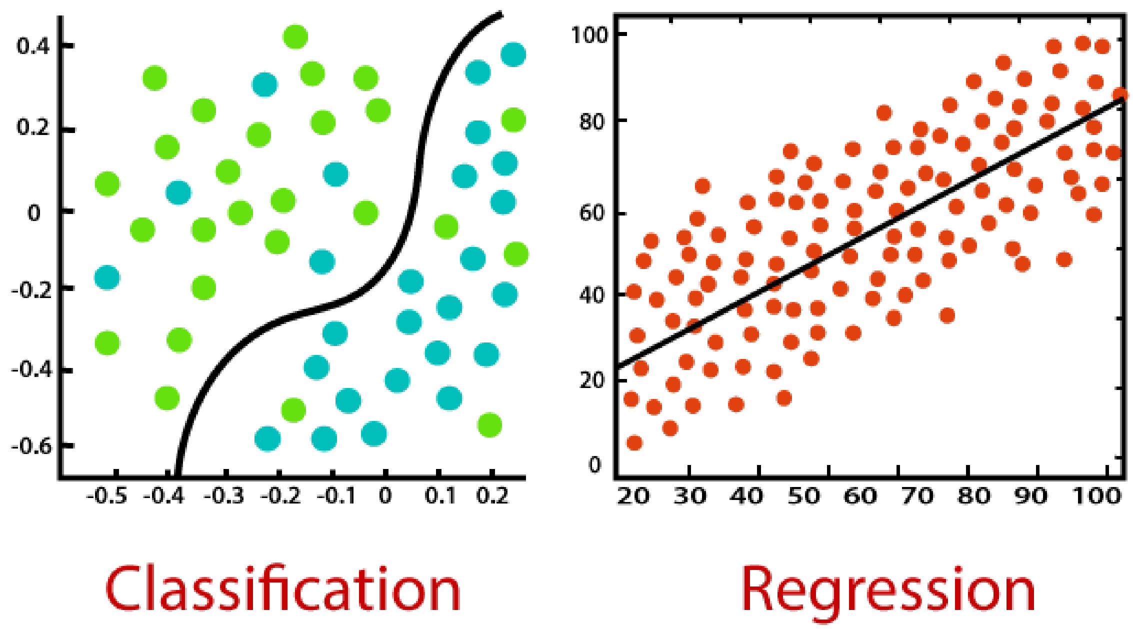
Classification models



A **classification models** are predictive modeling problem where a **class label** is predicted for a given example of input data.

Examples

- Given an email, classify if it is spam or not.
- Given a handwritten character, classify it as one of the known characters.
- Given recent user behavior, classify as churn or not.

Types of classification problems:

- Binary Classification
- Multi-Class Classification

Binary classification:

Refers to those classification tasks that have **two class** labels.

- Email spam detection (spam or not).
- Churn prediction (churn or not).
- Conversion prediction (buy or not).

Typically, binary classification tasks involve:

- A class representing the **normal state**, referred to as the **negative class**, which is usually assigned the **label 0**
  - (e.g., not spam, not churn, not buy).
- A class representing the **abnormal state**, referred to as the **positive class**, which is usually assigned the **label 1**
  - (e.g., spam, churn, buy).

Multi-class classification

Refers to those classification tasks that have more than two class labels.

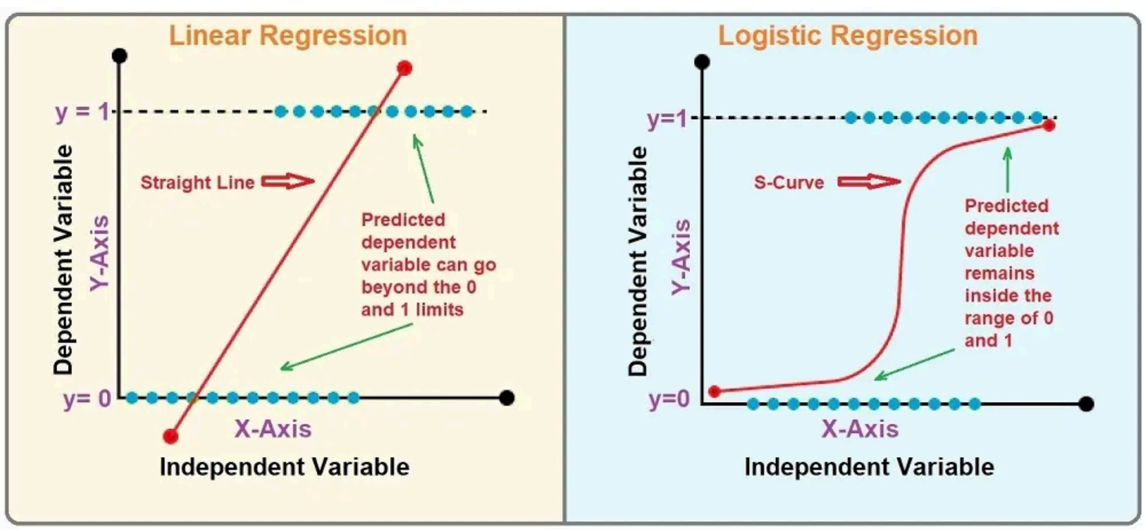
- Face classification.
- Plant species classification.
- Optical character recognition.

We will focus on Binary classification in this course

Classification algorithms

- **Logistic Regression**
- k-Nearest Neighbors
- Decision Trees
- Support Vector Machine
- Naive Bayes
- Random Forest
- Gradient Boosting

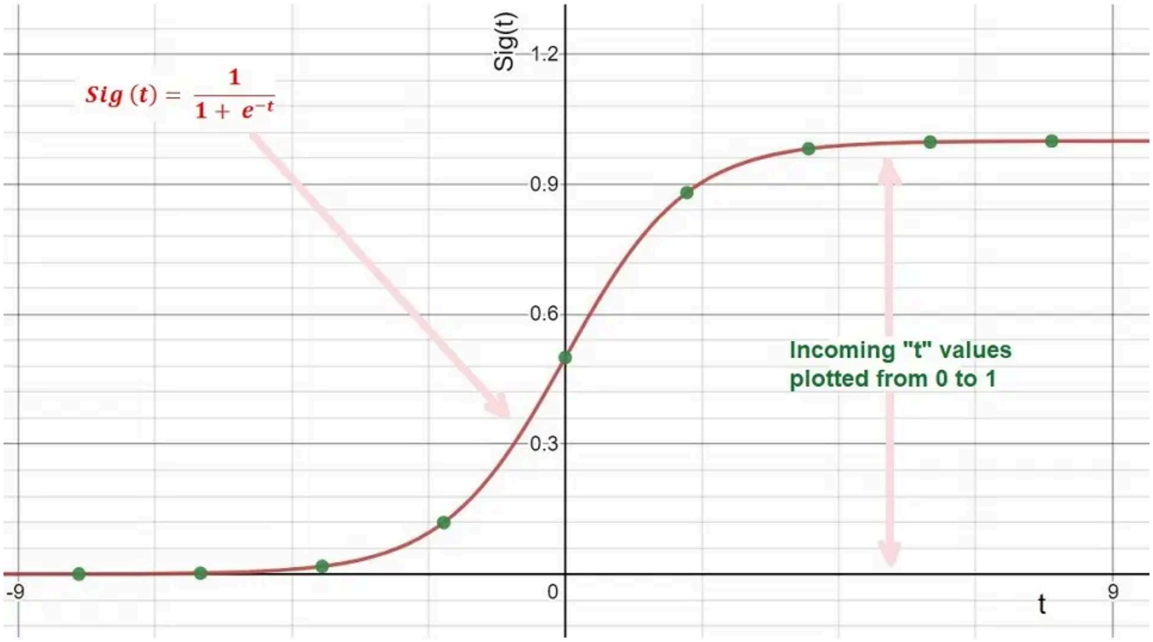
Logistic Regression

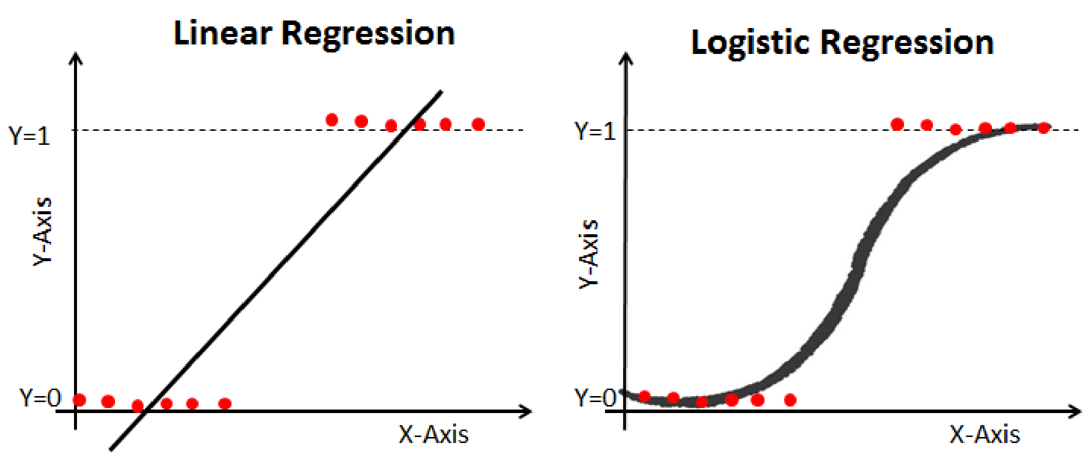


Sigmoid function

The sigmoid function is a mathematical function for mapping predicted values to probabilities. It can map any real value into another value within 0 and 1.

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





$$y = \beta_0 + \beta_1 x$$

$$y = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

Example: Heart disease detection

Read data

In [1]:
```
import pandas as pd

df = pd.read_csv("data_2/heart disease.csv")
df
```

Out[1]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Ex
0	40	M	ATA	140	289	0	Normal	172	
1	49	F	NAP	160	180	0	Normal	156	
2	37	M	ATA	130	283	0	ST	98	
3	48	F	ASY	138	214	0	Normal	108	
4	54	M	NAP	150	195	0	Normal	122	
...	...	...	...	...	...	...	...	...	
913	45	M	TA	110	264	0	Normal	132	
914	68	M	ASY	144	193	1	Normal	141	
915	57	M	ASY	130	131	0	Normal	115	
916	57	F	ATA	130	236	0	LVH	174	
917	38	M	NAP	138	175	0	Normal	173	

918 rows × 12 columns

Split data to train and test

In [2]:
```
from sklearn.model_selection import train_test_split

X = df.drop('HeartDisease', axis=1)
y = df['HeartDisease']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_train
```

Out[2]:

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	Ex
155	56	M	ASY	155	342	1	Normal	150	
362	56	M	NAP	155	0	0	ST	99	
869	59	M	NAP	150	212	1	Normal	157	
101	51	M	ASY	130	179	0	Normal	100	
199	57	F	TA	130	308	0	Normal	98	
...	...	...	...	...	...	...	...	...	
106	48	F	ASY	120	254	0	ST	110	
270	45	M	ASY	120	225	0	Normal	140	
860	60	M	ASY	130	253	0	Normal	144	
435	60	M	ASY	152	0	0	ST	118	
102	40	F	ASY	150	392	0	Normal	130	

688 rows × 11 columns

In [3]:

```
y
```

Out[3]:

```
0      0
1      1
2      0
3      1
4      0
      ..
913    1
914    1
915    1
916    1
917    0
Name: HeartDisease, Length: 918, dtype: int64
```

Data preprocessing

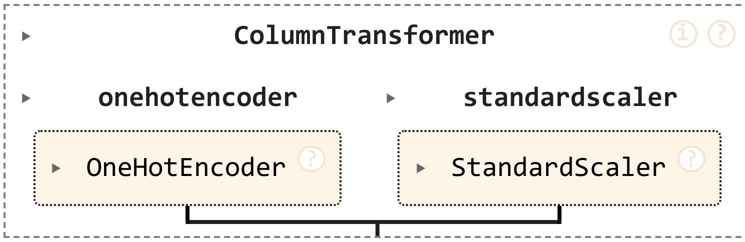
In [4]:

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import make_column_transformer

ohe = OneHotEncoder()
scaler = StandardScaler()

categ_cols =  ['Sex', 'ChestPainType', 'RestingECG', 'ExerciseAngina', 'ST_Slope',
num_cols = ['Age', 'RestingBP', 'Cholesterol', 'MaxHR', 'Oldpeak']
cat_tuple = (ohe, categ_cols)
num_tuple = (scaler, num_cols)
preprocessor = make_column_transformer(cat_tuple, num_tuple)
preprocessor
```

Out[4]:

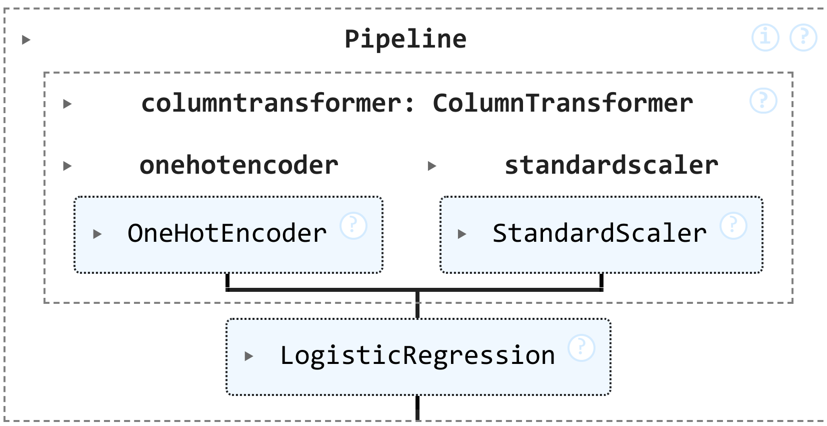


Fit model

In [5]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline

# Create a model pipeline with the preprocessor and a KNN model
model = LogisticRegression()
pipeline = make_pipeline(preprocessor, model)
pipeline.fit(X_train, y_train)
```

Out[5]:



Predict new instances

In [6]:
```
person = [43, 'F', 'TA', 100, 223, 0, 'Normal', 142, 'N', 0.0, 'Up']
```

In [7]:
```
X = pd.DataFrame(columns=X_train.columns, data  = [person])
X = preprocessor.transform(X)

print(model.predict_proba(X))
model.predict(X)
```

```
[[0.97918719 0.02081281]]
```

Out[7]:
```
array([0], dtype=int64)
```

In [8]:
```
person_2 = [65, 'M', 'ASY', 160, 0, 1, 'ST', 122, 'N', 1.2, 'Flat']
```

In [9]:
```
X = pd.DataFrame(columns=X_train.columns, data  = [person_2])
X = preprocessor.transform(X)

print(model.predict_proba(X))
model.predict(X)
```

```
[[0.02499716 0.97500284]]
```

Out[9]:
```
array([1], dtype=int64)
```

Model evaluation

In [10]:
```
test_preds = pipeline.predict(X_test)
test_preds
```

Out[10]:
```
array([0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 0, 0], dtype=int64)
```

How to compare the predicted labels against actual labels ?

In [11]:
```
import numpy as np
print(f"Predicted: {test_preds[:15]}")
print(f"Actual    : {np.array(y_test[:15])}")
```

```
Predicted: [0 0 1 1 0 1 1 0 1 1 1 0 1 0 1]
Actual    : [0 1 1 1 0 1 1 0 1 1 0 0 0 0 1]
```

Categorize every instance in the test set into one the four categories:

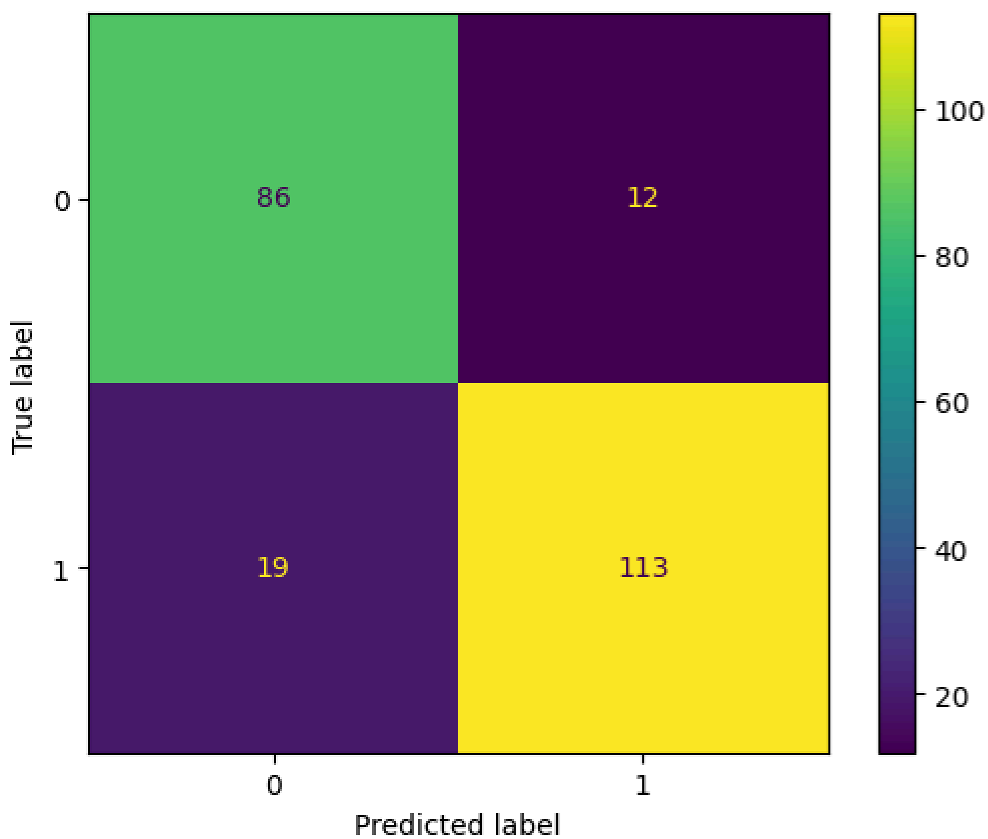
- TP: Actualy positive, and correctly predicted as positive
- TN: Actualy negative, and correctly predicted as negative
- FP: Actualy negative, and incorrectly predicted as positive
- FN: Actualy positive, and incorrectly predicted as negative

Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

In [12]:
```python
from sklearn.metrics import ConfusionMatrixDisplay

ConfusionMatrixDisplay.from_predictions(y_test, test_preds);
```



Is it a good model ?

Evaluation matrics

- Accuracy
- F1_score (Precision, Recall)
- ROC
- Sensitivity
- Specificity

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

In [17]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, test_preds)

y_test.value_counts(normalize=True)
```

Out[17]:
```
HeartDisease
1    0.573913
0    0.426087
Name: proportion, dtype: float64
```

- **Pros**:
  - Accuracy is easy to understand
- **Cons**:
  - Misleading in unbalanced data
  - Does not give specific information about the kinds of errors that a model is making.

Recall:

Out of all of the samples that belong to the positive class, what proportion/percentage did my model classify correctly?

$$\text{Recall} = \frac{TP}{TP + FN}$$

In [18]:
```
from sklearn.metrics import recall_score
recall_score(y_test, test_preds)
```

Out[18]:
```
0.8560606060606061
```

Precision

When our model predicts the class to be the positive class, how often is it correct?

$$\text{Precision} = \frac{TP}{TP + FP}$$

In [19]:
```python
from sklearn.metrics import precision_score
precision_score(y_test, test_preds)
```

Out[19]:
```
0.904
```

F1-score

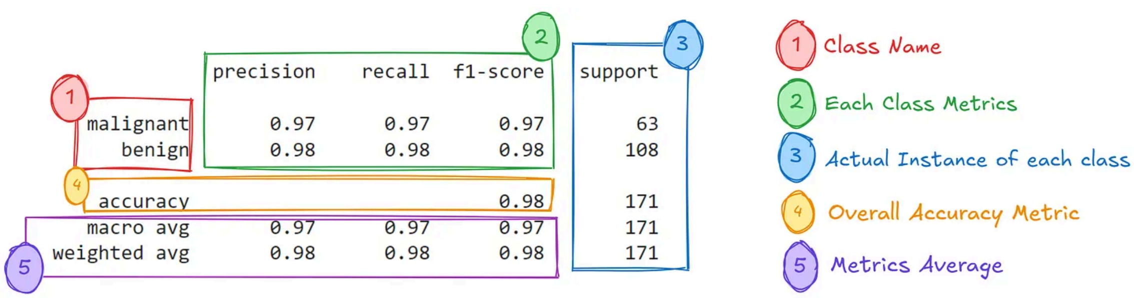
The harmonic mean of the precision and recall scores

$$\text{F1-Score} = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

In [21]:
```python
from sklearn.metrics import f1_score
f1_score(y_test, test_preds)
```

Out[21]:
```
0.8793774319066148
```

Classification report as summary to all previous metrices



In [22]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test, test_preds))
```

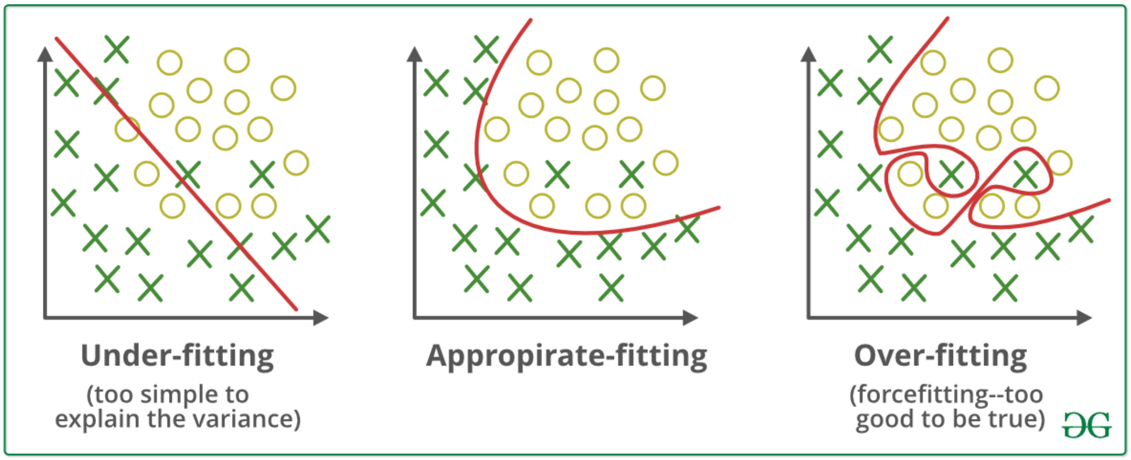
```
              precision    recall  f1-score   support

           0       0.82      0.88      0.85        98
           1       0.90      0.86      0.88       132

    accuracy                           0.87       230
   macro avg       0.86      0.87      0.86       230
weighted avg       0.87      0.87      0.87       230
```

Underfit and Overfit in Classification

Bias and Variance on LR



How to detect underfit ?

Low (unacceptable) value for the relevant metric (Accuracy, recall,...)

How to detect overfit ?

1. **Evaluate the model on both the training and testing data, then**
2. **If there is a significant gap between both quality => Overfit**

Example : How you interpret the quality of a model predicting students pass/not pass data science course ?

Classification report: Train data precision recall f1-score support 0 0.43 0.42 0.42 312 1 0.53 0.55 0.54 376 accuracy 0.49 688 macro avg 0.48 0.48 0.48 688 weighted avg 0.49 0.49 0.49 688 ### Classification report: Test data precision recall f1-score support 0 0.44 0.47 0.45 98 1 0.58 0.55 0.57 132 accuracy 0.52 230 macro avg 0.51 0.51 0.51 230 weighted avg 0.52 0.52 0.52 230

Underfit since the quality is low in both train and test

Classification report: Train data precision recall f1-score support 0 0.87 0.84 0.85 312 1 0.87 0.90 0.88 376 accuracy 0.87 688 macro avg 0.87 0.87 0.87 688 weighted avg 0.87 0.87 0.87 688 ### Classification report: Test data precision recall f1-score support 0 0.40 0.43 0.41 98 1 0.55 0.52 0.54 132 accuracy 0.48 230 macro avg 0.48 0.48 0.48 230 weighted avg 0.49 0.48 0.48 230

Overfit since there is a significant gap between train and test quality metrics

Classification report: Train data precision recall f1-score support 0 0.87 0.84 0.85 312 1 0.87 0.90 0.88 376 accuracy 0.87 688 macro avg 0.87 0.87 0.87 688 weighted avg 0.87 0.87 0.87 688 ### Classification report: Test data precision recall f1-score support 0 0.82 0.88 0.85 98 1 0.90 0.86 0.88 132 accuracy 0.87 230 macro avg 0.86 0.87 0.86 230 weighted avg 0.87 0.87 0.87 230

Good model; both train and test are acceptable and no gap exist