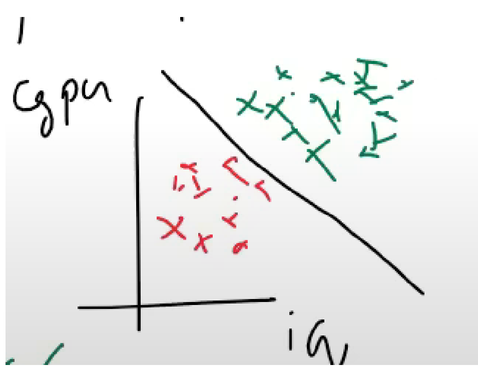
Logistic Regression(VIMP for Interviews)

- **Purpose**: A machine learning algorithm used for **binary classification** (predicting one of two classes, e.g., `0` or `1`).
- **Example**:
    - Predicting if an email is **spam (1)** or **not spam (0)**.
    - Predicting if a patient has a **disease (1)** or is **healthy (0)**.

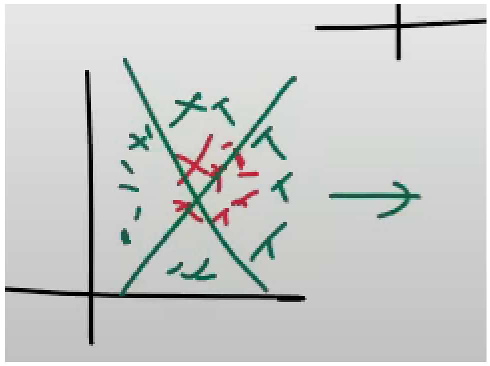
Uses a **sigmoid function** to map predictions to probabilities between `0` and `1`.

Assumption

- The data should be linear or sort of linear.



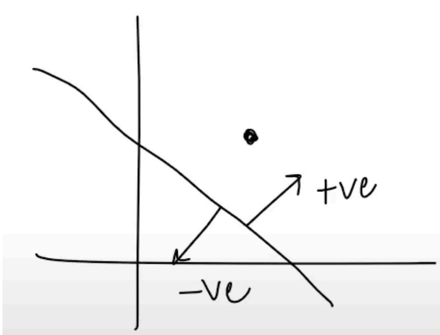
- It won't work on circular data like this:



Line Equation:

$$\boxed{Ax + By + c = 0}$$

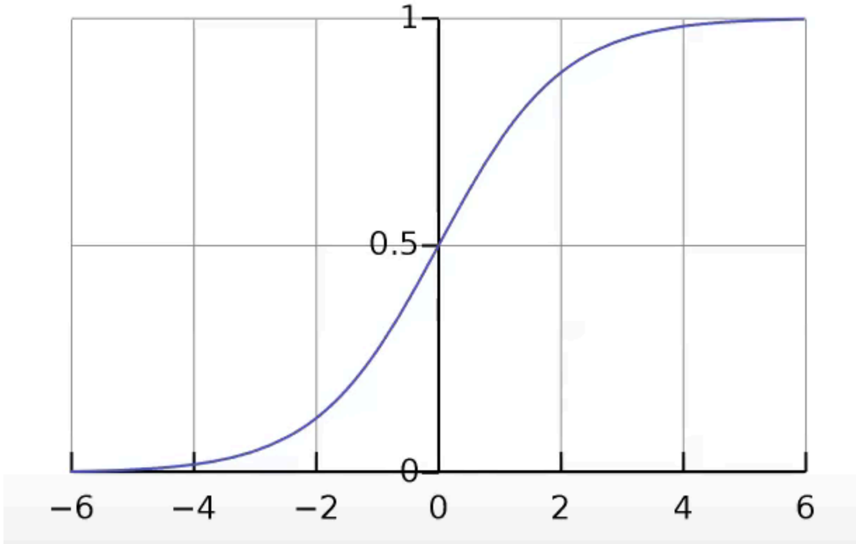
How to find out if a given point is on the positive side of the line or the negative side of the line?



$Ax_1 + By_1 + C > 0 \rightarrow$ +ve region

$Ax_1 + By_1 + C < 0 \rightarrow$ -ve region

Sigmoid Function



$$P(y=1) = \frac{1}{1+e^{-z}}$$

- z: A linear combination of features and weights.
- e: Euler's number (~2.718).
- **Shape**: An S-shaped curve that maps any input to a value between `0` and `1`.

Linear Equation:

$$z = b_0 + b_1 \cdot x_1 + b_2 \cdot x_2 + \ldots + b_n \cdot x_n$$

where:

- $b_0, b_1, \ldots, b_n$ are the model parameters (weights).
- $x_1, x_2, \ldots, x_n$ are the input features.

Cost Function:

Instead of using the mean squared error (like in linear regression), logistic regression uses a cost function called the **log loss (binary cross-entropy) / Maximum Likelihood**:

$$\text{Log Loss} = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

Where:

- y_i is the true label for the i-th sample (either 0 or 1).
- p_i is the predicted probability of the positive class (class 1) for the i-th sample.
- m is the total number of samples.

Optimization:

The model's coefficients are learned by minimizing the cost function.

- **Gradient Descent** is often used: the algorithm adjusts the coefficients iteratively to reduce the cost.
- For each coefficient, the gradient (derivative) is computed, and the coefficient is updated in the direction that minimizes the cost.

Prediction:

If $P(y = 1) \geq 0.5$, predict `1`.

If $P(y = 1) < 0.5$, predict `0`.

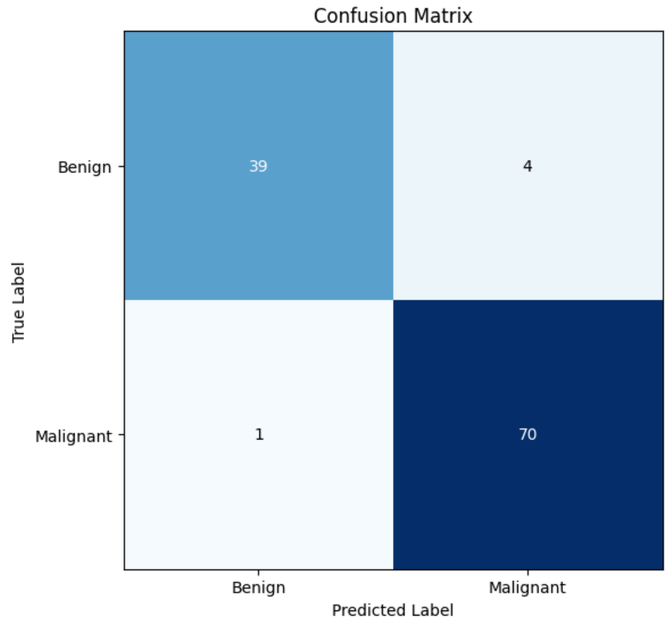
Python Code for Logistic Regression

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data  # Features
y = data.target  # Target labels: 0 = benign, 1 = malignant

# Split the dataset into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat
e=42)

# Train the Logistic Regression model
clf = LogisticRegression(max_iter=10000)
clf.fit(X_train, y_train)

# Predict the classes on the test set
y_pred = clf.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Visualize the confusion matrix
fig, ax = plt.subplots(figsize=(6, 6))
ax.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
ax.set_title('Confusion Matrix')
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
```

```
# Add labels to the matrix
classes = ['Benign', 'Malignant']
tick_marks = np.arange(len(classes))
ax.set_xticks(tick_marks)
ax.set_yticks(tick_marks)
ax.set_xticklabels(classes)
ax.set_yticklabels(classes)

# Add text inside the squares
thresh = cm.max() / 2.
for i, j in np.ndindex(cm.shape):
    ax.text(j, i, format(cm[i, j], 'd'),
            ha="center", va="center",
            color="white" if cm[i, j] > thresh else "black")

plt.tight_layout()
plt.show()
```

Accuracy: 0.96
Confusion Matrix:
[[39 4]
[1 70]]



Key Parameters

Parameter	Description
`C`	Inverse of regularization strength (smaller values = stronger regularization).
`penalty`	Type of regularization (`'l1'` , `'l2'` , or `'none'`).
`solver`	Algorithm to use for optimization (e.g., `'liblinear'` , `'lbfgs'`).

Another Example

- Dataset → IRIS

```
import seaborn as sns
import pandas as pd
import numpy as np
```

```
df=sns.load_dataset('iris')
df.head()
```

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
df['species'].unique()

Output:
array(['setosa', 'versicolor', 'virginica'], dtype=object)
```

- Remove `setosa`

```
df=df[df['species']!='setosa']
```

- Now `species` has only 2 columns.

- Convert them into numbers

```
df['species']=df['species'].map({'versicolor':0,'virginica':1})
```

```
X=df.iloc[:,:-1]
y=df.iloc[:,-1]
```

- **Train-Test-Split**

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)
```

- Apply Logistic Regression

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
```

- For hyperparameter tuning, use → `GridSearchCV`

```
from sklearn.model_selection import GridSearchCV
parameter={'penalty':['l1','l2','elasticnet'],'C':[1,2,3,4,5,6,10,20,30,40,50],'max_iter':[100,200,300]}

classifier_regressor=GridSearchCV(classifier,parameter,scoring='accuracy',cv=5)
```

What is `C`?

- **Definition**: `C` is the **inverse of regularization strength**.
    - Regularization is a technique used to prevent overfitting (when the model learns noise in the training data instead of the underlying pattern).
- **Range**: `C > 0`.
    - Smaller values of `C` mean **stronger regularization**.
    - Larger values of `C` mean **weaker regularization**.

Fit the model:

```
classifier_regressor.fit(X_train,y_train)
```

Best parameters:

```
classifier_regressor.best_params_

Output:
{'C': 1, 'max_iter': 100, 'penalty': 'l2'}
```

```
classifier_regressor.best_score_

Output:
0.9733333333333334
```

Predict:

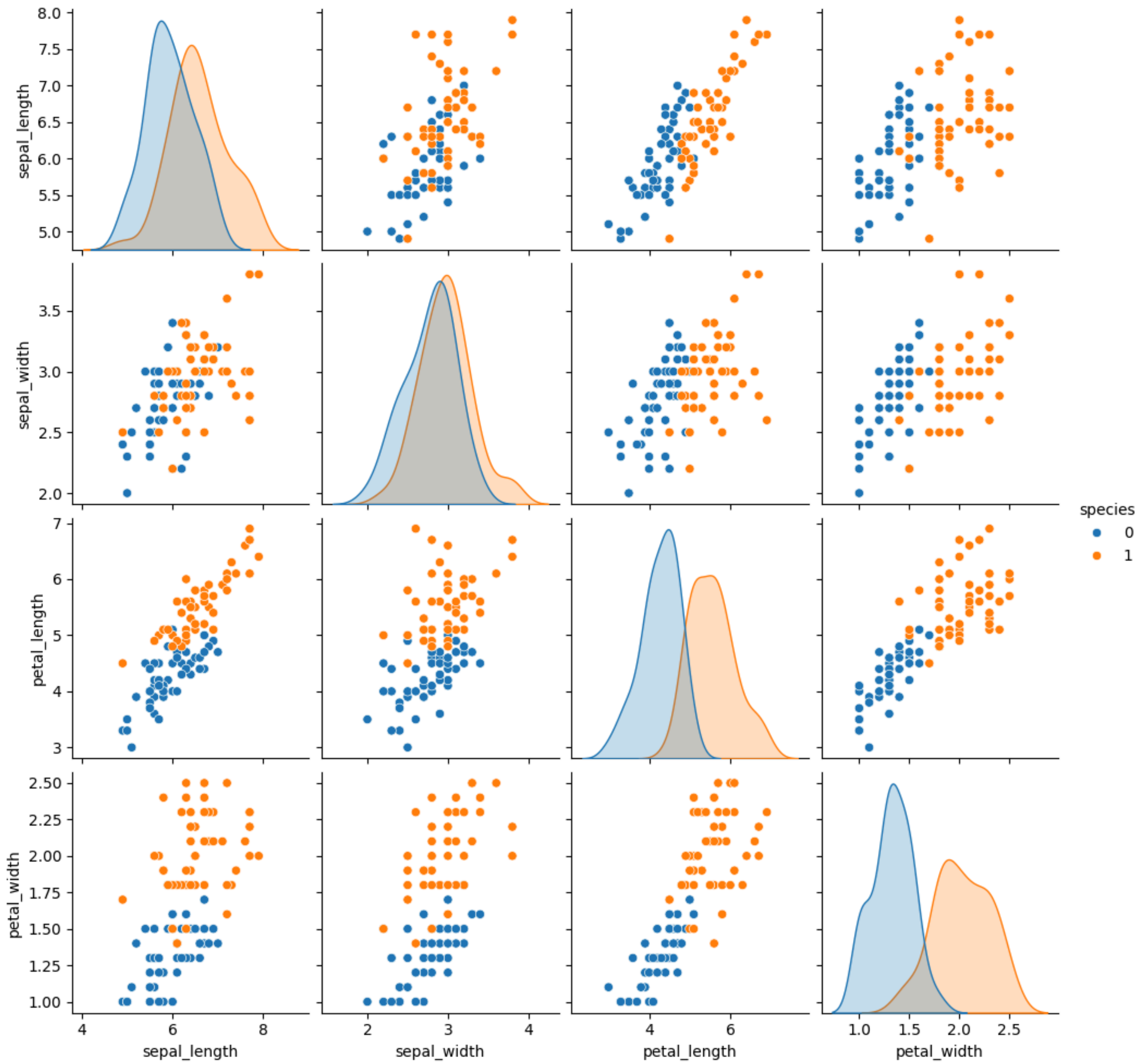
```
y_pred=classifier_regressor.predict(X_test)
```

Find out the accuracy:

```
## accuracy score
from sklearn.metrics import accuracy_score,classification_report

score=accuracy_score(y_pred,y_test)
print(score)

print(classification_report(y_pred,y_test))
```

```
0.92
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        14
           1       0.91      0.91      0.91        11

    accuracy                           0.92        25
   macro avg       0.92      0.92      0.92        25
weighted avg       0.92      0.92      0.92        25
```

```
##EDA
sns.pairplot(df,hue='species')
```




```
df.corr()
```

	sepal_length	sepal_width	petal_length	petal_width	species
sepal_length	1.000000	0.553855	0.828479	0.593709	0.494305
sepal_width	0.553855	1.000000	0.519802	0.566203	0.308080
petal_length	0.828479	0.519802	1.000000	0.823348	0.786424
petal_width	0.593709	0.566203	0.823348	1.000000	0.828129
species	0.494305	0.308080	0.786424	0.828129	1.000000

Advantages of Logistic Regression:

1. **Simplicity**: Logistic Regression is easy to understand and implement.
2. **Interpretable**: The model coefficients are easy to interpret, and they give us insight into the influence of each feature on the predicted probability.
3. **Efficient**: Logistic Regression is computationally efficient, especially for large datasets.
4. **Probabilistic Output**: Logistic Regression provides probabilities as output, which are useful when you want to assess the confidence of predictions.

Limitations of Logistic Regression:

1. **Linear Decision Boundaries**: Logistic Regression can only model linear decision boundaries. It may struggle with problems that require more complex, non-linear boundaries.
2. **Feature Scaling**: Logistic Regression may require feature scaling (standardization) to perform well if the features have different scales.
3. **Assumptions**: Logistic regression assumes that the relationship between the input features and the log-odds of the dependent variable is linear.