(X_i, X) = (X_i \cdot X)^e$
Normalized Polynomial kernel	$k(X_i, X) = [(X_i \cdot X)^e] / \sqrt{(X_i \cdot X_i)^e (X \cdot X)^e}$
Radial basis kernel (RBF)	$k(X_i, X) = e^{-(X_i - X ^2 / 2\sigma^2)}$

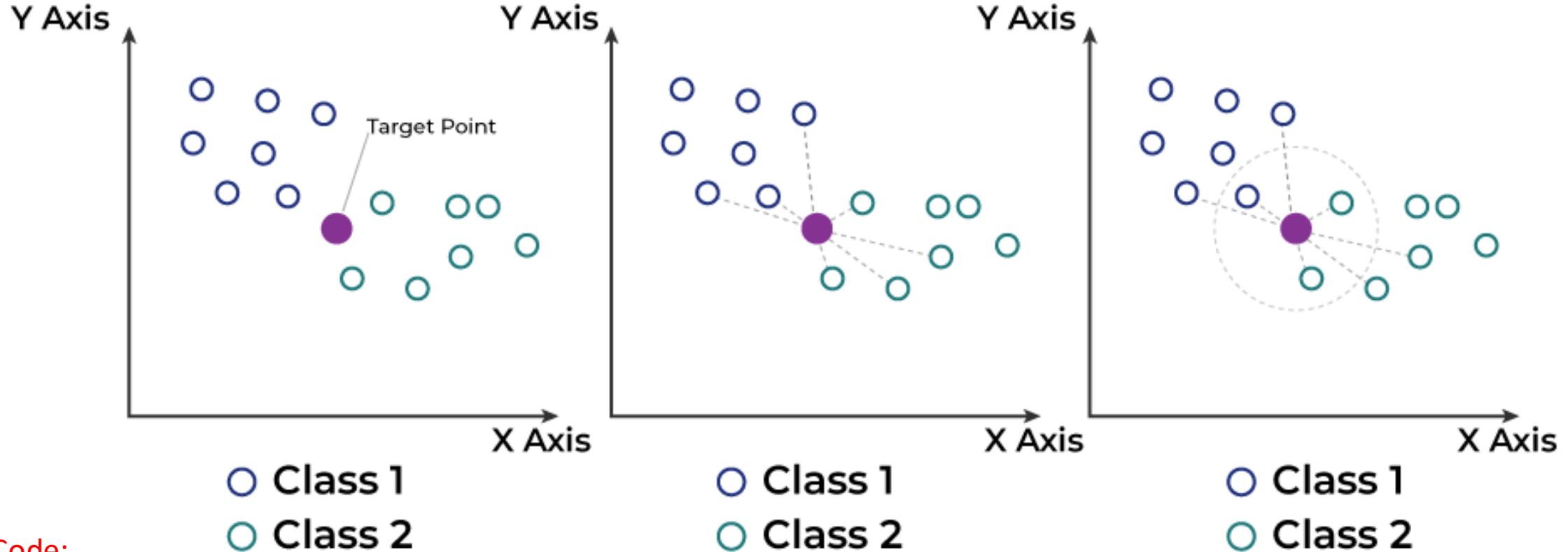
Decision Boundary in Different Kernels



C and gamma parameters in SVM



K-NN classification

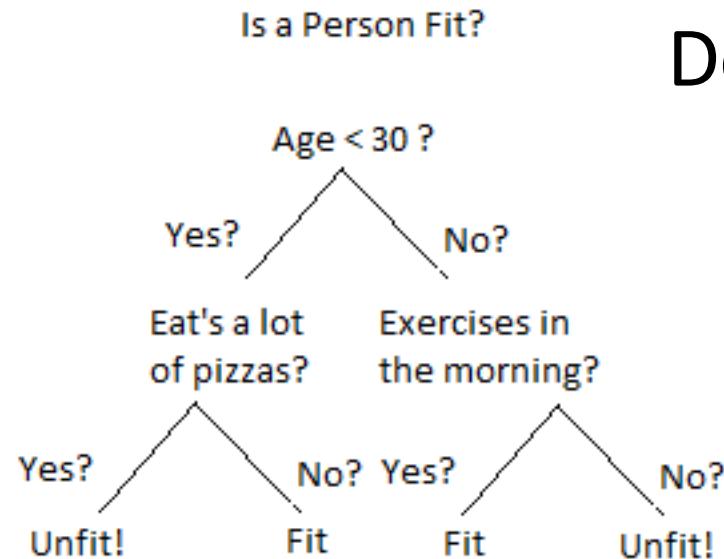


```
from sklearn.neighbors import KNeighborsClassifier  
clf = KNeighborsClassifier(n_neighbors = k)  
clf.fit(X_train, y_train)
```

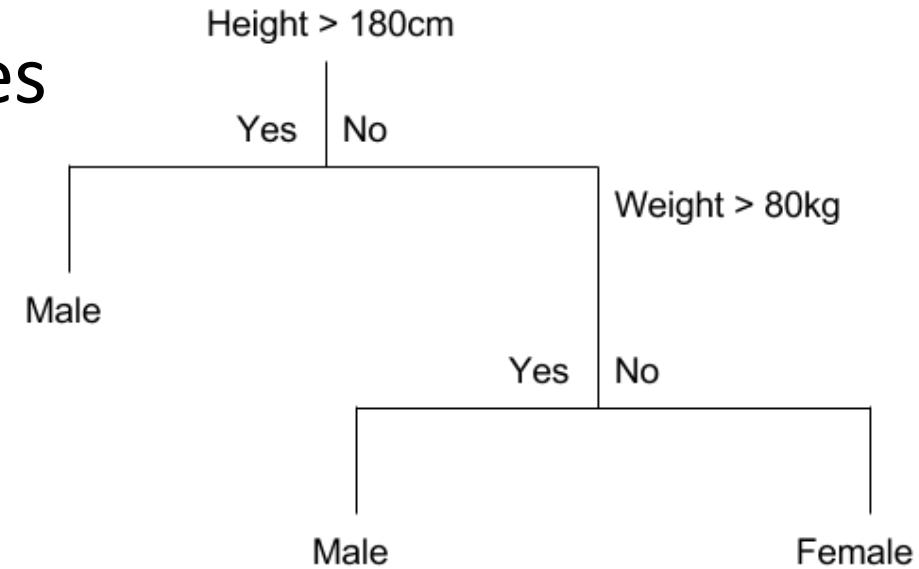
Determine the best value of k (neighbours)

Random Forest

- A group of decision trees on a subset of data and features voting for a classification of target

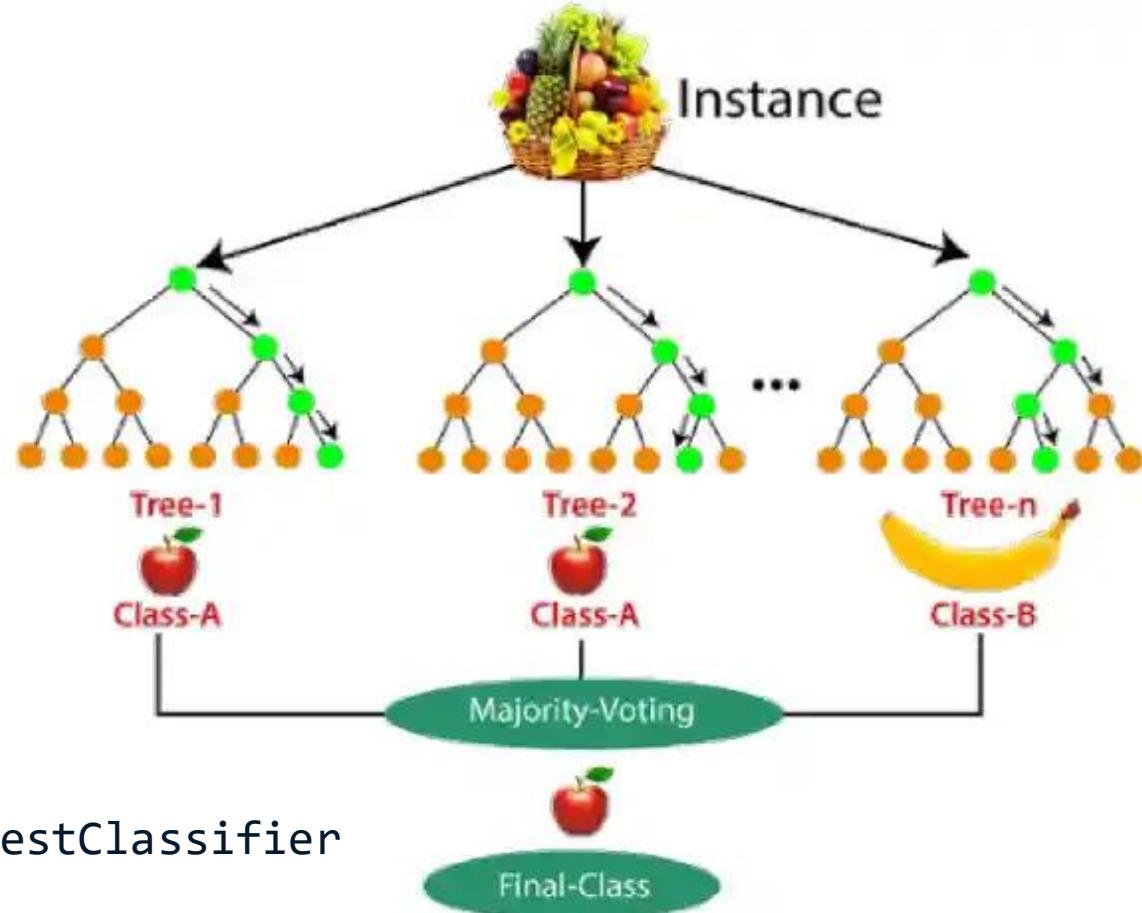


Decision trees



Random Forest

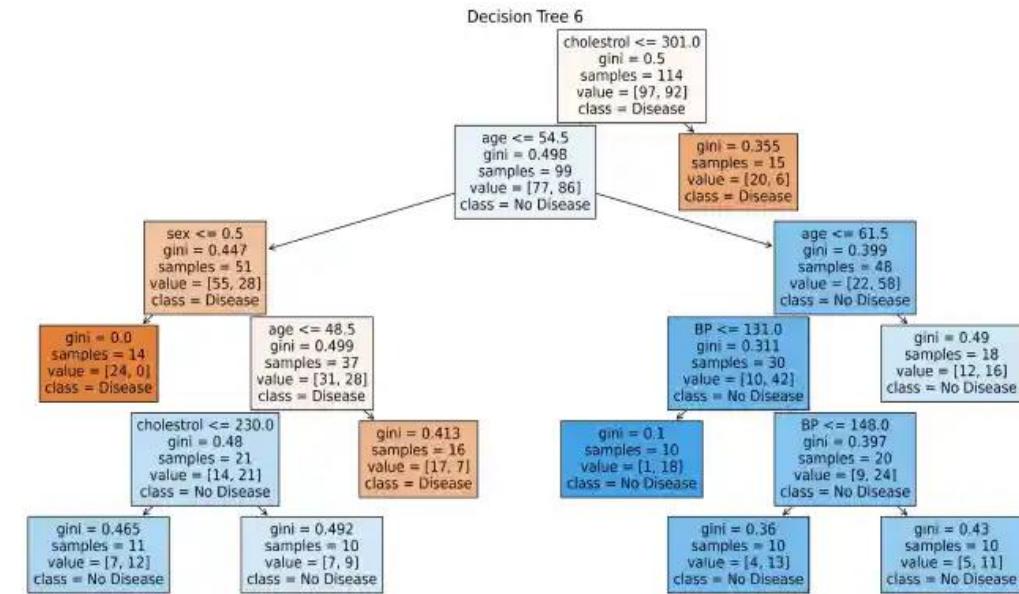
- A group of decision trees on a subset of data and features voting for a classification of target



```
from sklearn.ensemble import RandomForestClassifier  
rf = RandomForestClassifier()  
rf.fit(X_train, y_train)
```

Random Forest

- A group of decision trees on a subset of data and features voting for a classification of target



```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

Random Forest: Hyperparameters

There are two main hyperparameters

- 1) n_estimators: There are number of decision trees. Large numbers will improve the model but increase the computational cost
- 2) max_depth: This is the depth that a tree can grow. Higher values will lead to overfitting

Code:

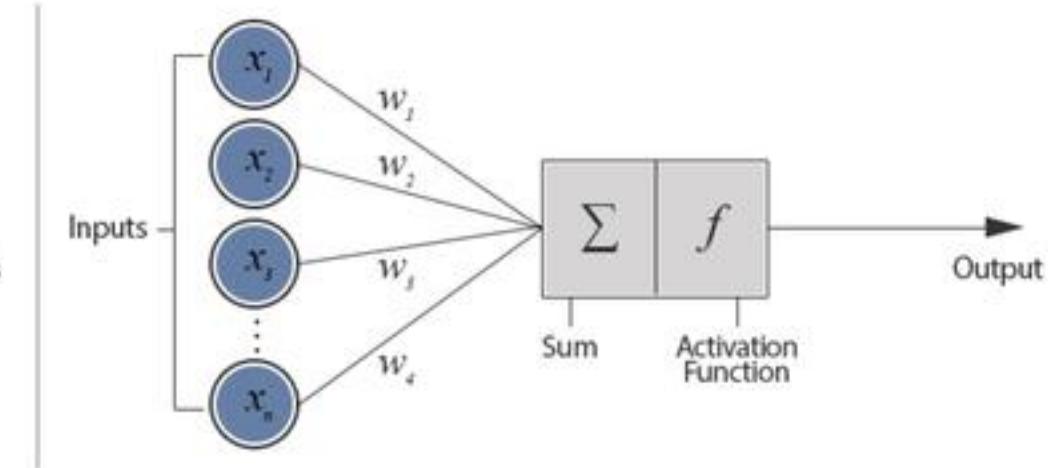
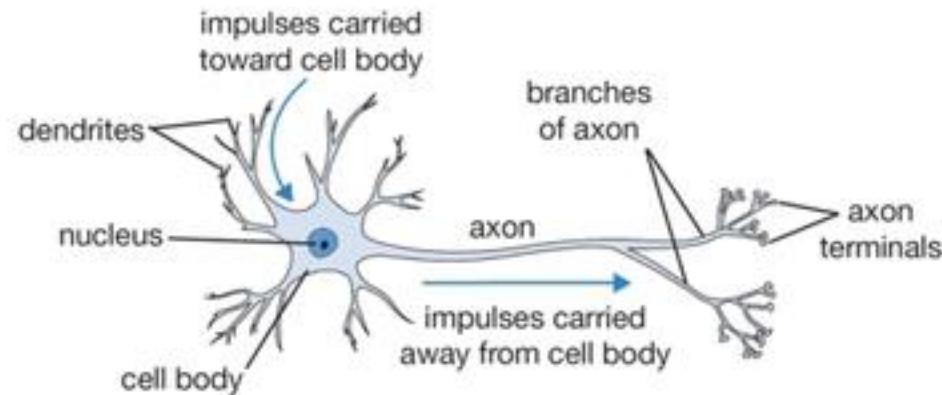
```
param_dist = {'n_estimators': randint(50,500), 'max_depth': randint(1,20)}
```

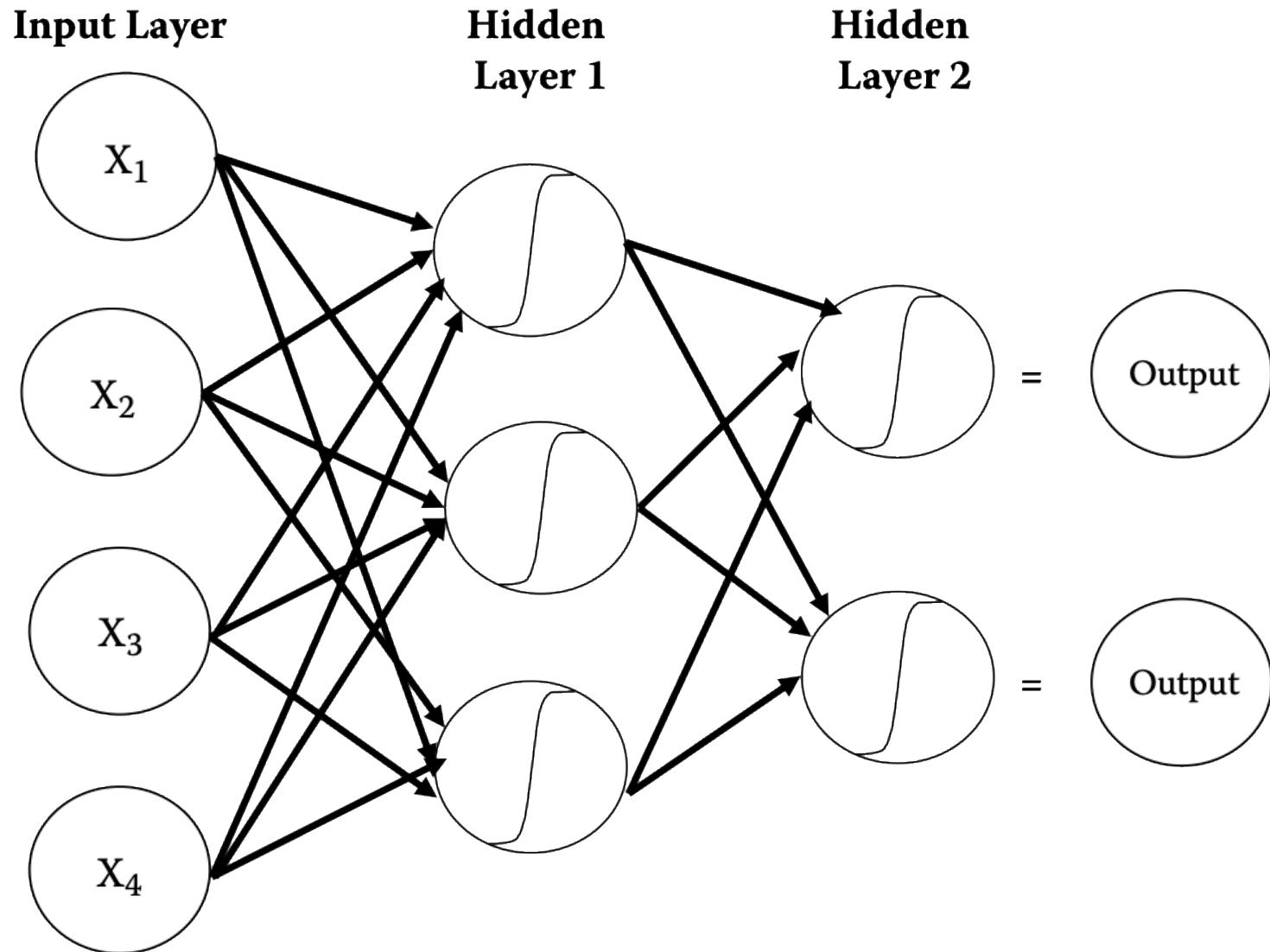
Use random search hyper parameter tuning

Artificial Neural Network

Perceptron: The foundation of Neural network

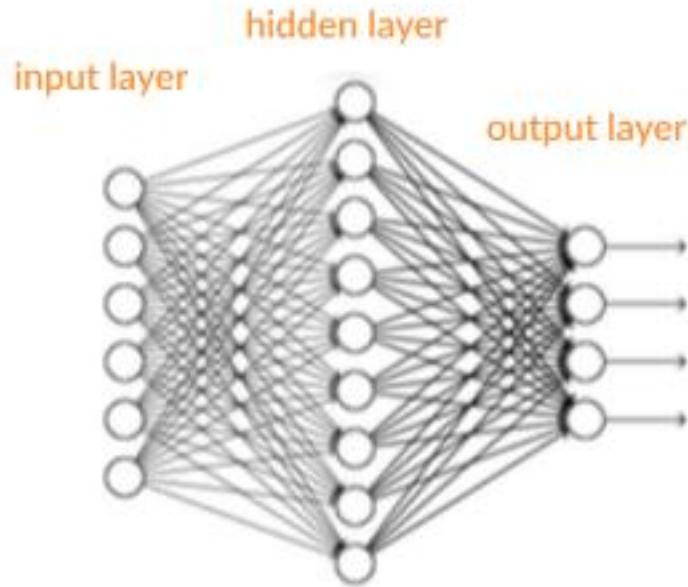
Biological Neuron versus Artificial Neural Network



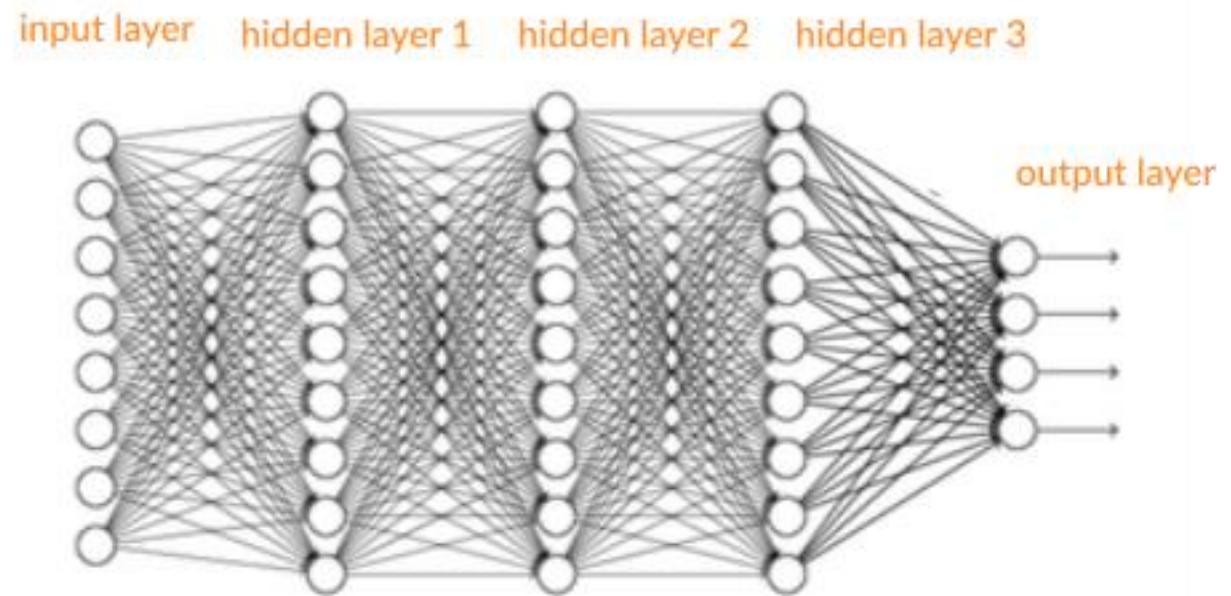


Shallow vs deep neural networks

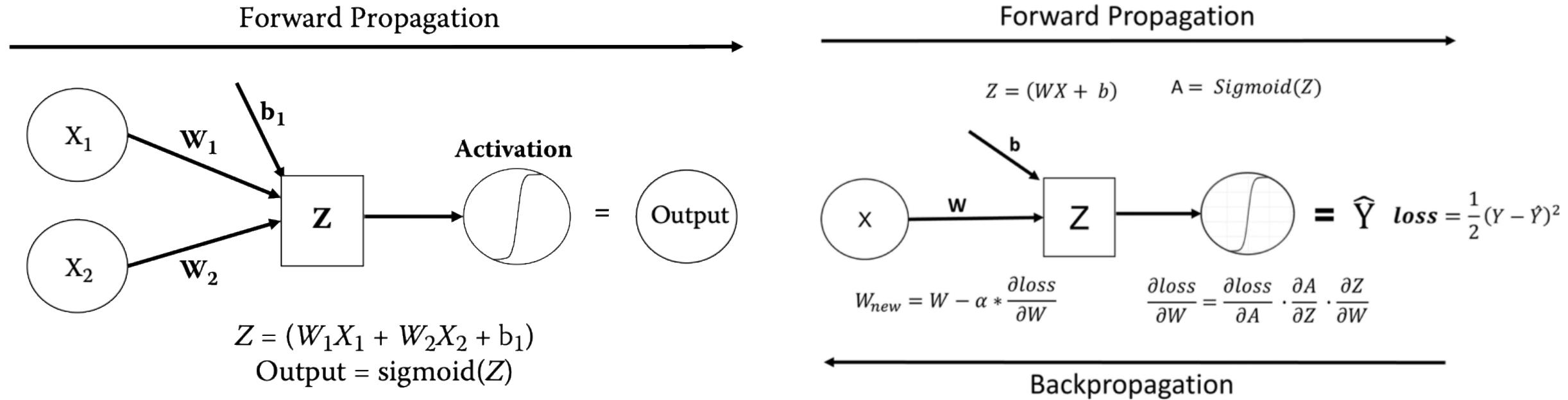
shallow feedforward
neural network



Deep neural network

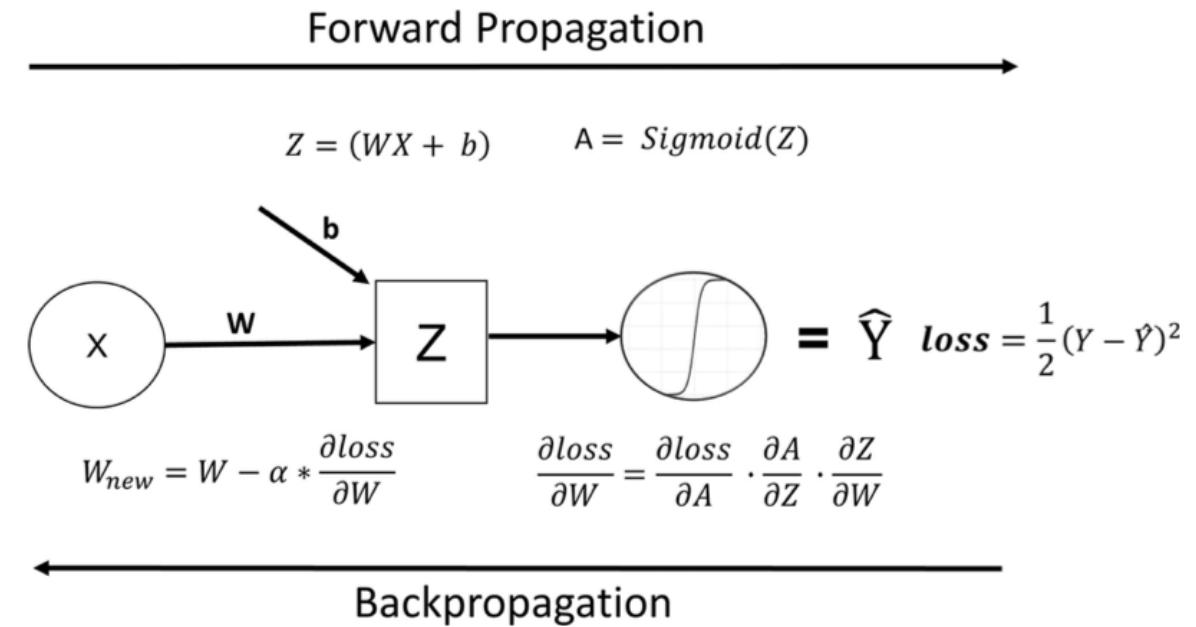
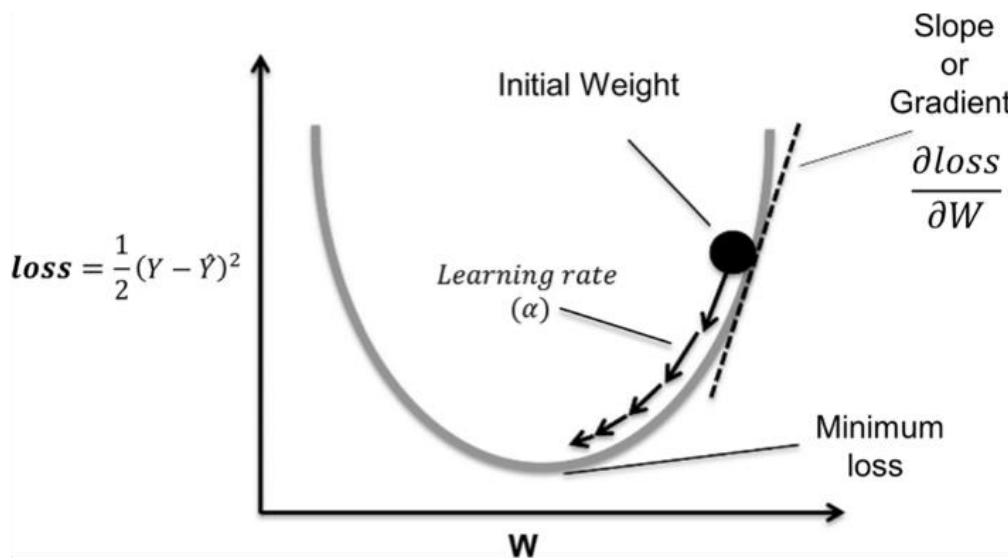


Forward and Backward Propagation: A way to optimize weights

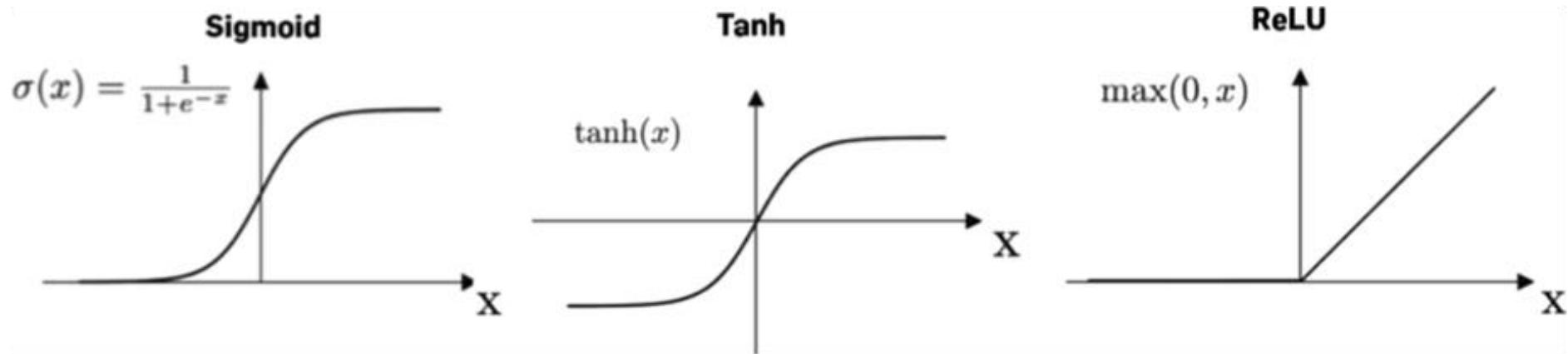


EPOCH is one cycle of forward and back passes.

Forward and Backward Propagation: A way to optimize weights



Activation functions



- Sigmoid is the most common activation function for input and output nodes
- Use the softmax function for multiclass problems
- ReLU captures non-linearity when used in hidden layers