14/10/22

CSE 313

Fundamentals of Machine Learning

HITESH
A023119820027
5 AI 1

ASSIGNMENT - 2

(1)

class / genre	Recommend	Not Recommend
Romance	2/6	1/2
Thriller	1/6	1/2
classic	3/6	0

class / Price	Recommend	Not Recommend
Low	1/2	0
Medium	2/6	0
High	1/6	2/2

$P(\text{Recommend}) = \frac{6}{8} = \frac{3}{4}$

$P(\text{Not Recommend}) = \frac{2}{8} = \frac{1}{4}$

$P(\text{Recommend} \mid \text{Thriller, Medium})$

$= P(\text{Recommend})\, P(\text{Thriller} \mid \text{Recommend})\, P(\text{Medium} \mid \text{Recommend})$

$= \frac{3}{4} \times \frac{1}{6} \times \frac{2}{6} = \frac{1}{24}$

(2) $P(\text{Not Recommend} \mid \text{Thriller, Medium})$

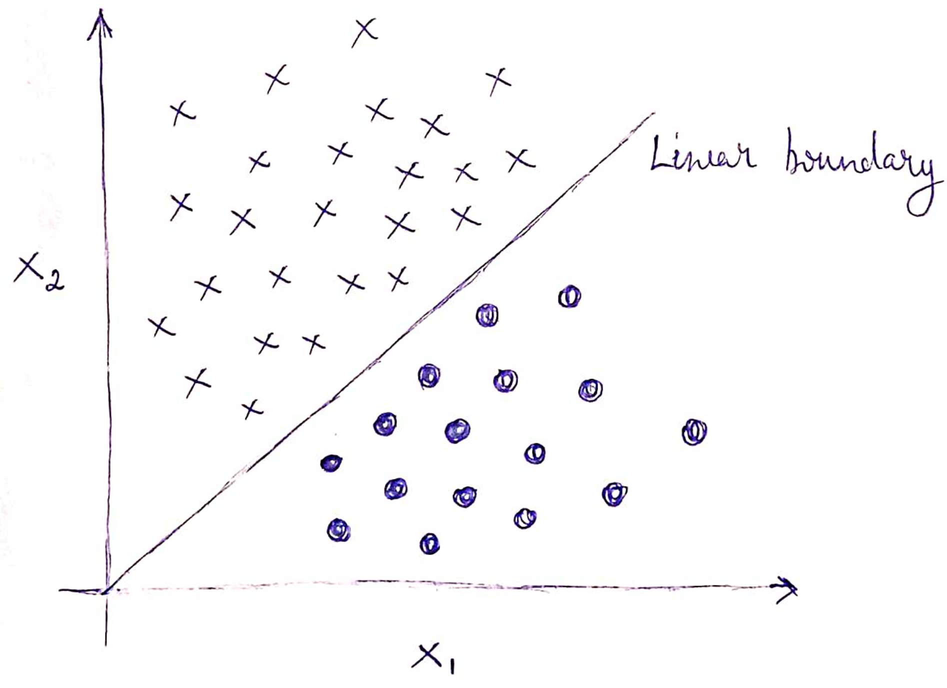
$= P(\text{Not Recommend})\, P(\text{Thriller} / \text{Not Recommend})\, P(\text{Medium} / \text{Not Recommend})$

$= \frac{1}{4} \times \frac{1}{2} \times 0$

$= 0$

③ The concept of Linear Separability applies to binary classification. Linear separability is a property of two sets of points.

The linear separability of the network is based on the decision-boundary line. If there exist weight for which the training input vectors having a positive (correct) response or lie on one side of the decision boundary and all the other vectors having negative, -1, response lies on the other side of the decision boundary then we can conclude the problem as "Linearly Separable".



Class A (x) and class B (●) are linearly separated from each other.

④ Prior Probability

$$P(\text{class} = +) = \frac{2}{7}$$

$$P(\text{class} = -) = \frac{5}{7}$$

Instances	Feature 1 \ class	+	-
	T	$\frac{1}{2}$	$\frac{3}{5}$
	F	$\frac{1}{2}$	$\frac{2}{5}$

Feature 2 \ class	+	-
T	$\frac{2}{2} = 1$	0
F	0	$\frac{5}{5} = 1$

New instance = { Feature 1 = T, Feature 2 = T }

Now,

$P(\text{class}=+ \mid \text{New instance})$

$\Rightarrow P(+)\ P(\text{Feature } 1 = T \mid +)\ P(\text{Feature } 2 = T \mid +)$

$\Rightarrow \frac{2}{7} \times \frac{1}{2} \times 1$

$\Rightarrow \frac{1}{7} = 0.1429$

$P(\text{class} = - \mid \text{New instance})$

$\Rightarrow P(-)\, P(\text{Feature } 2 = T \mid -)\; P(\text{Feature } 1 = T \mid -)$

$\Rightarrow \frac{5}{7} \times 0 \times \frac{3}{5}$

$\Rightarrow 0$

Since, $P(\text{class} = + \mid \text{New instance}) > P(\text{class} = - \mid \text{New instance})$

Therefore, the class for instance 8 with Feature 1 = T and Feature 2 = T is <u>+ class</u>.

---

(5)

x	y	xy	x^2
0	1	0	0
1	2	2	1
2	2	4	4
3	3	9	9
4	3	12	16
5	4	20	25
15	15	47	55

By the method of least square regression

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

$$= \frac{6 \times 47 - 15 \times 15}{6 \times 55 - 15^2}$$

$$= 0.5429$$

$$b_0 = \frac{1}{n}\left(\sum y - b_1 \sum x\right)$$

$$= \frac{1}{6}(15 - 0.5429 \times 15)$$

$$= 1.14285$$

The regression line

$$\boxed{y = 0.543x + 1.1428}$$

When $x = 15$,

$$y = 0.543 \times 15 - 1.1428$$

$$\boxed{y = 9.2857}$$

HWeek

Linear Regression on Kaggle insurance dataset

In [16]:
```
import pandas  as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.figsize'] = [8,5]
plt.rcParams['font.size'] =14
plt.rcParams['font.weight']= 'bold'
plt.style.use('seaborn-whitegrid')
```

In [17]:
```
df = pd.read_csv('insurance.csv')
print('\nNumber of rows and columns in the data set: ',df.shape)
print('')

df.head()
```

```
Number of rows and columns in the data set:  (1338, 7)
```

Out[17]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

In [18]:
```
df.describe()
```

Out[18]:

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

In [19]:
```
df.groupby('children').agg(['mean','min','max'])['charges']
```

Out[19]:

children	mean	min	max
0	12365.975602	1121.8739	63770.42801
1	12731.171832	1711.0268	58571.07448
2	15073.563734	2304.0022	49577.66240
3	15355.318367	3443.0640	60021.39897
4	13850.656311	4504.6624	40182.24600
5	8786.035247	4687.7970	19023.26000

Hitesh

In [20]:
```
categorical_columns = ['sex','children', 'smoker', 'region']
df_encode = pd.get_dummies(data = df, prefix = 'OHE', prefix_sep='_',
               columns = categorical_columns,
               drop_first =True,
              dtype='int8')
```

In [21]:
```
print('Columns in original data frame:\n',df.columns.values)
print('\nNumber of rows and columns in the dataset:',df.shape)
print('\nColumns in data frame after encoding dummy variable:\n',df_encode.columns.values)
print('\nNumber of rows and columns in the dataset:',df_encode.shape)
```

```
Columns in original data frame:
 ['age' 'sex' 'bmi' 'children' 'smoker' 'region' 'charges']

Number of rows and columns in the dataset: (1338, 7)

Columns in data frame after encoding dummy variable:
 ['age' 'bmi' 'charges' 'OHE_male' 'OHE_1' 'OHE_2' 'OHE_3' 'OHE_4' 'OHE_5'
 'OHE_yes' 'OHE_northwest' 'OHE_southeast' 'OHE_southwest']

Number of rows and columns in the dataset: (1338, 13)
```

In [22]:
```
from scipy.stats import boxcox
y_bc,lam, ci= boxcox(df_encode['charges'],alpha=0.05)

ci,lam
```

Out[22]:
```
((-0.01140290617294196, 0.09856968597675454), 0.043649053770664956)
```

In [23]:
```
df_encode['charges'] = np.log(df_encode['charges'])
```

In [24]:
```
from sklearn.model_selection import train_test_split
X = df_encode.drop('charges',axis=1) # Independet variable
y = df_encode['charges'] # dependent variable

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3,random_state=23)
```

In [25]:
```
X_train_0 = np.c_[np.ones((X_train.shape[0],1)),X_train]
X_test_0 = np.c_[np.ones((X_test.shape[0],1)),X_test]

theta = np.matmul(np.linalg.inv( np.matmul(X_train_0.T,X_train_0) ), np.matmul(X_train_0.T,y_train))
```

In [26]:
```
parameter = ['theta_'+str(i) for i in range(X_train_0.shape[1])]
columns = ['intersect:x_0=1'] + list(X.columns.values)
parameter_df = pd.DataFrame({'Parameter':parameter,'Columns':columns,'theta':theta})
```

In [27]:
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X_train,y_train) # Note: x_0 =1 is no need to add, sklearn will take care of it.

sk_theta = [lin_reg.intercept_]+list(lin_reg.coef_)
parameter_df = parameter_df.join(pd.Series(sk_theta, name='Sklearn_theta'))
parameter_df
```

Out[27]:

	Parameter	Columns	theta	Sklearn_theta
0	theta_0	intersect:x_0=1	7.059171	7.059171
1	theta_1	age	0.033134	0.033134
2	theta_2	bmi	0.013517	0.013517
3	theta_3	OHE_male	-0.067767	-0.067767
4	theta_4	OHE_1	0.149457	0.149457
5	theta_5	OHE_2	0.272919	0.272919
6	theta_6	OHE_3	0.244095	0.244095
7	theta_7	OHE_4	0.523339	0.523339
8	theta_8	OHE_5	0.466030	0.466030
9	theta_9	OHE_yes	1.550481	1.550481
10	theta_10	OHE_northwest	-0.055845	-0.055845
11	theta_11	OHE_southeast	-0.146578	-0.146578
12	theta_12	OHE_southwest	-0.133508	-0.133508

```
In [28]: y_pred_norm =  np.matmul(X_test_0,theta)

         J_mse = np.sum((y_pred_norm - y_test)**2)/ X_test_0.shape[0]

         sse = np.sum((y_pred_norm - y_test)**2)
         sst = np.sum((y_test - y_test.mean())**2)
         R_square = 1 - (sse/sst)
         print('The Mean Square Error(MSE) or J(theta) is: ',J_mse)
         print('R square obtain for normal equation method is :',R_square)

         The Mean Square Error(MSE) or J(theta) is:  0.1872962232298182
         R square obtain for normal equation method is : 0.7795687545055328

In [29]: y_pred_sk = lin_reg.predict(X_test)

         from sklearn.metrics import mean_squared_error
         J_mse_sk = mean_squared_error(y_pred_sk, y_test)

         R_square_sk = lin_reg.score(X_test,y_test)
         print('The Mean Square Error(MSE) or J(theta) is: ',J_mse_sk)
         print('R square obtain for scikit learn library is :',R_square_sk)

         The Mean Square Error(MSE) or J(theta) is:  0.18729622322981898
         R square obtain for scikit learn library is : 0.7795687545055318
```

Hitesh

⑦ Classification dataset of diabetes from Kaggle.

```
In [11]: #Load the necessary python libraries
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         plt.style.use('ggplot')
         import warnings
         warnings.filterwarnings('ignore')
```

```
In [12]: df = pd.read_csv('diabetes.csv')
         df.head()
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
In [13]: df.shape

Out[13]: (768, 9)
```

```
In [14]: X = df.drop('Outcome',axis=1).values
         y = df['Outcome'].values
```

```
In [26]: from sklearn.model_selection import train_test_split
         X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.4,random_state=42, stratify=y)
```

```
In [17]: from sklearn.neighbors import KNeighborsClassifier

         neighbors = np.arange(1,9)
         train_accuracy =np.empty(len(neighbors))
         test_accuracy = np.empty(len(neighbors))

         for i,k in enumerate(neighbors):
             knn = KNeighborsClassifier(n_neighbors=k)

             knn.fit(X_train, y_train)

             train_accuracy[i] = knn.score(X_train, y_train)

             test_accuracy[i] = knn.score(X_test, y_test)
```

```
In [27]: knn = KNeighborsClassifier(n_neighbors=7)
         knn.fit(X_train,y_train)

Out[27]: KNeighborsClassifier(n_neighbors=7)
```

```
In [21]: knn.score(X_test,y_test)

Out[21]: 0.7305194805194806
```

```
In [23]: from sklearn.metrics import confusion_matrix
         y_pred = knn.predict(X_test)
         confusion_matrix(y_test,y_pred)

Out[23]: array([[165,  36],
                [ 47,  60]], dtype=int64)
```

```
In [24]: pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'], margins=True)
```

Out[24]:

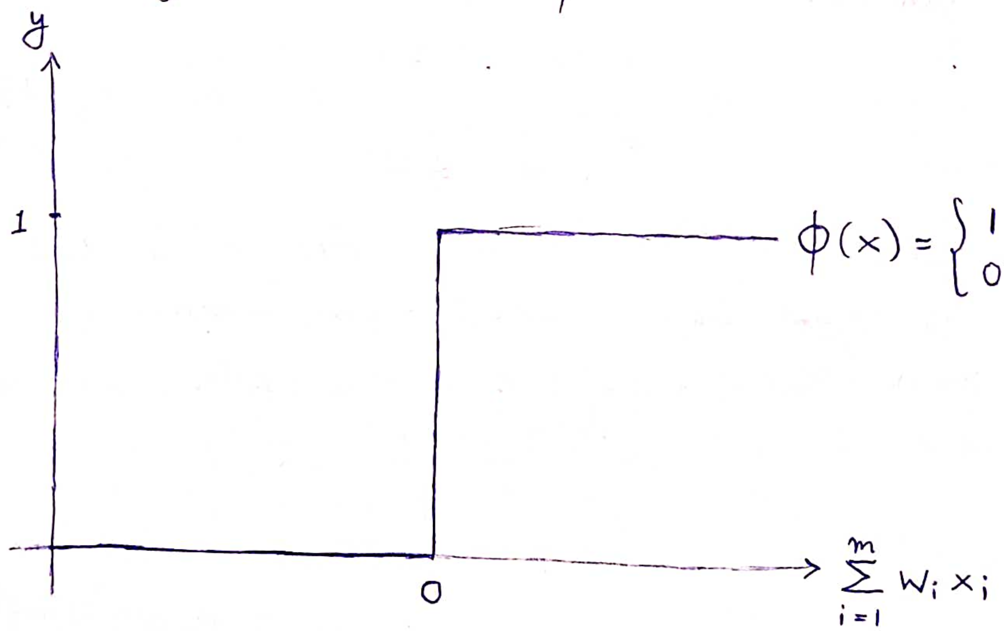
Predicted	0	1	All
True			
0	165	36	201
1	47	60	107
All	212	96	308

```
In [25]: from sklearn.metrics import classification_report
         print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.78      0.82      0.80       201
           1       0.62      0.56      0.59       107

    accuracy                           0.73       308
   macro avg       0.70      0.69      0.70       308
weighted avg       0.73      0.73      0.73       308
```

⑧ In Artificial Neural Network, the value of net input can be anything from -inf to +inf. The neuron doesn't really know how to bound to value and thus is not able to decide the firing pattern. An activation function results in an output signal only when an input signal exceeding a specific threshold value comes as an input. It is similar to the biological neuron which transmits the signal only when the total input signal meets the firing threshold. Different types of activation functions for firing a neuron are -

1) Threshold / Step Function

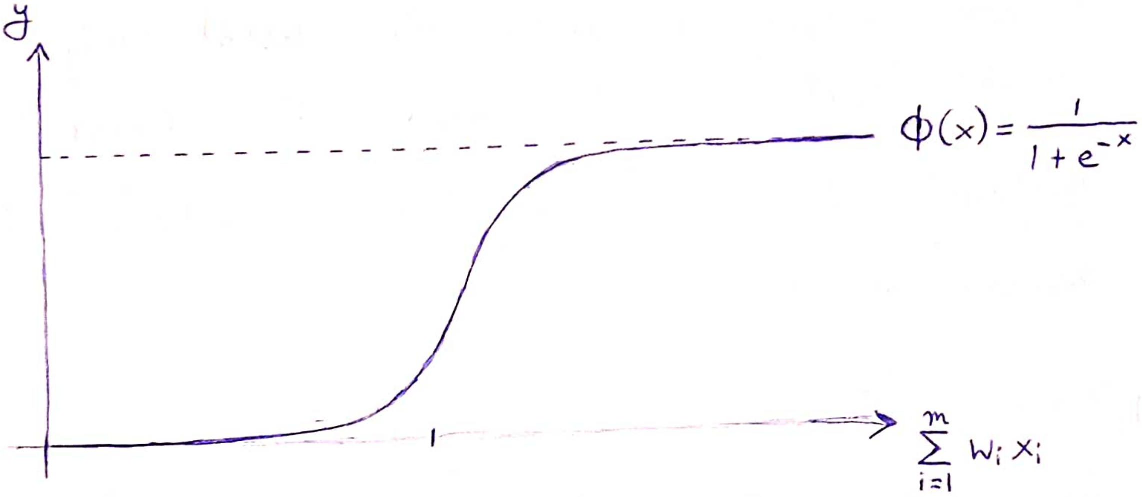
It is a commonly used activation function. It gives 1 as output of the input either 0 or positive. If the input is negative, it gives 0 as output.



$$\phi(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

2) Sigmoid Function

The need for sigmoid function stems from the fact that many learning algorithms require the activation function to be differentiable and hence continuous. The biggest advantage is that it is non-linear. It can be used when predicting probabilities. The function ranges from 0 to 1 having an S-shape. It is defined as $\frac{1}{1+e^{-x}}$

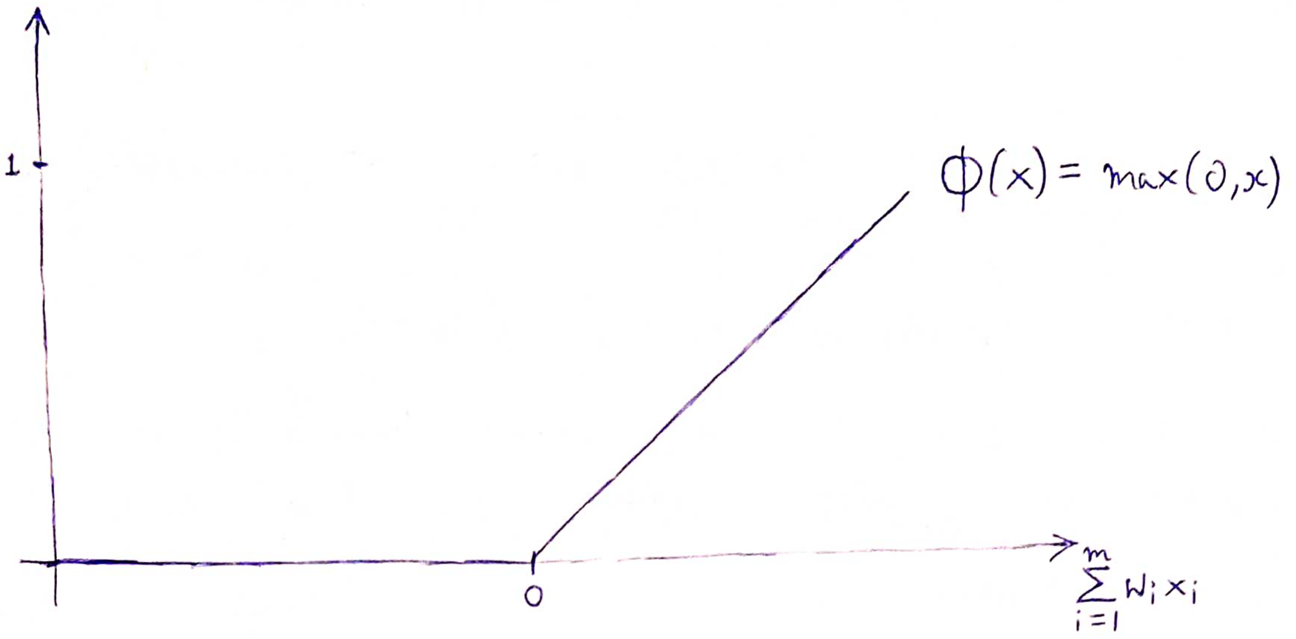


3) ReLu (or Rectifier) Function

ReLu function is the Rectified Linear Unit. It is derivied as

$$f(x) = \max(0, x)$$
$$= \begin{cases} x, & x \geq 0 \\ 0, & x \leq 0 \end{cases}$$

This means that $f(x)$ is zero when x is less than zero and $f(x)$ is equal to x when x is above or equal to zero. The main advantage of using the ReLu function over others is that it does not activate all the neurons at the same time.

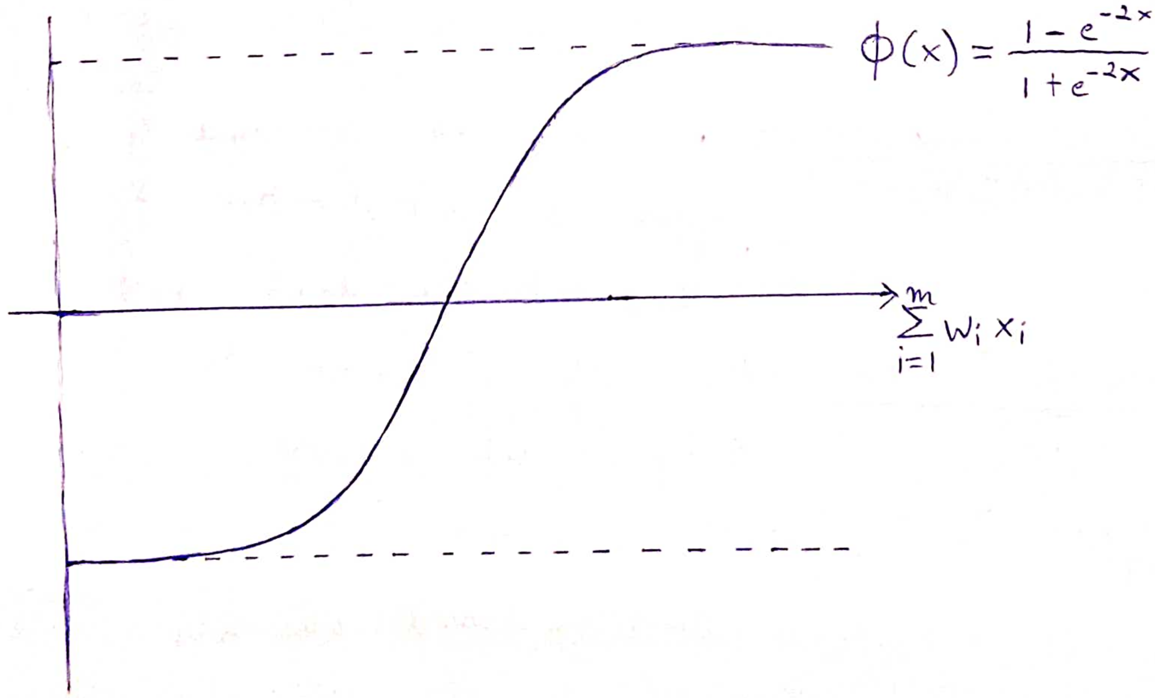


4) Hyperbolic Tangent Function

It is bipolar in nature. It is a widely adopted activation function for a special type of neural network known as Backpropagation Network. It is of the form of

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

It is similar to bipolar sigmoid function.



(9) The Perceptron learning algorithm is inspired by the information processing of a single neutral cell called a neuron.
The perceptron receives input from signals from examples of training data that we weight and combined in a linear equation called the activation.

The activation is then transformed into an output value or prediction using a transfer function, such as the step transfer function.

Perceptron consists of –

1) Input – All the feature becomes the input for a perceptron
$[x_1, x_2, x_3 \ldots\ldots, x_n]$

2) Weights – are the values that are computed over the time of training the model.
$[w_1, w_2, w_3, \ldots\ldots\ldots, w_n]$

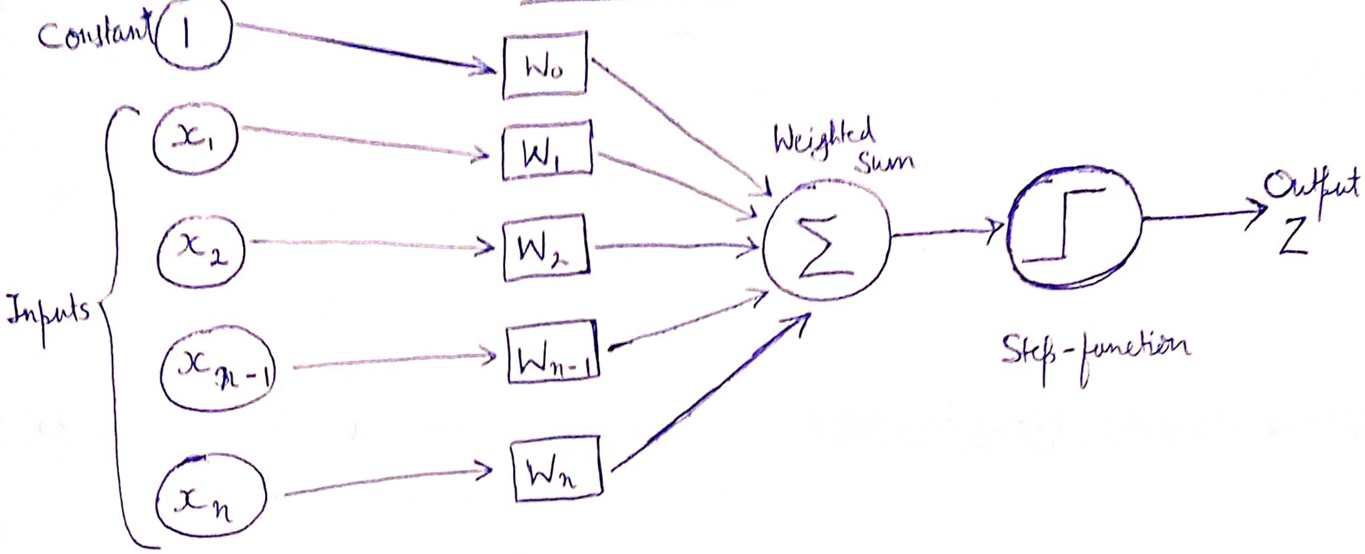
3) Bias – A bias neuron allows a classifier to shift the decision boundary left or right.

4) Weighted Summation – is the sum of value that we get after the multiplication of each weight $[w_n$ associated the each feature value $[x_n]$.

5) Activation Function – the role of activation function is to make neural networks non-linear.

6) Output – The weighted summation is passed to the step/activation function and whatever value we get after compution is the predicted output.

PERCEPTRON



$$Z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} W_i x_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} W_i x_i < \theta \end{cases}$$

Z - output
x - input
W - weights
n - no. of inputs
θ - threshold for step function

The weights of the perceptron algorithm must be estimated from your training data using stochastic gradient descent.

(10) A loss function is a function that compares the target and predicted output values, measures how well the neural network models the training data.
When training, we aim to minimize this loss between the predicted and target outputs.

The 2 major types of loss functions are -

1) Regression Loss Functions - used in regression neural networks
E.g. Mean Squared Error, Mean Absolute Error

2) Classification Loss Function - used in classification neural networks
E.g. Binary cross-Entropy, Categorical Cross-Entropy.

Various Loss functions in neural networks are-

1) Mean Squared Error (MSE)

MSE finds the average of the squared differences between the target and predicted outputs.

$$\boxed{MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2}$$

The difference is squared, which means it does not matter whether the predicted value is above or below the target value; however values with a large error are penalized. MSE is also a convex function with its clearly defined global minimum.
One disadvantage is that it is very sensitive to outliers.

2) Mean Absolute Error (MAE)

MAE finds the average of the absolute differences between the target and the predicted outputs.

$$\boxed{MAE = \frac{1}{n} \sum_{i=1}^{n} \left| y^{(i)} - \hat{y}^{(i)} \right|}$$

MAE is used in cases when the training data has a large number of outliers to mitigate the over-sensitivity to outliers (like in case of MSE).

Its disadvantage is that as the average distance approaches 0, gradient descent optimization will not work, as the function's derivative at 0 is undefined.

3) Binary Cross Entropy / Log loss

It is a loss function in binary classification models.

$$\boxed{CE\ Loss = \frac{1}{n} \sum_{i=1}^{n} -(y_i \cdot \log(p_i)) + (1-y_i) \cdot \log(1-p))}$$

4) Categorical Cross-Entropy Loss

In cases where the number of classes is greater than 2, we utilize categorical cross-entropy.

$$\boxed{CE\ Loss = -\frac{1}{n} \sum_{i=1}^{N} \sum_{j=1}^{N} y_{ij} \cdot \log(p_{ij})}$$