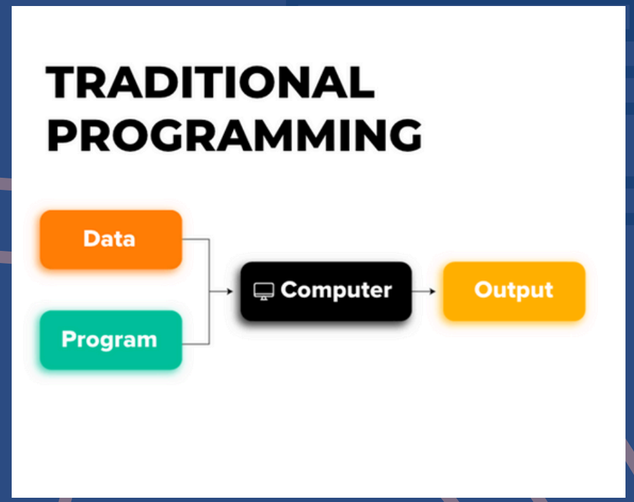
Electronics Club, IITK

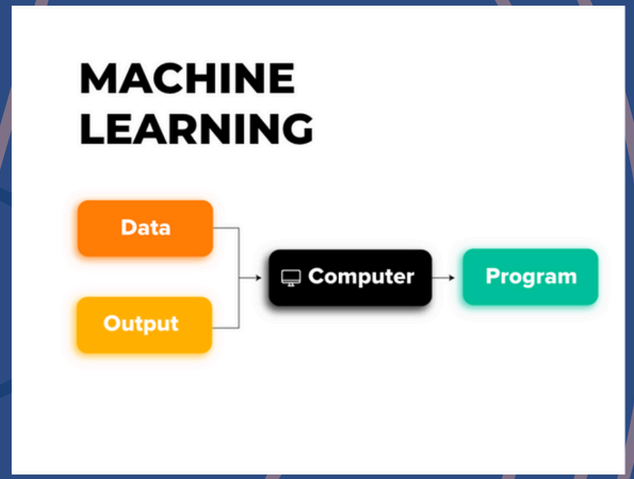
Machine Learning

Machine Learning

Machine learning is a subfield of computer science that concerns building algorithms that, to be useful, rely on a collection of examples of some phenomenon. These examples can come from nature, be handcrafted by humans, or be generated by another algorithm.
Machine learning can also be defined as the process of solving a practical problem by:

1.Gathering a dataset
2.Algorithmically building a statistical model based on that dataset. The statistical model is assumed to be used somehow to solve the practical problem.





Types Of ML

SUPERVISED

UNSUPERVISED

REINFORCEMENT

In supervised learning, the dataset is the collection of labeled examples:

$$(\mathbf{x}_i, y_i)\}_{i=1}^{N}$$

- Let us say that we want to build a simple ML model which will predict the house price based on a number of factors.
- One can include the Location, Square Footage, Number of rooms, whether it has an open terrace or not etc. to try and determine the price of a given house.
- These "features" are represented as a vector Xi,called the feature vector.
- These feauture vectors are then mapped to a label yi by the model. Here, the labels are house prices.
- In a Supervised ML model, we feed in the input feature vectors with the labels of the training data and then try to predict the labels for the corresponding feature vectors of the tesing data.

Types Of ML

SUPERVISED

UNSUPERVISED

REINFORCEMENT

In unsupervised learning, the dataset is the collection of unlabeled examples:

$$\{\mathbf{x}_i\}_{i=1}^{N}$$

- It is used for clustering and anamoly detection. We do not know anything about the data, and want the model to find patterns and group the data accordingly.
- Say, if you decide to buy Dove body wash products on Amazon, you'll probably be offered to add some toothpaste and a set of toothbrushes to your cart because the algorithm calculated that these products are often purchased together by other customers.

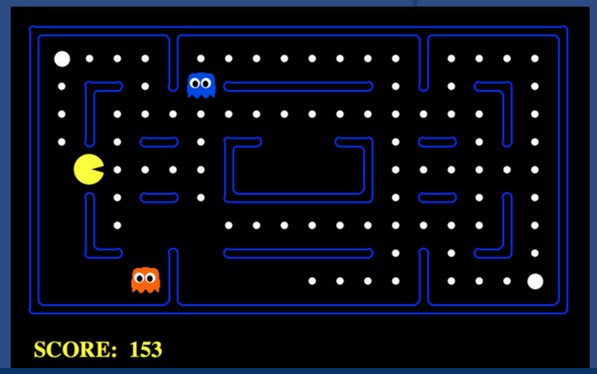
Types Of ML

SUPERVISED

UNSUPERVISED

REINFORCEMENT

TRIAL and ERROR approach. Gain Points when a "Correct" move is made, and attempt:



- A phenomenal RL application is in the Robotics field. A robot has to have the intelligence to perform unknown tasks when no successmeasure is given.
- It has to explore all the possibilities and reach its destined goal. A robot here is an agent if we consider the typical Reinforcement Learning nomenclature.
- The action is its movement. The environment is, say, a maze. The reward is the points that it receives on making a successful move.
- All four components taken together explain a Reinforcement Learning scenario.

Your First Step

NUMPY, PANDAS AND MATPLOTLIB

- Familiarise yourself with some of the liabraries in Python.
- Numpy is a mathematical and computing library with a strong support for arrays,matrices and more. Pandas and Matplotlib,Seaborn helps you manipulate and visualise data.
- Go through the undermentioned guides to get a good understanding of how it works and of its basic functions.

Data Preprocessing

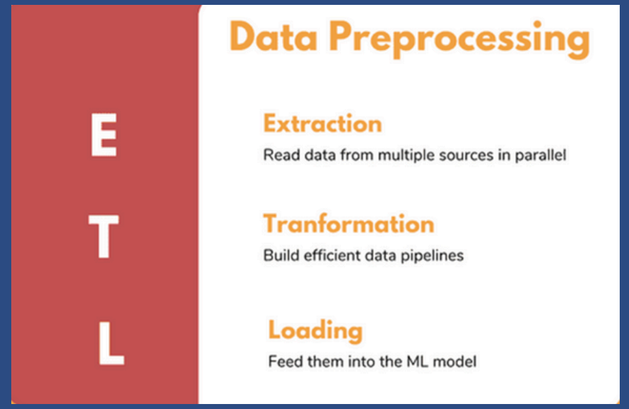
The steps involved in data preprocessing can be grouped as follow:

Merging data sets on common fields brings all data into a single table. The Pandas library contains suitable tools for merging, concatenation, and similar operations on datasets. pandas.merge() , pandas.concat() .

Cleaning the data by dealing with duplicate rows, incorrect or missing values, and other issues. Duplicated data needs to be removed to maintain accuracy and to avoid misleading statistics. A column with most of its values missing can be dropped as it provides less information. However, losing data can be bad sometimes. In that case, data imputation is done using measures of central tendency or advanced methods. Moreover, datasets with imbalanced classes are balanced using techniques like SMOTE.

Feature engineering to improve data quality, for example, using dimensionality reduction techniques (PCA and LDA) to build new features.

Building the training data sets by standardizing or normalizing the numerical data, encoding the categorical data, and splitting it into training and testing sets.



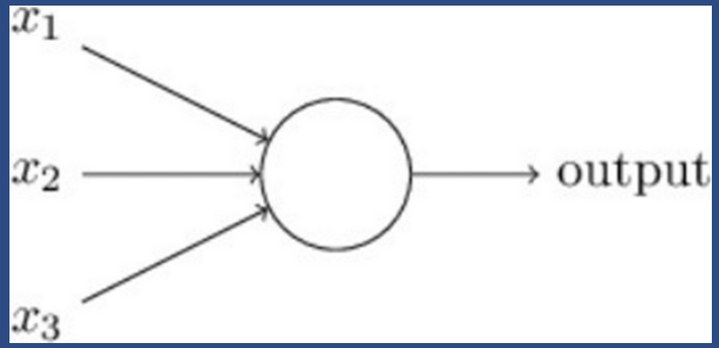
Electronics club, IITK

SUPERVISED LEARNING

REGRESSION

The Perceptron

A perceptron takes several binary inputs, x1,x2,x3,.....and produces asingle binary output:



How does it make a decision?

Say you want to go to Z-Square, you weigh the importance of some factors before making a decision.

1. Is there a quiz coming up?
2. Is there an assignment?
3. Is the weather good?

You may assign different "weights" or importance to each of the factors to make a decision. A quiz the next day may have a higher weight than a bad weather.

So,say the relative weights are 0.7, 0.1, 0.1.
Say, you have aquiz, no assignments and the weather is good. You apply the perceptron as follows: -0.7(1) +0.1(0) +0.1(1)

How does it make a decision?

Meaning, you would rather go out if you do not have a quiz or assignments, leading to a lower weight. Similarly you would rather have good weather to go out.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

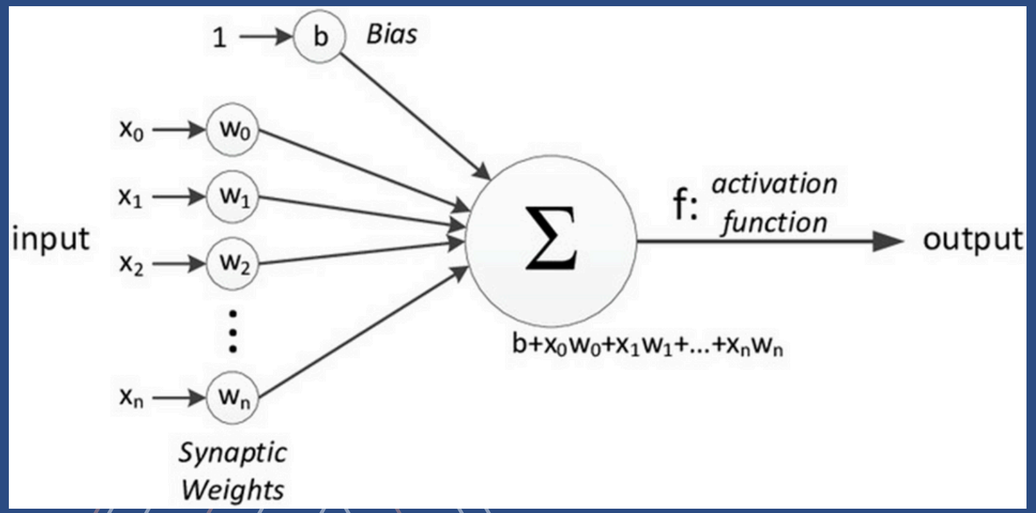
We need to set a threshold, if the output of the perceptron is lesser than it, we output a 0 and vice versa.
A better representaion is using the dot product w.x, and using a bias term "b"

Here, w and x are represented as vectors:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Activation Functions

Activation functions helps us add non-linearity to the output by 1 perceptron. Each perceptron function on the basis of w.x + b, but if we add an extra step of non-linearity, the model can capture complex patterns in our dataset.



Linear Regression

We have a collection of labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where N is the size of the collection, $\mathbf{x}_i$ is the D-dimensional feature vector of example $i = 1, \ldots, N$, y_i is a real-valued[1] target and every feature $x_i^{(j)}$, $j = 1, \ldots, D$, is also a real number.

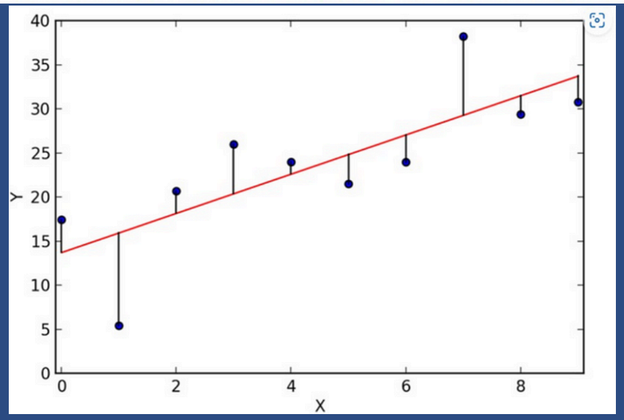
We want to build a model $f_{\mathbf{w},b}(\mathbf{x})$ as a linear combination of features of example $\mathbf{x}$:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b, \tag{1}$$

where $\mathbf{w}$ is a D-dimensional vector of parameters and b is a real number. The notation $f_{\mathbf{w},b}$ means that the model f is parametrized by two values: $\mathbf{w}$ and b.

Linear Regression

We will use the model to predict the unknown y for a given $\mathbf{x}$ like this: $y \leftarrow f_{\mathbf{w},b}(\mathbf{x})$. Two models parametrized by two different pairs $(\mathbf{w}, b)$ will likely produce two different predictions when applied to the same example. We want to find the optimal values $(\mathbf{w}^*, b^*)$. Obviously, the optimal values of parameters define the model that makes the most accurate predictions.



This is an example where x is 1-dimensionsal. If x would be n-dimensional, the only difference would that instead of the cartesian plane, we would be in the n-dimensional plane hyper plane.

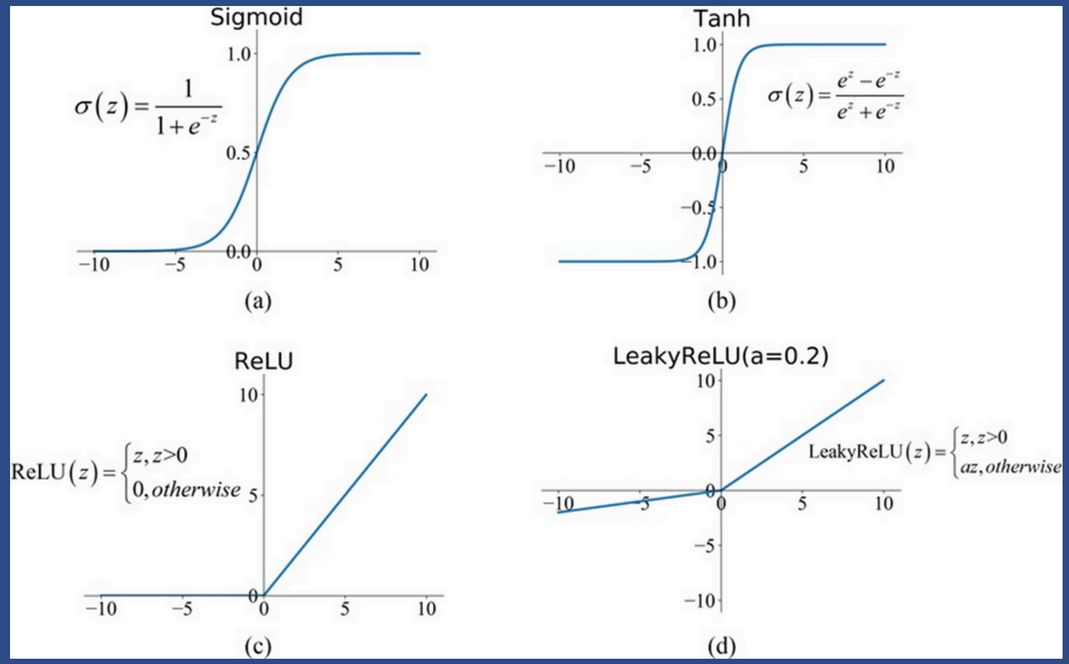
How to find w and b?

To find the line-of-best-fit, we need to do an optimization process:

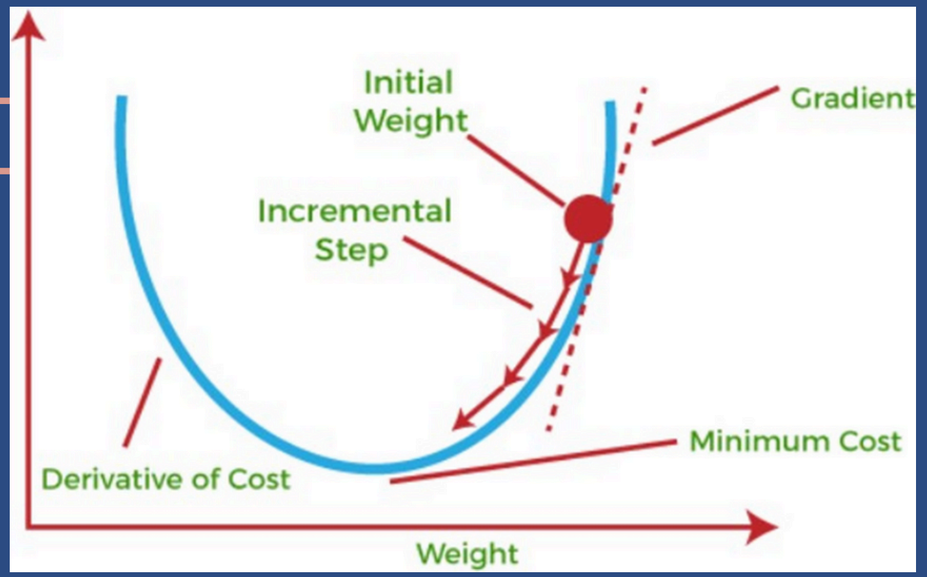
$$\frac{1}{N}\sum_{i=1\ldots N}(f_{\mathbf{w},b}(\mathbf{x}_i)-y_i)^2. \quad (2)$$

In mathematics, the expression we minimize or maximize is called an objective function, or, simply, an objective. The expression $(f(\mathbf{x}_i)-y_i)^2$ in the above objective is called the **loss function**. It's a measure of penalty for misclassification of example i. This particular choice of the loss function is called **squared error loss**. All model-based learning algorithms have a loss function and what we do to find the best model is we try to minimize the objective known as the **cost function**. In linear regression, the cost function is given by the average loss, also called the **empirical risk**. The average loss, or empirical risk, for a model, is the average of all penalties obtained by applying the model to the training data.

Activation Functions



X is a vector of dimensions (num_samples,n_features). W is a column vector of dimensions (num_features,1). B is a real number. So equation is y = f(X.W+b).



Algorithm :

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

(Learning rate — It is a number b/w 0 and 1 which controls how big a step is during gradient descent)

↓ assignment operator.

$$b := b - \alpha \frac{\partial}{\partial b} J(w,b)$$

$$tmp\_w = w - \alpha \frac{\partial}{\partial w} J(w,b)$$

$$tmp\_b = b - \alpha \frac{\partial J}{\partial b}(w,b)$$

$$w = tmp\_w$$

$$b = tmp\_b$$

(✓)

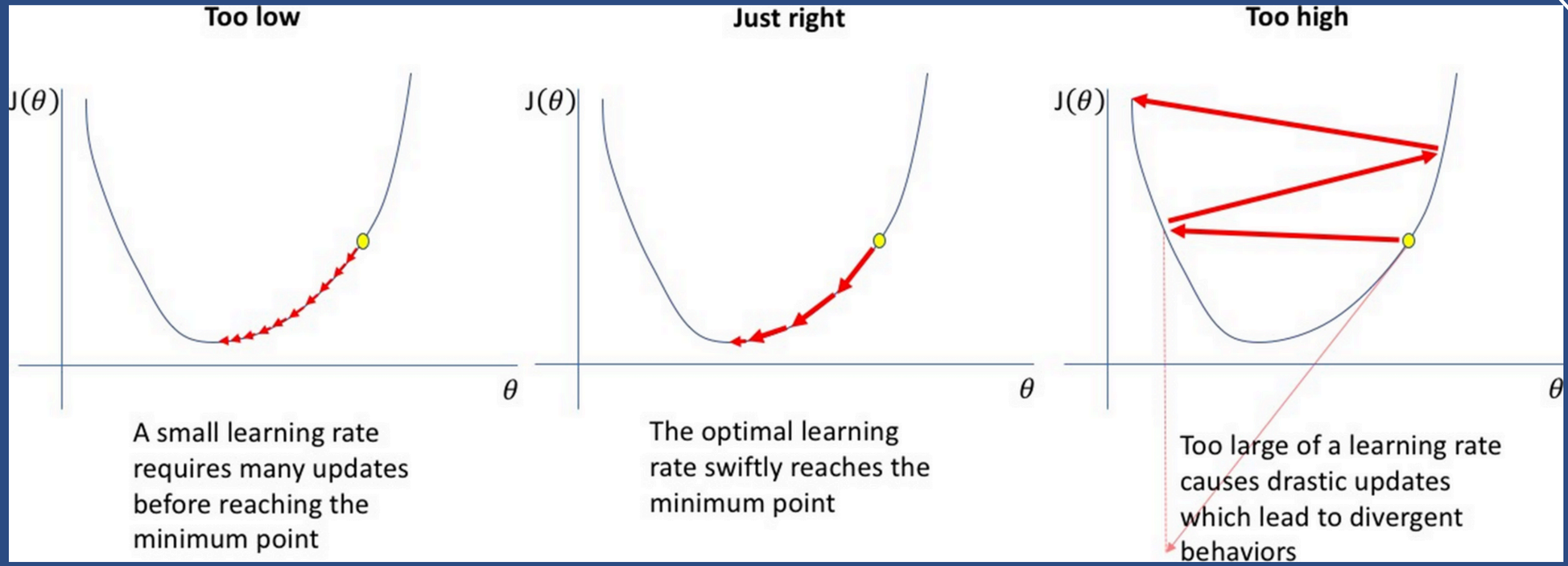
$$tmp\_w = w - \alpha \frac{\partial J}{\partial w}(w,b)$$

$$w = tmp\_w$$

$$tmp\_b = b - \alpha \frac{\partial J}{\partial w}(w,b)$$

$$b = tmp\_b$$

(X)


Too low
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
Just right
J(θ)
θ
The optimal learning rate swiftly reaches the minimum point
Too high
J(θ)
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

The Gradient Descent Algorithm

$$l = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2.$$

We must minimize the above cost function to find optimum values of w and b. Gradient descent starts with calculating the partial derivative for every parameter:

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b));$$

$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b)).$$

Evaluation Metrics for Regression

Mean Absolute Error(MAE) : MAE calculates the absolute difference between actual and predicted values. sum of (y_test - y_pred) / total_samples.
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y_test,y_pred)

Mean Squared Error(MSE) : finds the squared difference between actual and predicted value.
sum of (y_test - y_pred)^2 / total_samples.
from sklearn.metrics import mean_squared_error
mean_squared_error(y_test,y_pred)

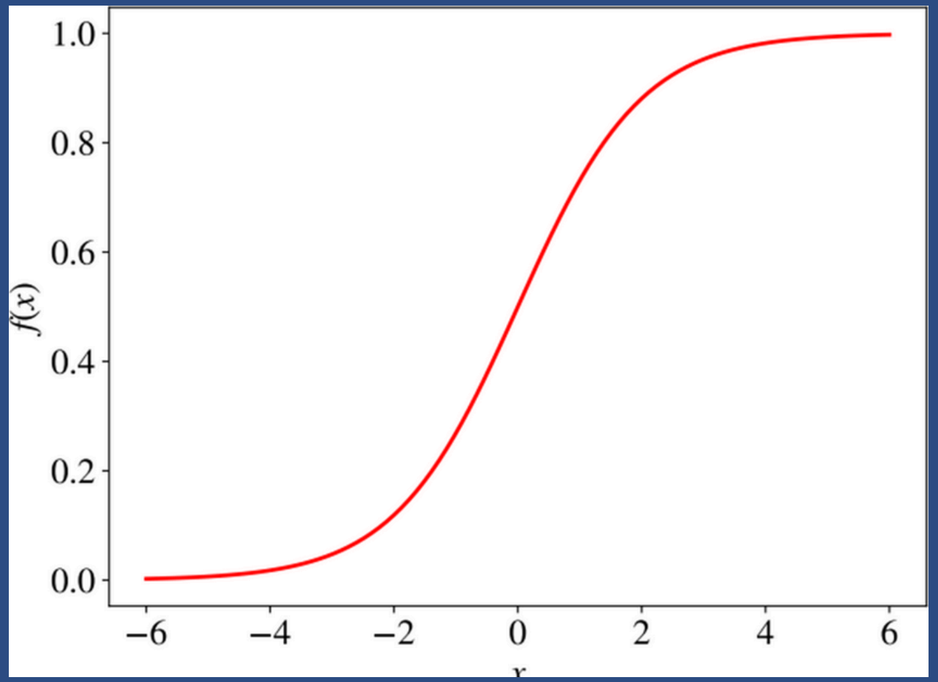
Root Mean Squared Error(RMSE) : root over MSE
np.sqrt(mean_squared_error(y_test,y_pred))

R Squared (R2) : also known as Coefficient of Determination or Goodness of fit. R2 squared calculates how much is the regression line better than the mean line.

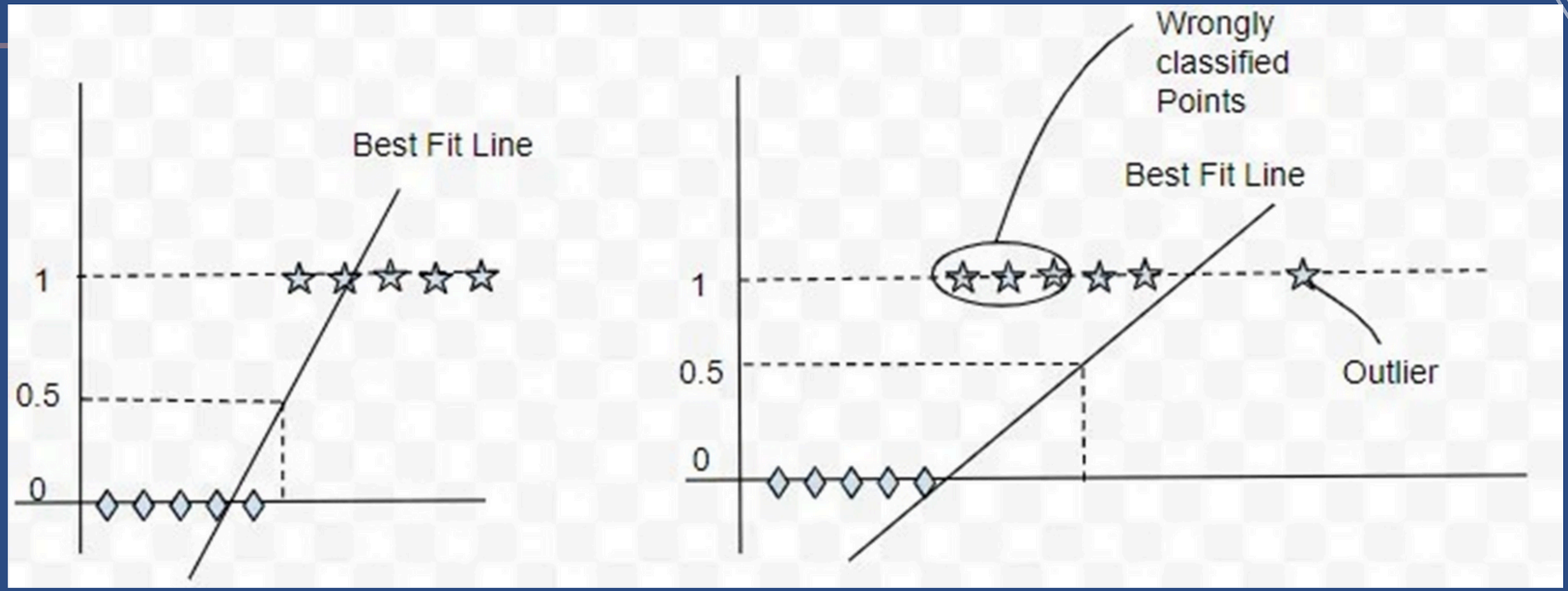
R2 = 1 - (sum of (y_test - y_pred)^2)/(sum of (y_test - y_test_mean)^2) from sklearn.metrics import r2_score
r2 = r2_score(y_test,y_pred)

Logistic Regression

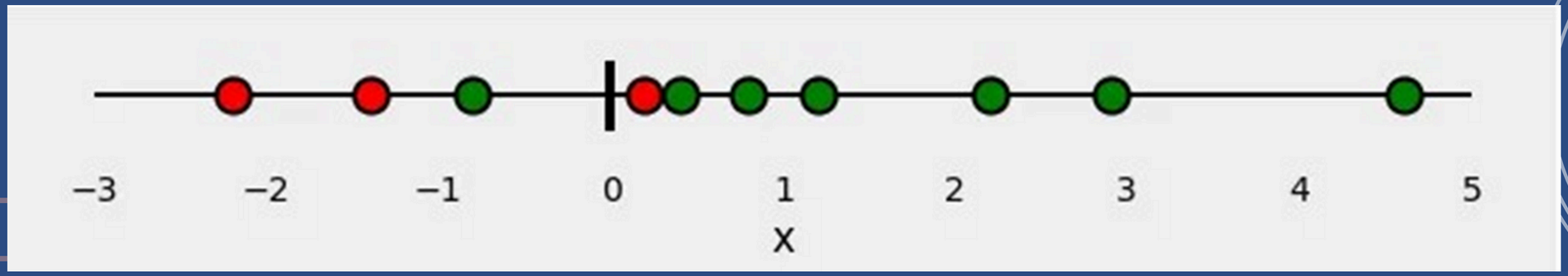
In logistic regression, we still want to model y_i as a linear function of $\mathbf{x}_i$, however, with a binary y_i this is not straightforward. The linear combination of features such as $\mathbf{w}\mathbf{x}_i + b$ is a function that spans from minus infinity to plus infinity, while y_i has only two possible values.



Why not Linear Regression?



Outlier: If the data set has an outlier, linear regression will not perform better.

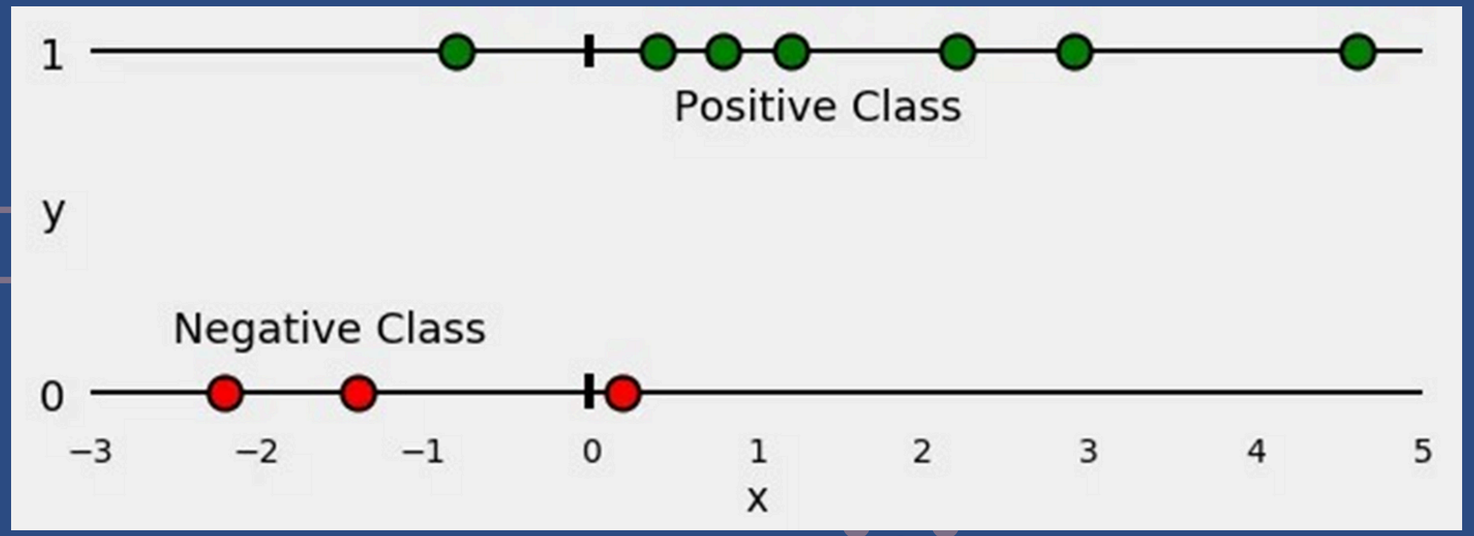


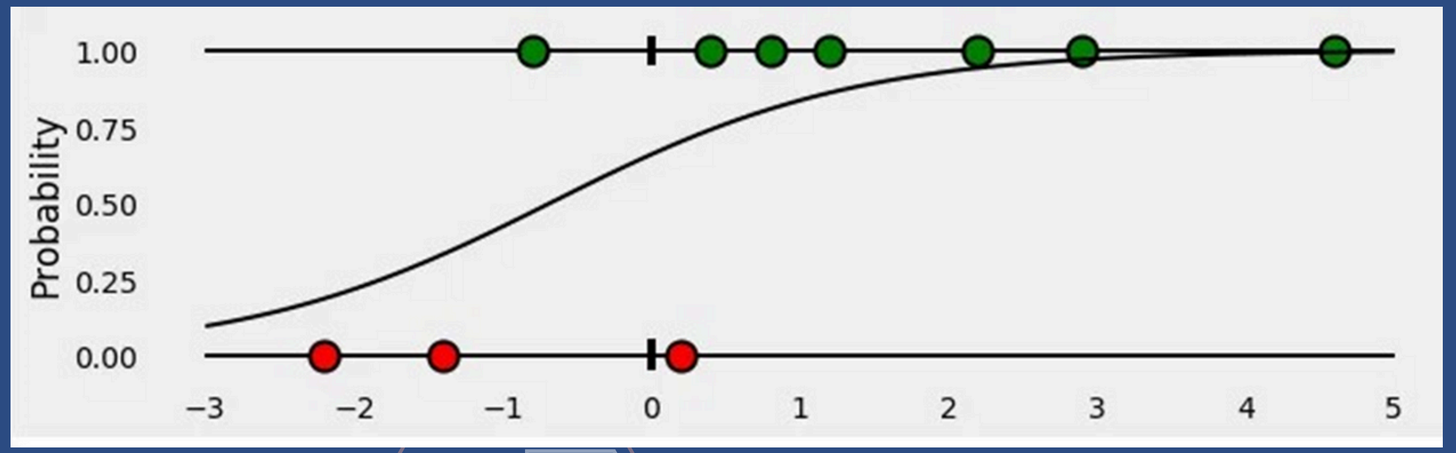
If we fit a model to perform this classification, it will predict a probability of being green to each one of our points.

Given what we know about the color of the points, how can we evaluate how good (or bad) are the predicted probabilities?

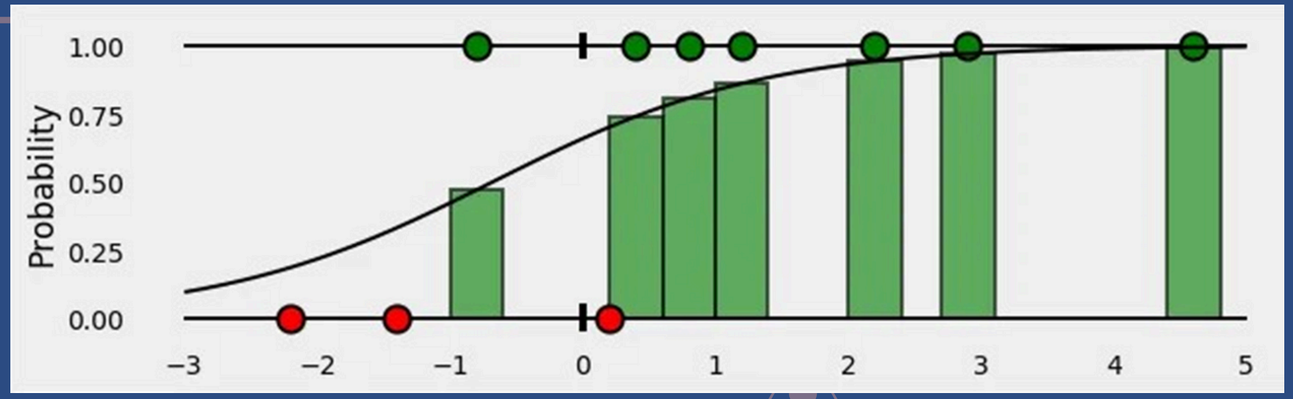
This is the whole purpose of the loss function! It should return high values for bad predictions and low values for good predictions.

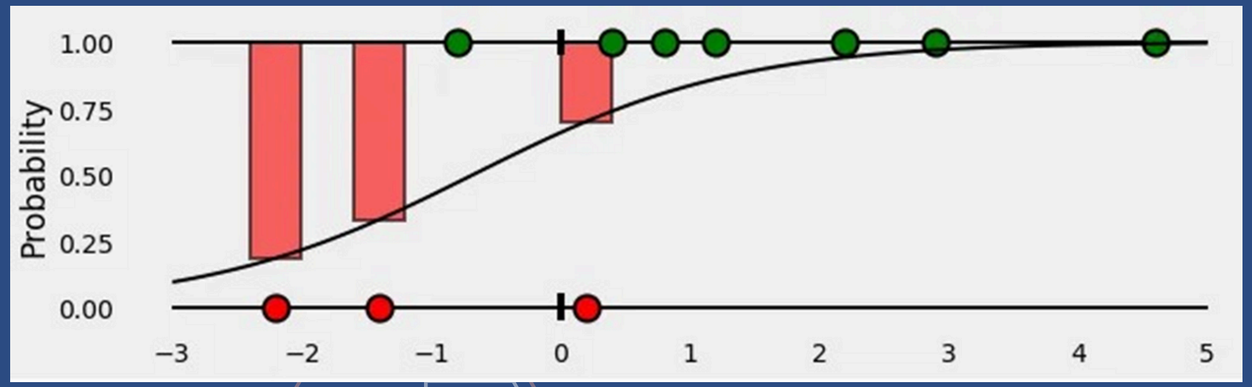
$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

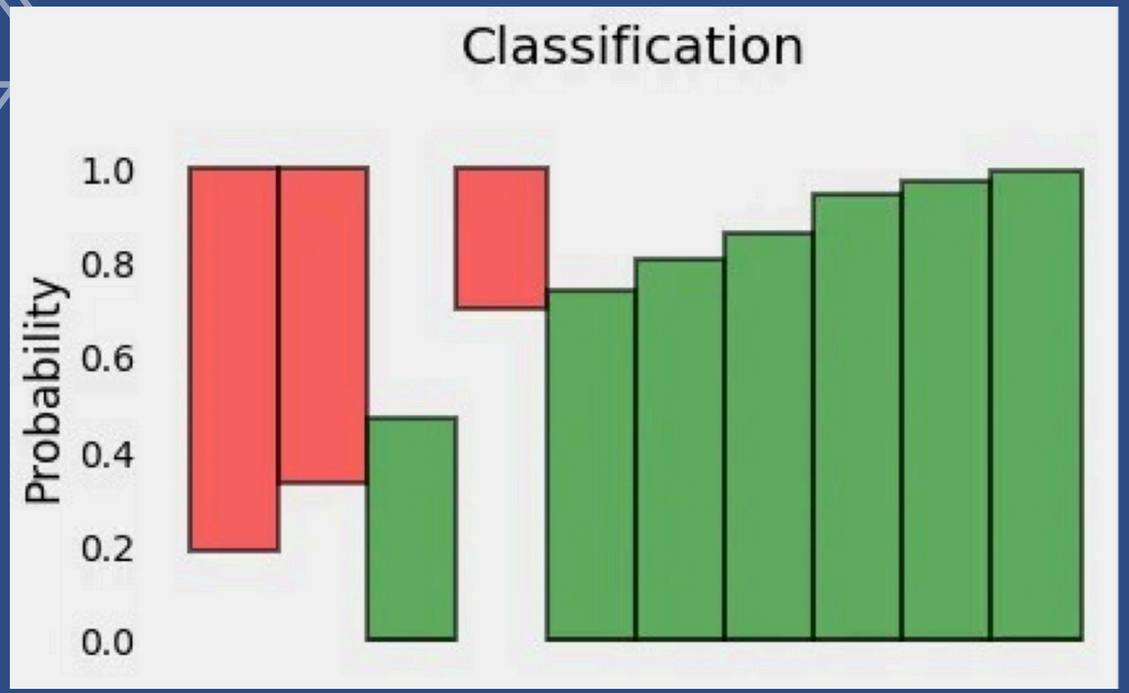

1
Positive Class
y
Negative Class
0
−3
−2
−1
0
1
2
3
4
5
x


1.00
0.75
0.50
0.25
0.00
Probability
−3
−2
−1
0
1
2
3
4
5

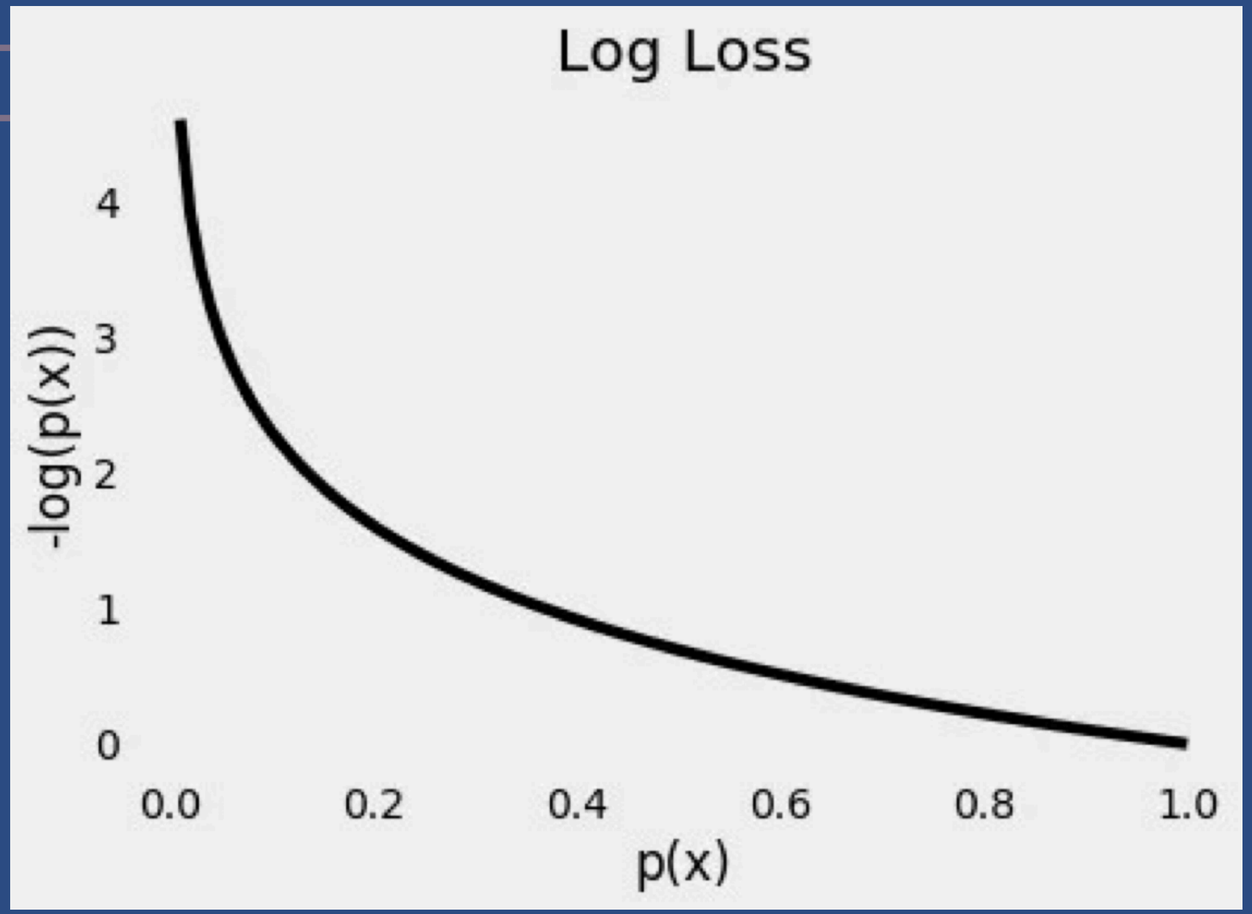
Then, for all points belonging to the positive class (green), what are the predicted probabilities given by our classifier? These are the green bars under the sigmoid curve, at the x coordinates corresponding to the points.

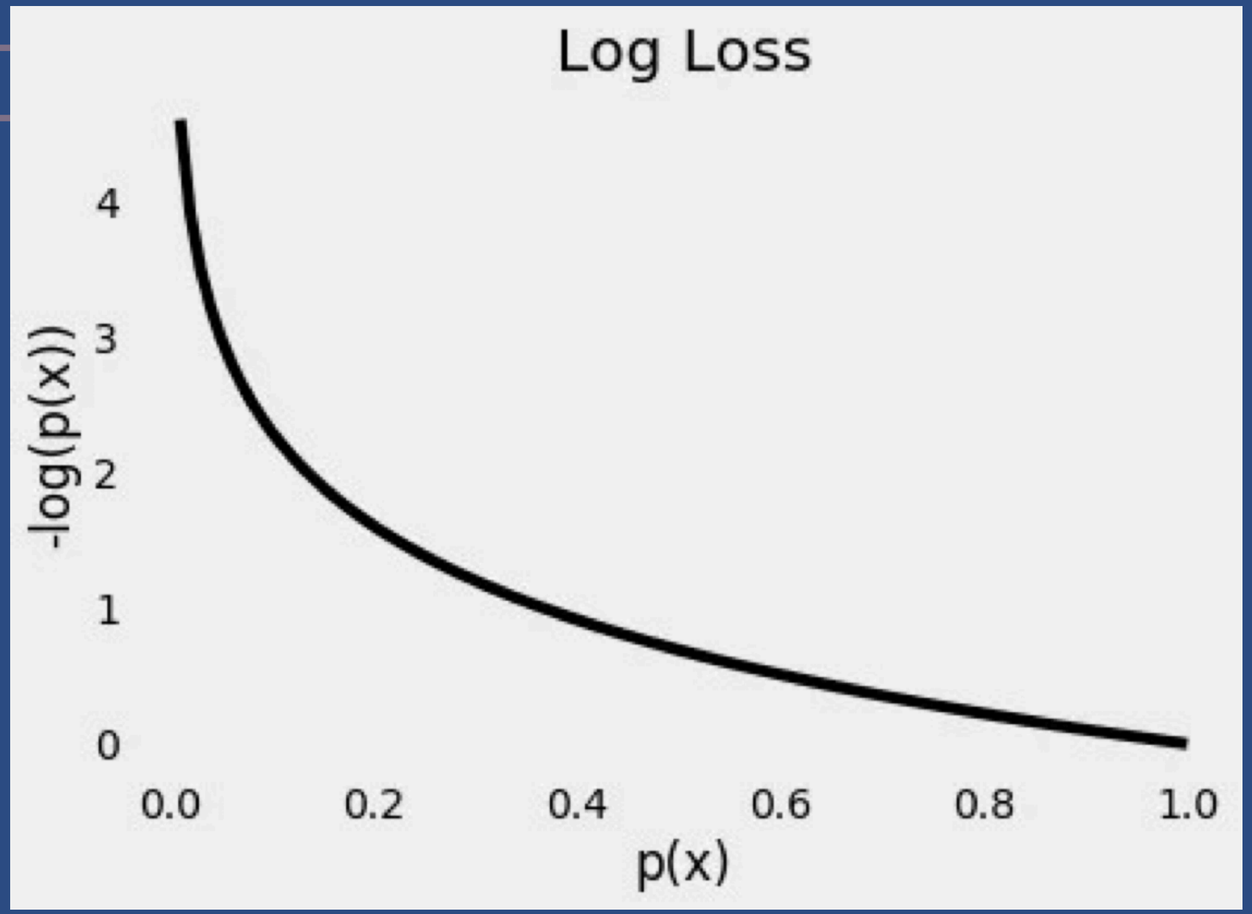





Classification
Probability
1.0
0.8
0.6
0.4
0.2
0.0

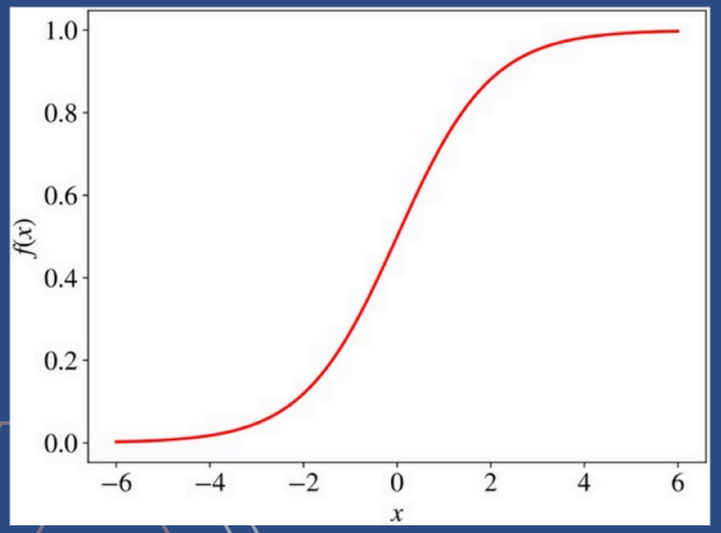

Classification
Negative
Positive
Probability
1.0
0.8
0.6
0.4
0.2
0.0


Log Loss
-log(p(x))
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
p(x)


Log Loss
-log(p(x))
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0
p(x)

By looking at the graph of the standard logistic function, we can see how well it fits our classification purpose: if we optimize the values of $\mathbf{x}$ and b appropriately, we could interpret the output of $f(\mathbf{x})$ as the probability of y_i being positive. For example, if it's higher than or equal to the threshold 0.5 we would say that the class of $\mathbf{x}$ is positive; otherwise, it's negative. In practice, the choice of the threshold could be different depending on the problem. We

$$f_{\mathbf{w},b}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{1+e^{-(\mathbf{w}\mathbf{x}+b)}}. \tag{3}$$



Squared error cost

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right) - y^{(i)}\right)^2$$

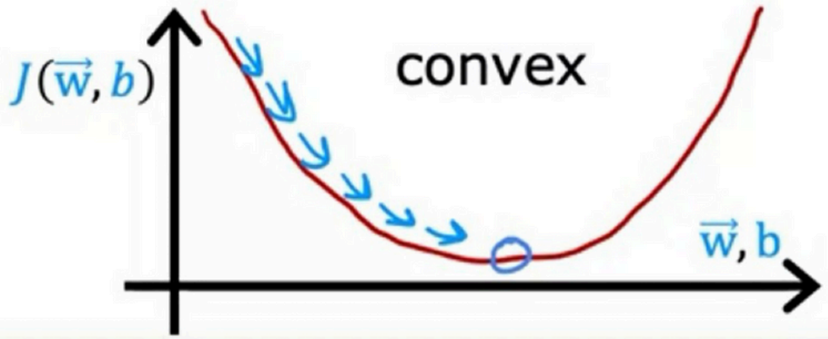
loss $L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)$

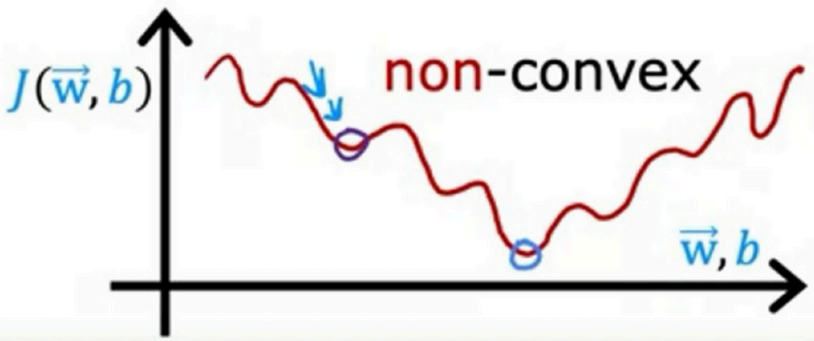
linear regression

$$f_{\vec{w},b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

logistic regression

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$





Logistic loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} \underbrace{L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)}_{\text{loss}}$$

$$= \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

Simplified loss function

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = \begin{cases} -\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 1 \\ -\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) & \text{if } y^{(i)} = 0 \end{cases}$$

$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

Simplified cost function

loss

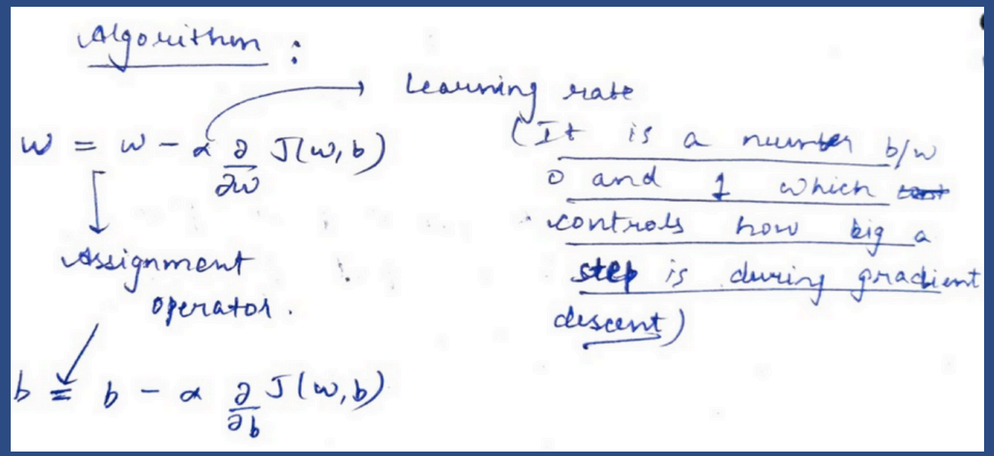
$$L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right) = -y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) - \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)$$

cost

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left[L\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right), y^{(i)}\right)\right]$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right) + \left(1 - y^{(i)}\right)\log\left(1 - f_{\vec{w},b}\left(\vec{x}^{(i)}\right)\right)\right]$$

Gradient Descent



$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\frac{\partial(J(\theta))}{\partial(\theta)} = \frac{1}{m} X^T [\, h_{\theta}(x) - y]$$

This derivative of cost function is with respect to weight and biases for each sample.
For calculating the total derivative we do summation over all the samples

Evaluation Metrics for Classification

Accuracy : correct predictions / total predictions. OR (TP + TN)/(TP + TN+ FP +FN) **sklearn.metrics.accuracy_score**

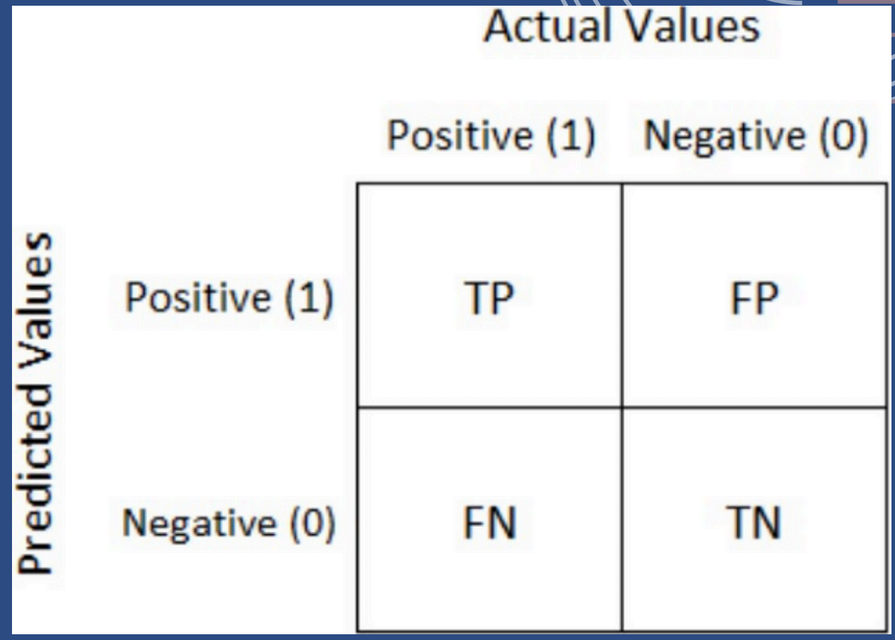
Precision : number of true positives divided by the number of predicted positives. (TP)/(TP+FP)
sklearn.metrics.average_precision_score

Recall (Sensitivity): number of true positives divided by the total number of actual positives. TP/(TP+FN)

F1 Score : harmonic mean of precision and recall.

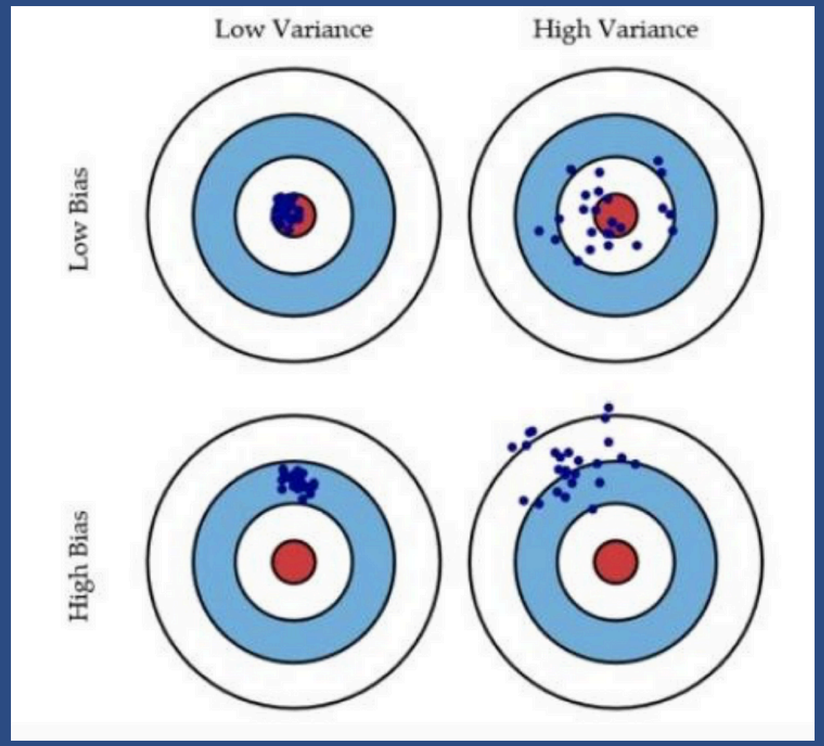
AUC-ROC— **Read here**
sklearn.metrics.roc_auc_score

Binary Cross Entropy Loss or Log Loss :
y_test ln (y_pred) + (1-y_test) ln (1- y_pred)



sklearn.metrics.confusion_matrix

Bias and Variance in Regression Models



Let’s say we have model which is very accurate, therefore the error of our model will be low, meaning a low bias and low variance as shown in first figure. All the data points fit within the bulls-eye. Similarly we can say that if the variance increases, the spread of our data point increases which results in less accurate prediction. And as the bias increases the error between our predicted value and the observed values increases.

Electronics club, IITK

SUPERVISED LEARNING

Classification Algorithms

KNN Algorithm

KNN stands for K nearest neighbour. The name itself suggests that it considers the nearest neighbour. It is one of the supervised machine learning algorithms. Interestingly we can solve both classification and regression problems with the algorithm.It is one of the simplest Machine Learning models. Though it is a simple model, sometimes it plays a significant role,basically when our dataset is small,and the problem is simple.

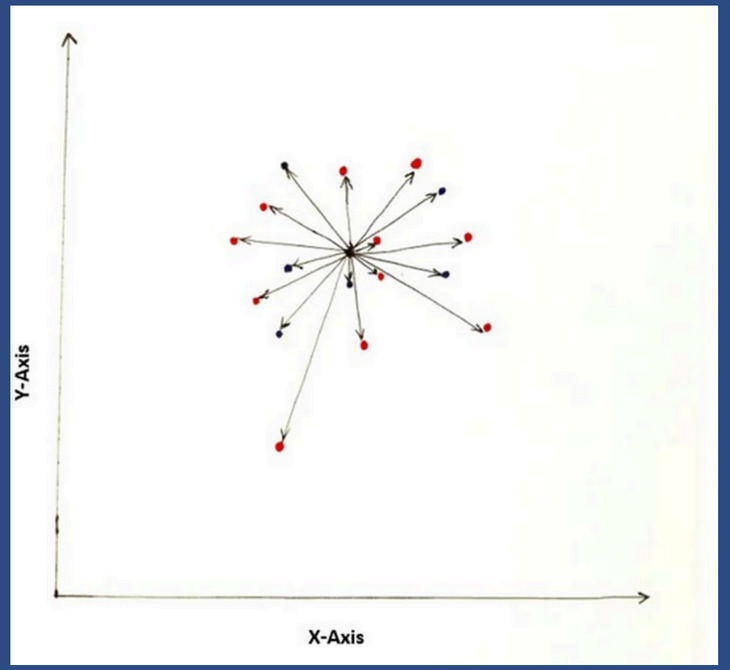
Overview of the KNN Algorithm

Step 1: Calculating the Distance

First of all, we need to load the labelled dataset as the KNN algorithm is a supervised learning algorithm

Suppose our dataset has only two features, and we plotted the data as shown in the image. Blue and Red points indicate two different categories. Let's have new unlabelled data that requires classification based on the given dataset.

In the image, the central point needs to be classified. Now, we will calculate the distance of all the data from the unlabelled data. The arrow from the central point represents the distances.



Overview of the KNN Algorithm

Step 2: Selecting the K-nearest neighbour

In the previous step, we calculated the distances of the new point from all other data. We will sort the data points in ascending order according to the distance.Finally, we will consider the Knumber of nearest points from the unlabelled data.

Lets consider, I have considered the 3 nearest data points (K=3). Observe the image; among 3 nearest points, 2 data belong to the red category, and 1 to the blue category.
So, red is the majority class. According to the KNN algorithm, new data points will be classified as red.

In case of a regression problem, we will consider the average value of K nearest data points.