Introduction to Machine Learning

Ass. Prof. Sara El-Metwally
Computer Science Department,
Faculty of Computers and Information,
Mansoura University, Egypt.

sarah_almetwally4@mans.edu.eg

Agenda

01 Logistic Regression

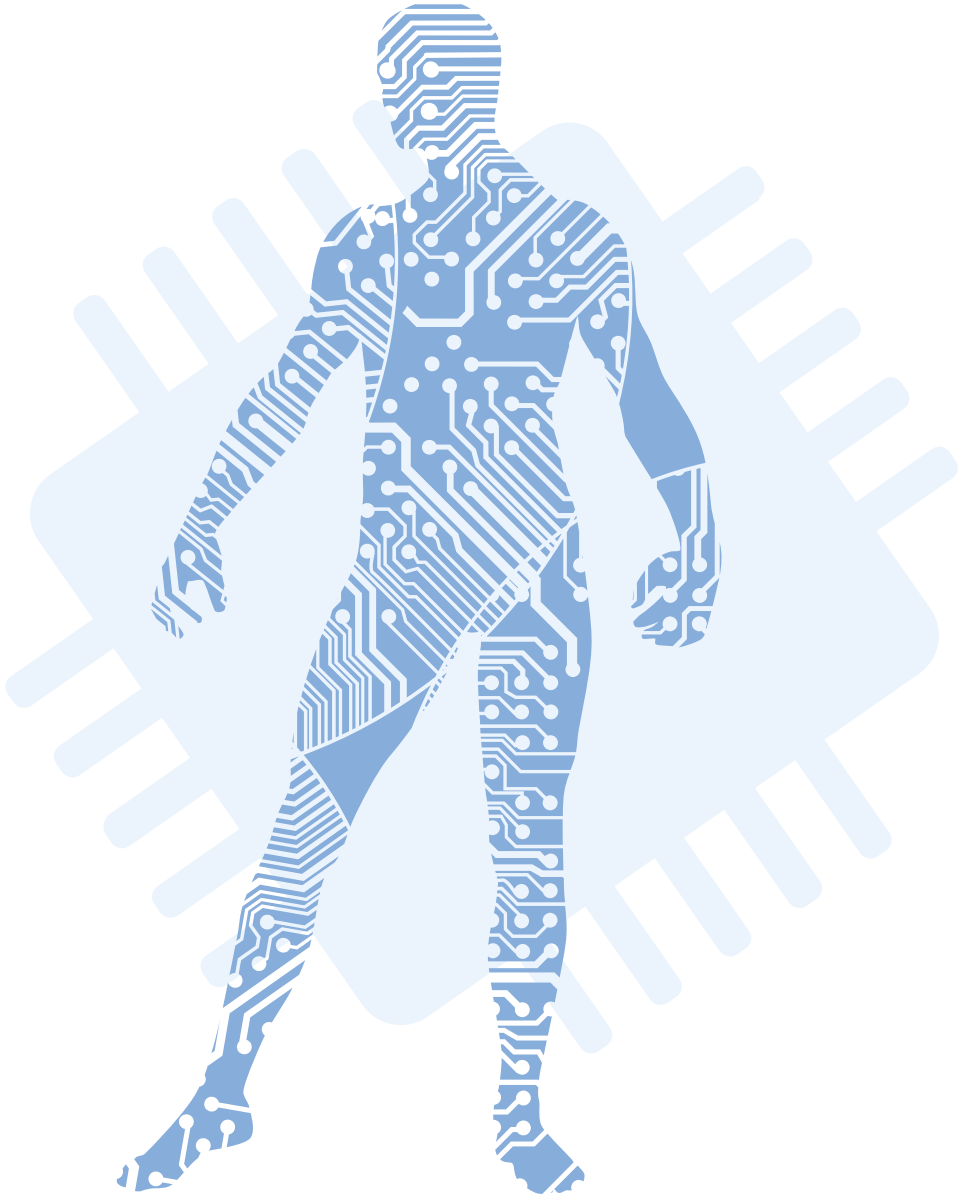
02 Logistic Regression Loss

03 KNN Algorithm

04 Distance Metrics

05 Examples

06 Applications



Logistic Regression

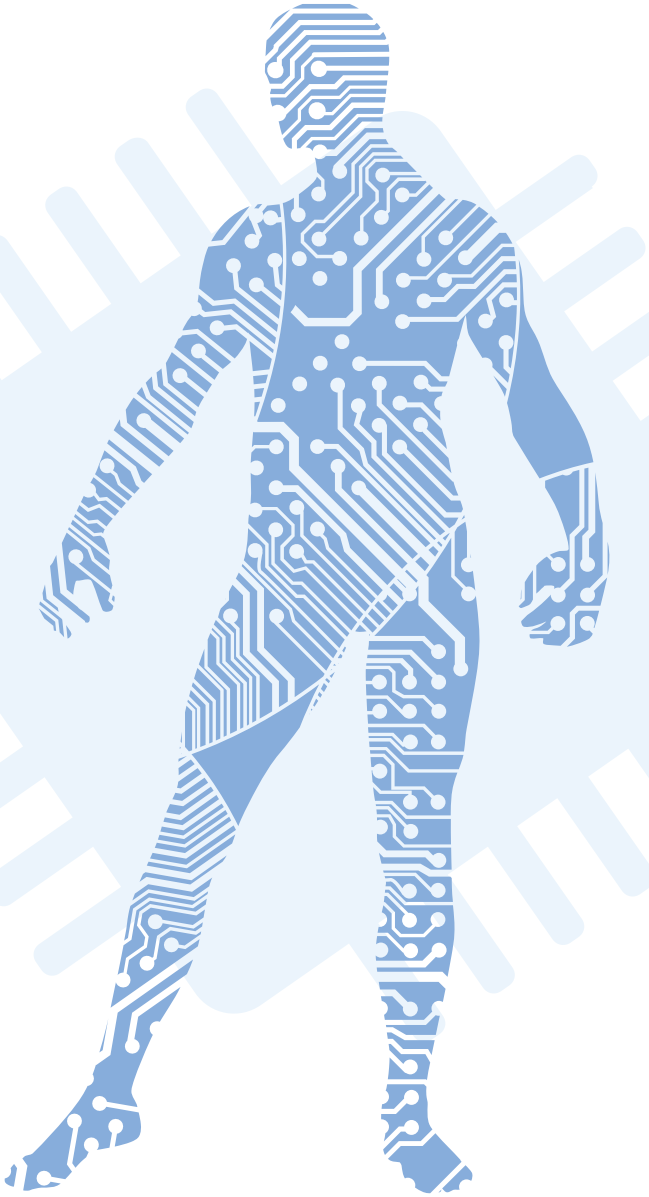
❑ In the Linear regression module, you explored how to construct a model to make continuous numerical predictions, such as the fuel efficiency of a car. But what if you want to build a model to answer questions like "Will it rain today?" or "Is this email spam?"

❑This module introduces a new type of regression model called logistic regression that is designed to predict the probability of a given outcome.

❑ Many problems require a probability estimate as output. Logistic regression is an extremely efficient mechanism for calculating probabilities.

❑Applied "as is." For example, if a spam-prediction model takes an email as input and outputs a value of 0.932, this implies a 93.2% probability that the email is spam.

❑ Converted to a binary category such as True or False, Spam or Not Spam.



Logistic Regression

❑ The standard logistic function, also known as the sigmoid function (sigmoid means "s-shaped"), has the formula:

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

Figure 1 shows the corresponding graph of the sigmoid function.

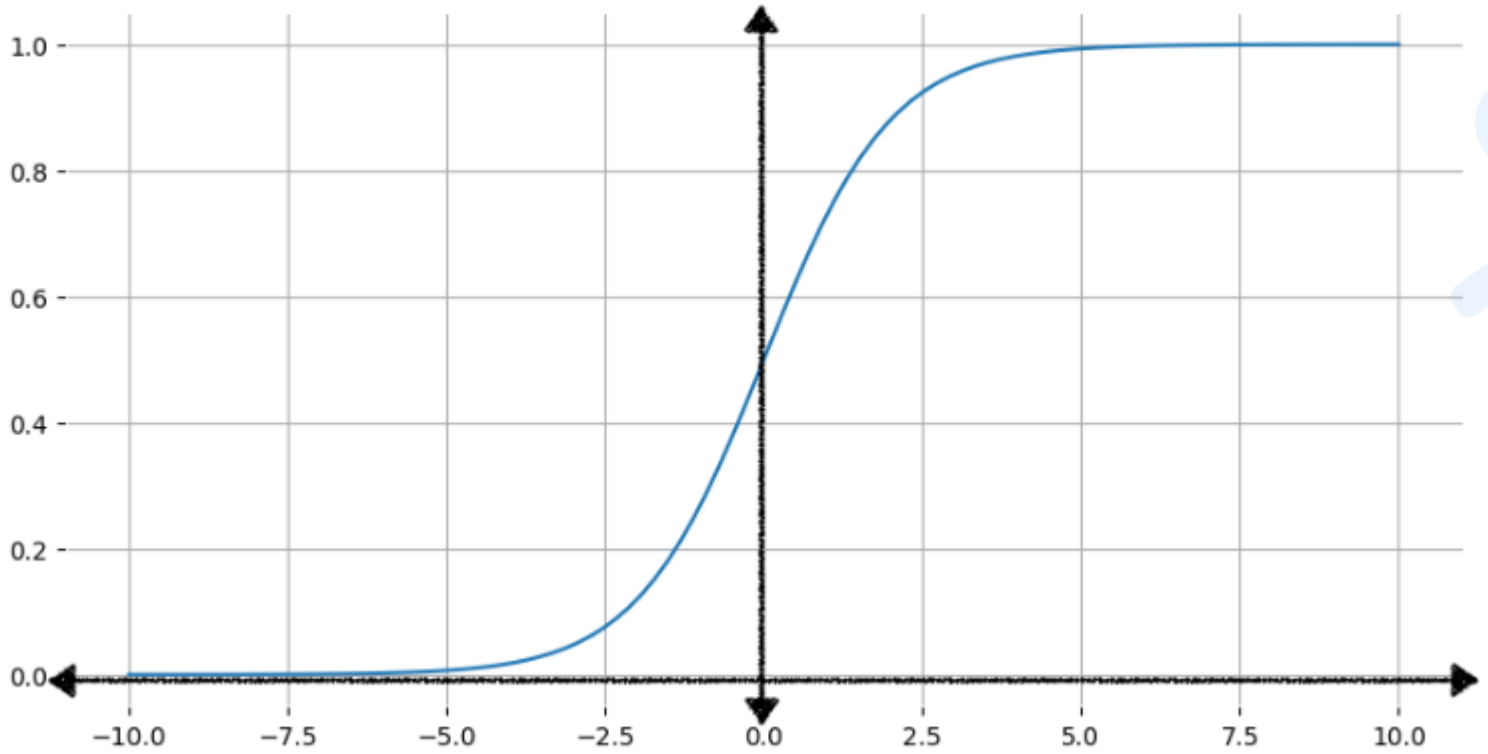


Figure 1. *Graph of the sigmoid function. The curve approaches 0 as x values decrease to negative infinity, and 1 as x values increase toward infinity.*

Logistic Regression

- As the input, x, increases, the output of the sigmoid function approaches but never reaches 1. Similarly, as the input decreases, the sigmoid function's output approaches but never reaches 0.

- The following equation represents the linear component of a logistic regression model:

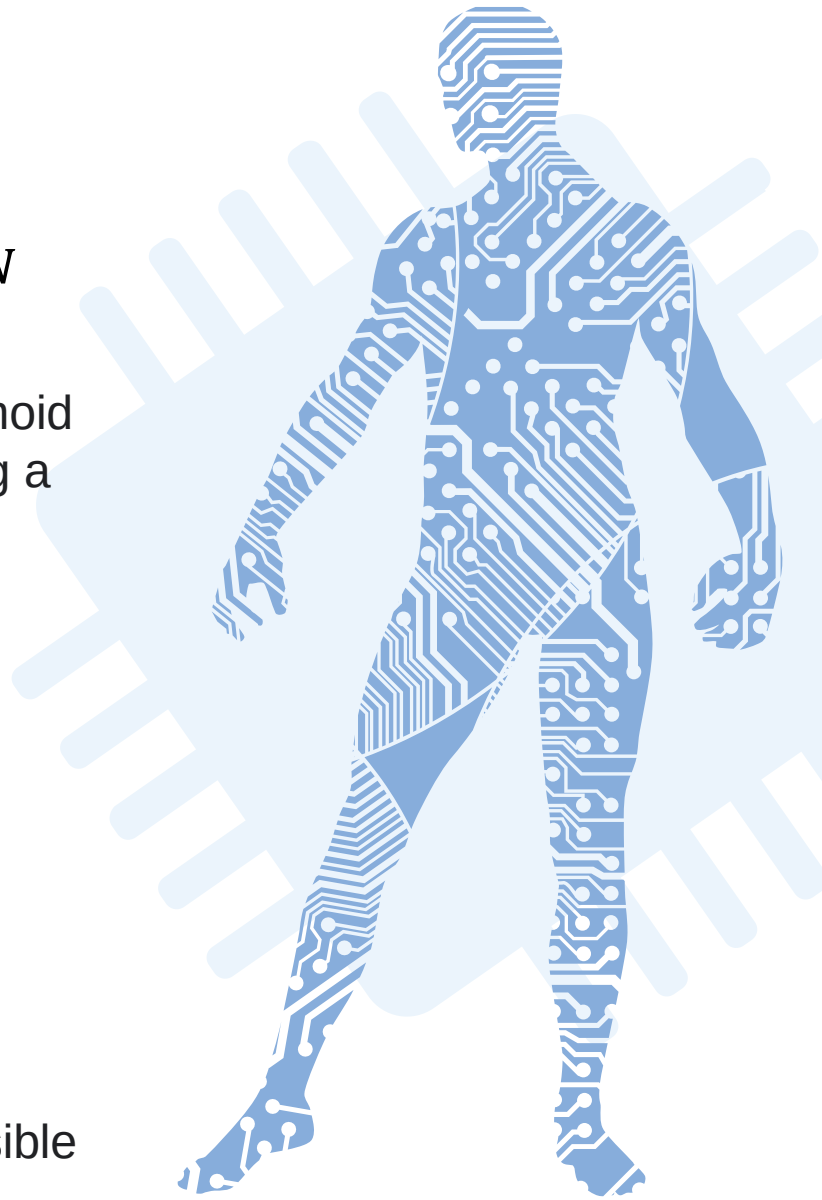
$$z = b + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots + w_Nx_N$$

- To obtain the logistic regression prediction y', the z value is then passed to the sigmoid function, yielding a value (a probability) between 0 and 1:

$$y' = \frac{1}{1 + e^{-z}}$$

Logistic Regression

$$z = b + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + \cdots + w_Nx_N$$



z is referred to as the *log-odds* because if you start with the following sigmoid function (where y is the output of a logistic regression model, representing a probability):

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

And then solve for *z*:

$$z = \log\left(\frac{y}{1 - y}\right)$$

Then *z* is defined as the log of the ratio of the probabilities of the two possible outcomes: *y* and *1 – y*.

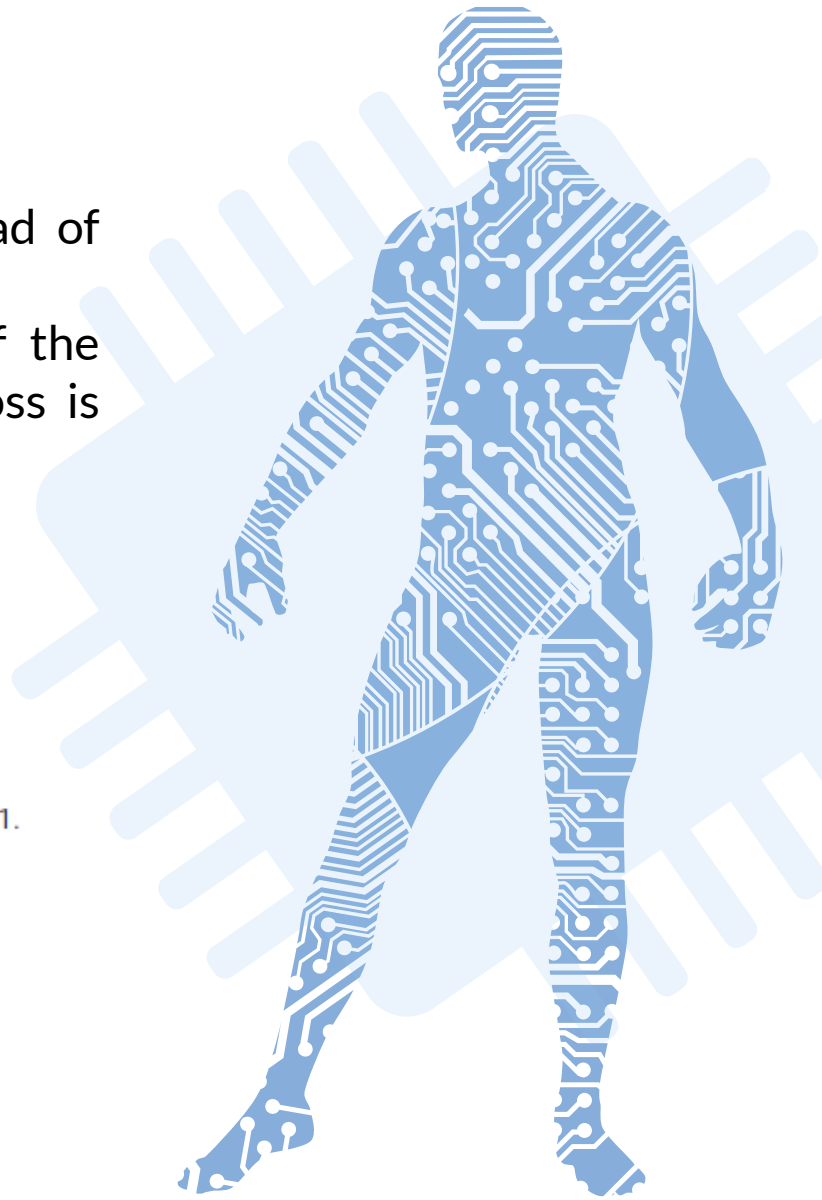
Logistic Regression loss

❑ Logistic regression models use Log Loss as the loss function instead of squared loss.
❑ The Log Loss equation returns the logarithm of the magnitude of the change, rather than just the distance from data to prediction. Log Loss is calculated as follows:

$$\text{Log Loss} = \sum_{(x,y)\in D} -y \log(y') - (1-y) \log(1-y')$$

where:

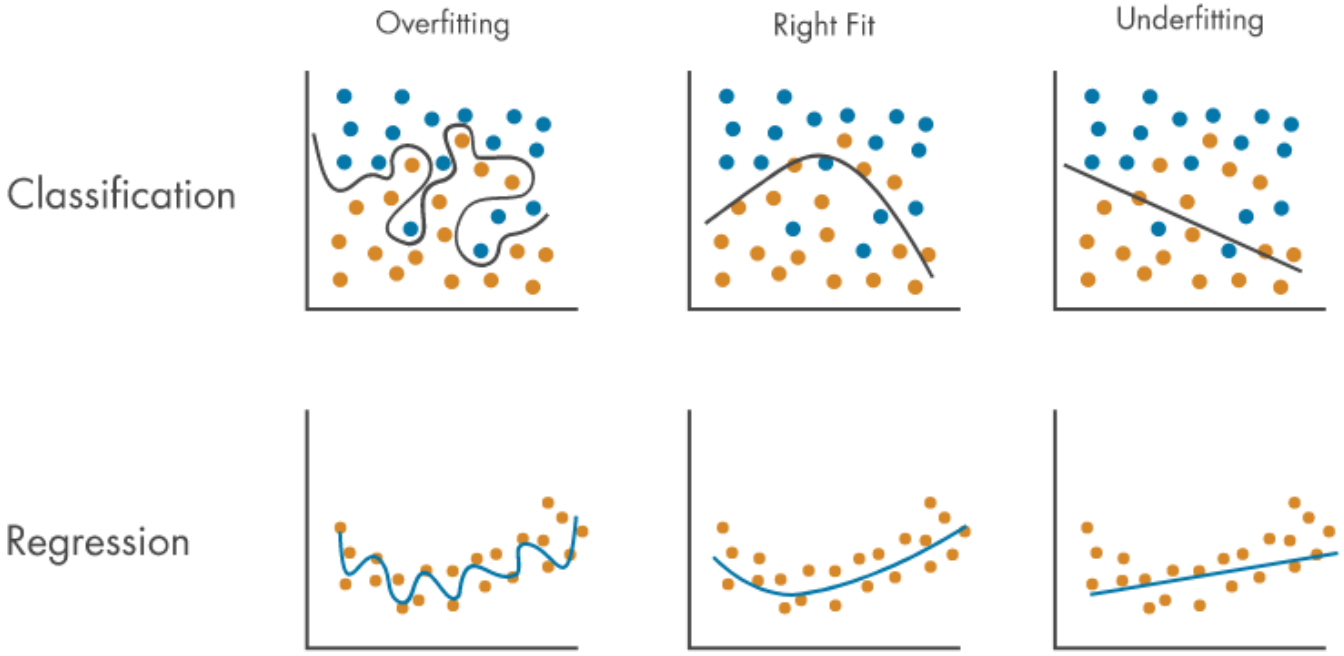
- $(x, y) \in D$ is the dataset containing many labeled examples, which are (x, y) pairs.
- y is the label in a labeled example. Since this is logistic regression, every value of y must either be 0 or 1.
- y' is your model's prediction (somewhere between 0 and 1), given the set of features in x.

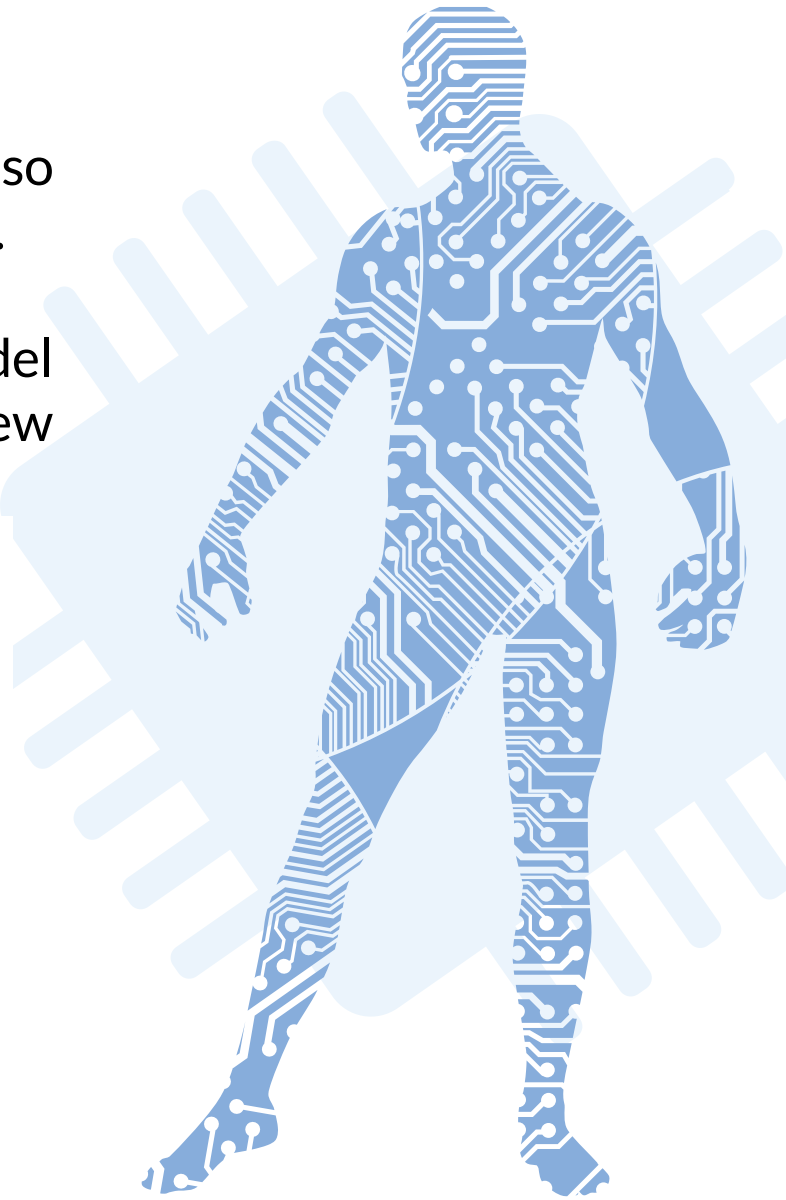


Overfitting

❑ Overfitting: Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.

❑ Underfitting is the opposite concept of overfitting; the model doesn't align well with the training data or generalize well to new data.





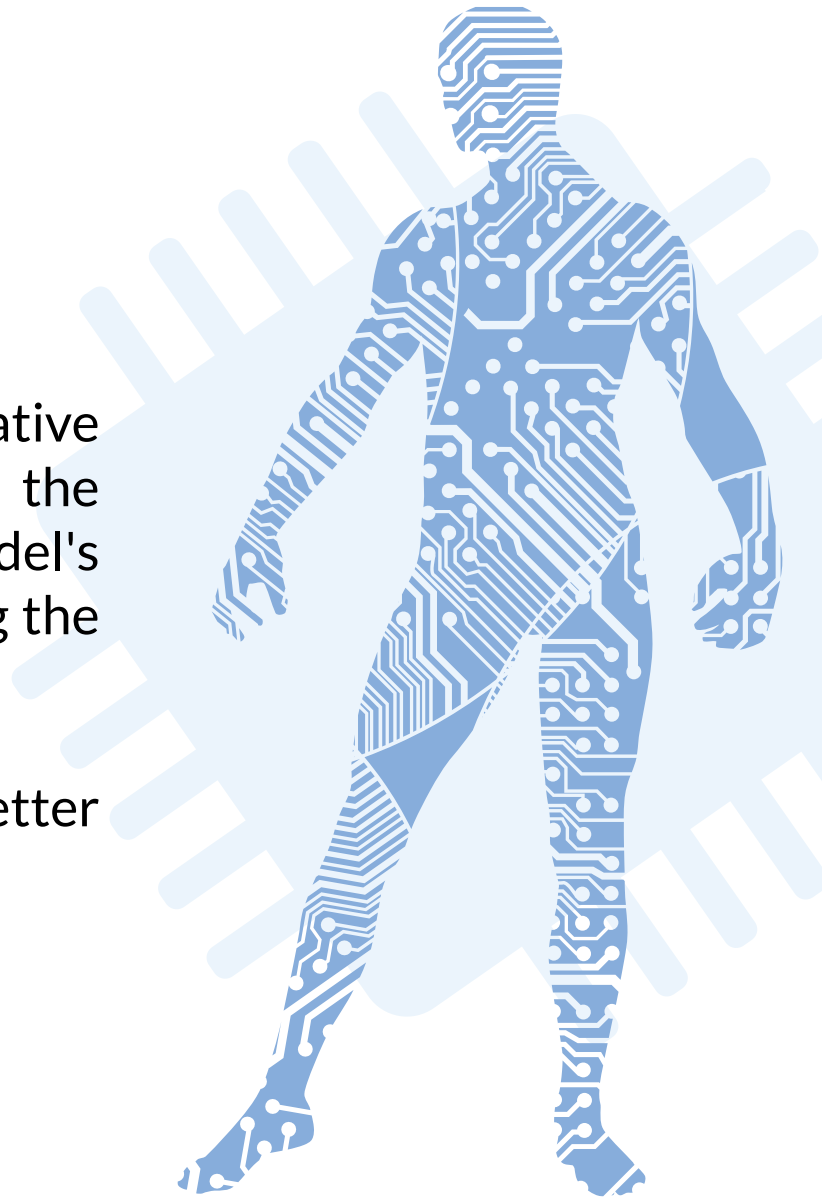
Overfitting

❑ Overfitting: Creating a model that matches the training data so closely that the model fails to make correct predictions on new data.

❑ Underfitting is the opposite concept of overfitting; the model doesn't align well with the training data or generalize well to new data.

Error	Overfitting	Right Fit	Underfitting
Training	Low	Low	High
Test	High	Low	High

regularization



- Regularization: any mechanism that reduces overfitting.

- Regularization rate λ: A number that specifies the relative importance of regularization during training. Raising the regularization rate reduces overfitting but may reduce the model's predictive power (increase loss). Conversely, reducing or omitting the regularization rate increases overfitting.

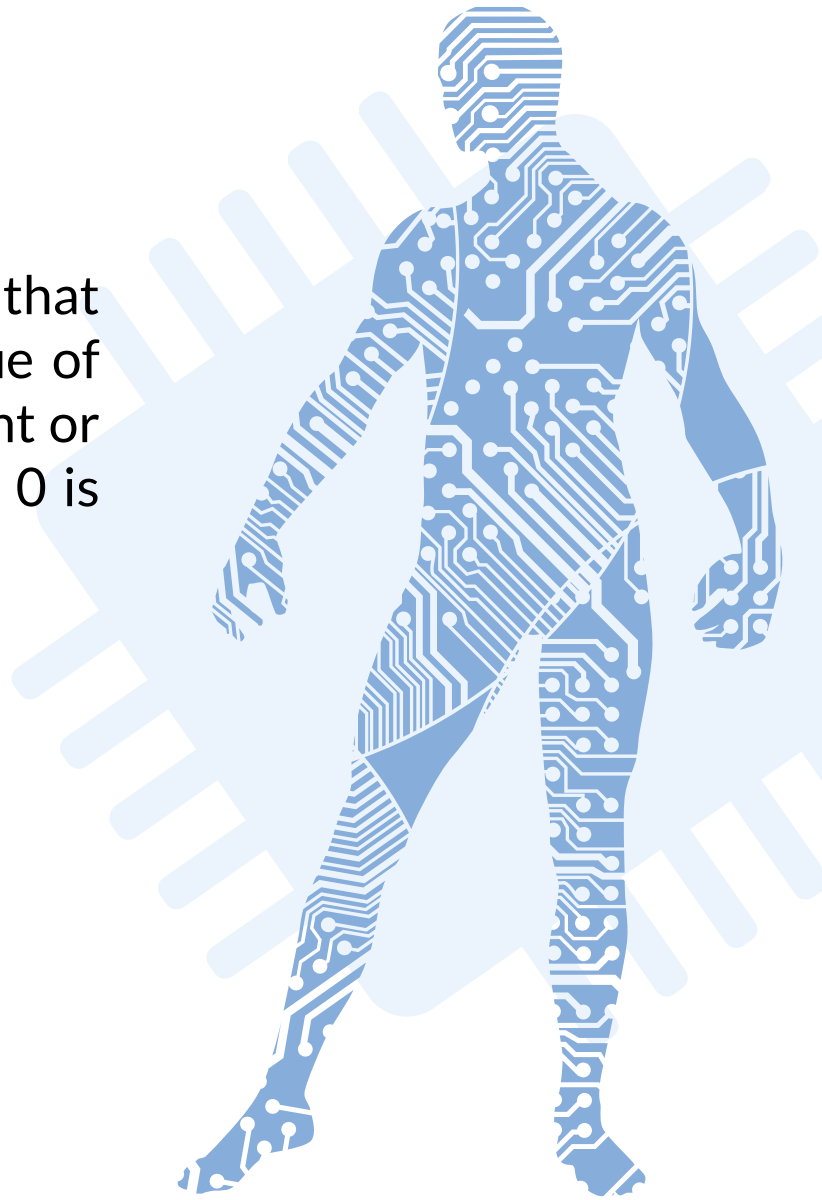
- The regularization rate is usually represented as the Greek letter lambda.

$$minimiz(loss\ function + \lambda(regularization))$$

regularization

❑ L1 regularization (L1 norm or Lasso) : A type of regularization that penalizes weights in proportion to the sum of the absolute value of the weights. L1 regularization helps drive the weights of irrelevant or barely relevant features to exactly 0. A feature with a weight of 0 is effectively removed from the model.

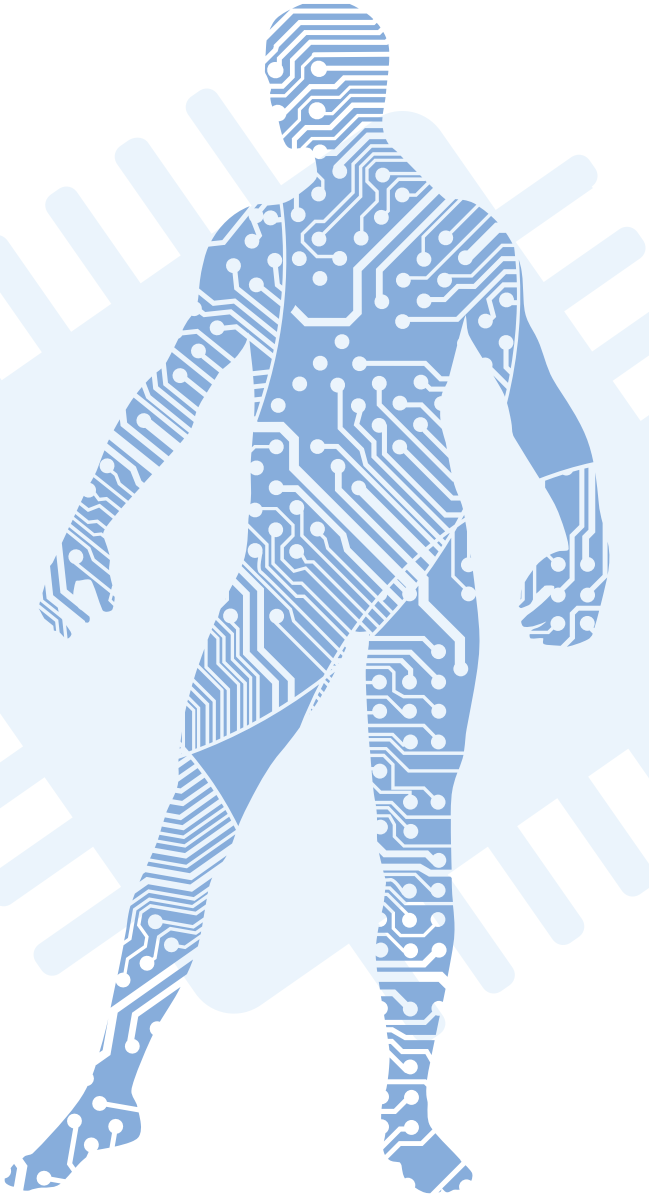
$$loss = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2 + \lambda \sum_{i=1}^{N}|w_i|$$



regularization

❑ L2 regularization the L2 norm, or Ridge : A type of regularization that penalizes weights in proportion to the sum of the squares of the weights. L2 regularization helps drive outlier weights (those with high positive or low negative values) closer to 0 but not quite to 0. Features with values very close to 0 remain in the model but don't influence the model's prediction very much. L2 regularization always improves generalization in linear models.

$$loss = \frac{1}{N}\sum_{i=1}^{N}(\hat{y} - y)^2 + \lambda \sum_{i=1}^{N} w_i^2$$



regularization

❑ L2 regularization is a popular regularization metric, which uses the following formula:

$$L_2\,regularization = w_1^2 + w_2^2 + w_3^2 + \cdots + w_n^2$$

For example, the following table shows the calculation of L_2 regularization for a model with six weights:

	Value	Squared value
w_1	0.2	0.04
w_2	-0.5	0.25
w_3	5.0	25.0
w_4	-1.2	1.44
w_5	0.3	0.09
w_6	-0.1	0.01
		26.83 = total

regularization

❑ Notice that weights close to zero don't affect L2 regularization much, but large weights can have a huge impact. For example, in the preceding calculation:

- ❑A single weight (w3) contributes about 93% of the total complexity.
- ❑The other five weights collectively contribute only about 7% of the total complexity.
- ❑ L2 regularization encourages weights toward 0, but never pushes weights all the way to zero.

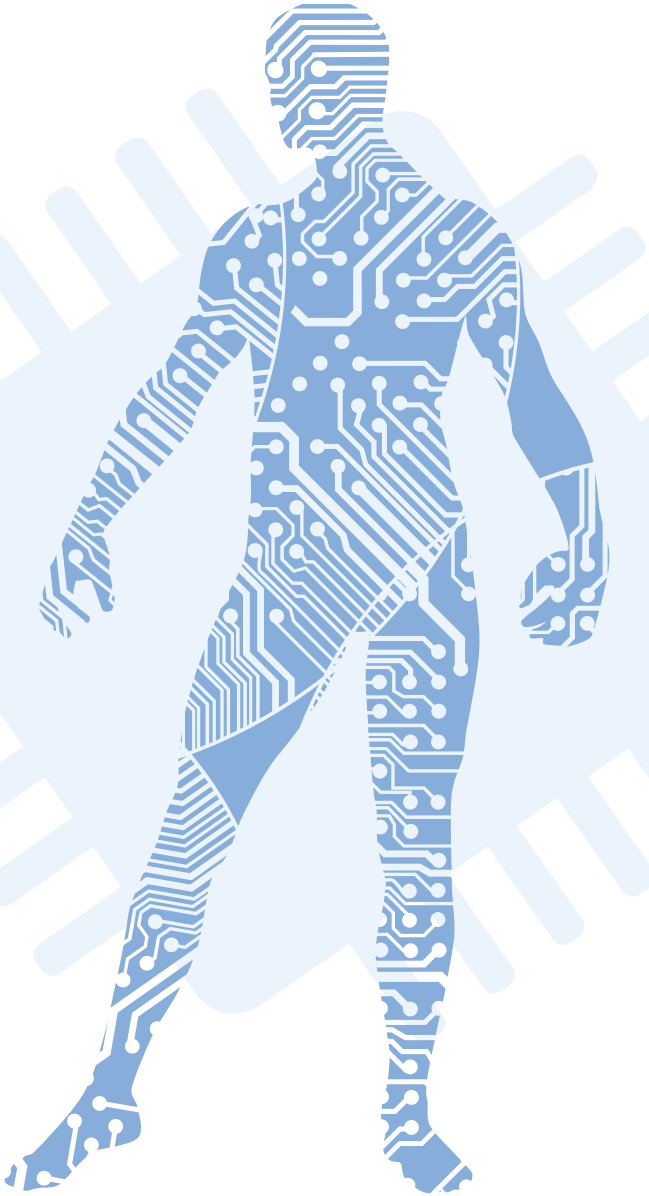
Regularization rate (lambda)

❑ As noted, training attempts to minimize some combination of loss and complexity:

$$minimiz(loss + complexity)$$

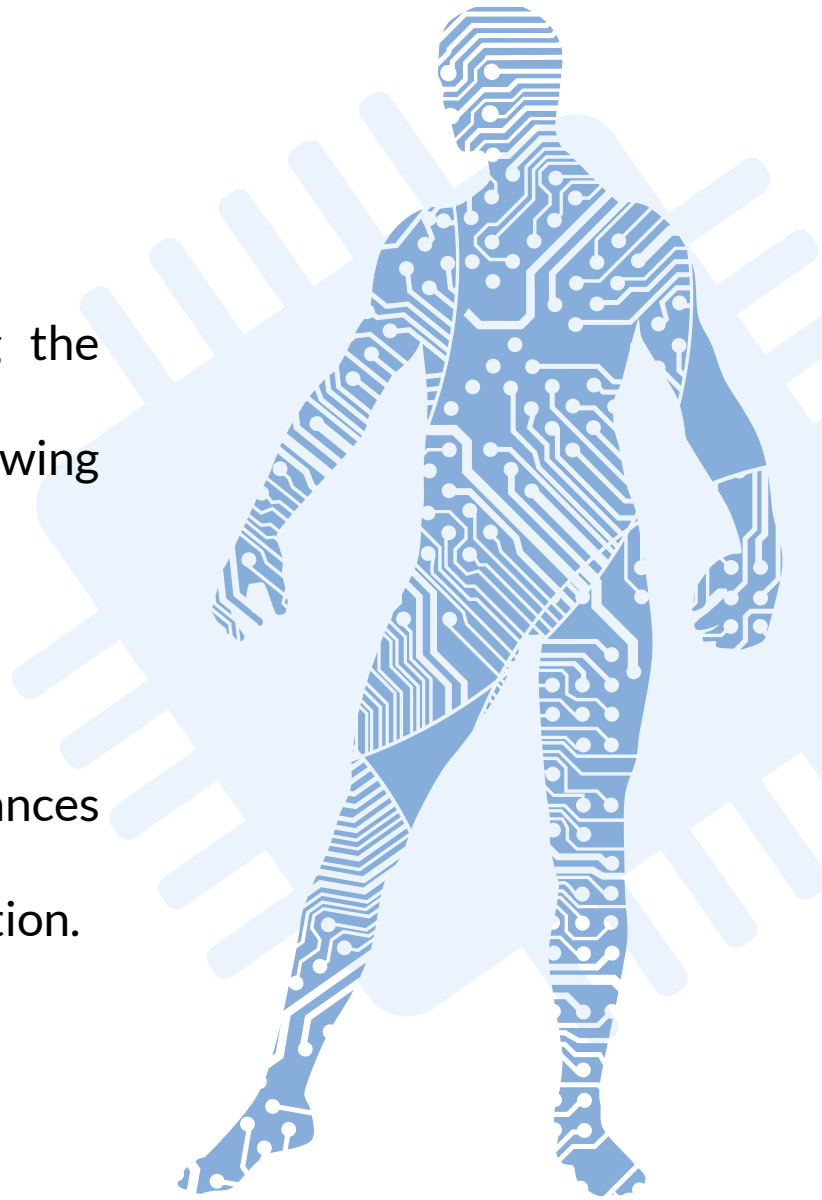
❑Model developers tune the overall impact of complexity on model training by multiplying its value by a scalar called the regularization rate. The Greek character lambda typically symbolizes the regularization rate. That is, model developers aim to do the following:

$$minimiz(loss\ function + \lambda(regularization))$$

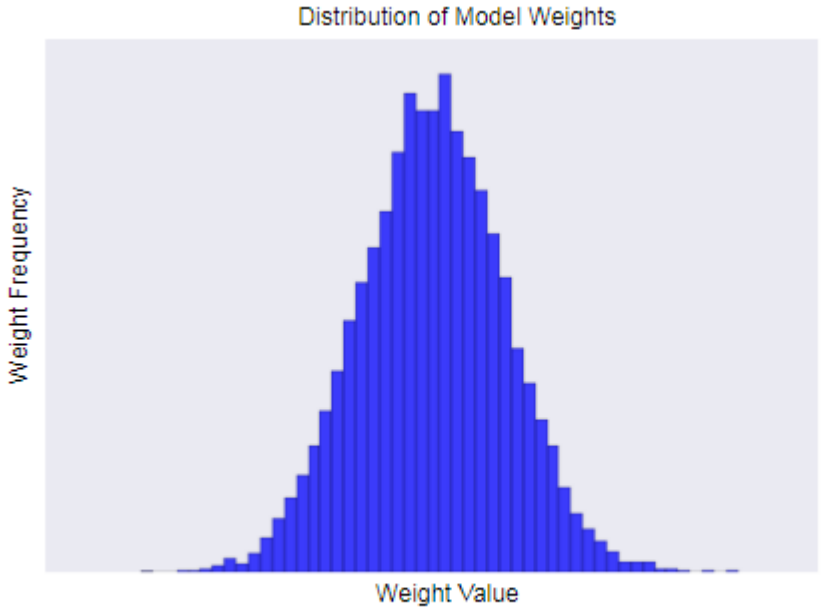


Regularization rate (lambda)

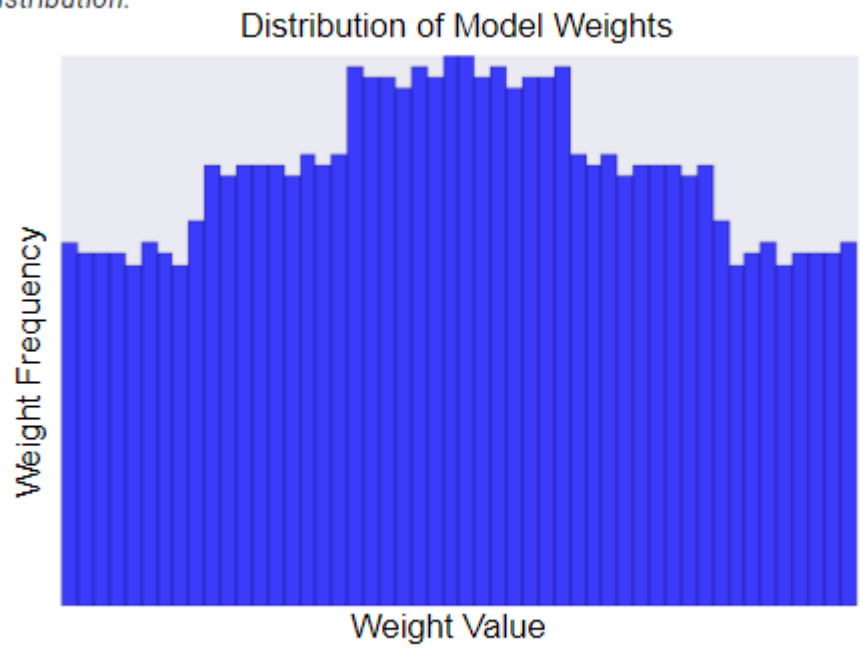
- A high regularization rate:
  - Strengthens the influence of regularization, thereby reducing the chances of overfitting.
  - Tends to produce a histogram of model weights having the following characteristics:
    - a normal distribution
    - a mean weight of 0.
- A low regularization rate:
  - Lowers the influence of regularization, thereby increasing the chances of overfitting.
  - Tends to produce a histogram of model weights with a flat distribution.



Notes



Weight histogram for a high regularization rate. Mean is zero. Normal distribution.



Weight histogram for a low regularization rate. Mean may or may not be zero.

Notes

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
```

```
#Model
lr = LinearRegression()

#Fit model
lr.fit(X_train, y_train)

#predict
#prediction = lr.predict(X_test)

#actual
actual = y_test

train_score_lr = lr.score(X_train, y_train)
test_score_lr = lr.score(X_test, y_test)

print("The train score for lr model is {}".format(train_score_lr))
print("The test score for lr model is {}".format(test_score_lr))
```

Notes

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
```

```
#Lasso regression model
print("\nLasso Model.....................................\n")
lasso = Lasso(alpha = 10)
lasso.fit(X_train,y_train)
train_score_ls =lasso.score(X_train,y_train)
test_score_ls =lasso.score(X_test,y_test)

print("The train score for ls model is {}".format(train_score_ls))
print("The test score for ls model is {}".format(test_score_ls))
```

Notes

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge, RidgeCV, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_boston
```

```
#Ridge Regression Model
ridgeReg = Ridge(alpha=10)

ridgeReg.fit(X_train,y_train)

#train and test scorefor ridge regression
train_score_ridge = ridgeReg.score(X_train, y_train)
test_score_ridge = ridgeReg.score(X_test, y_test)

print("\nRidge Model.....................................\n")
print("The train score for ridge model is {}".format(train_score_ridge))
print("The test score for ridge model is {}".format(test_score_ridge))
```

K-Nearest Neighbors (KNN)

❑The K-Nearest Neighbors algorithm is a supervised learning classifier, which uses proximity to make predictions/classifications about the group of an individual data points.

❑ It is based on the assumption that similar points can be found near one another.

❑ For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used.

Distance Metrics

❑The Euclidean Distance

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

❑ Manhattan distance

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i| \right)$$

Distance Metrics

- **Minkowski Distance:**

  This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

$$D(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

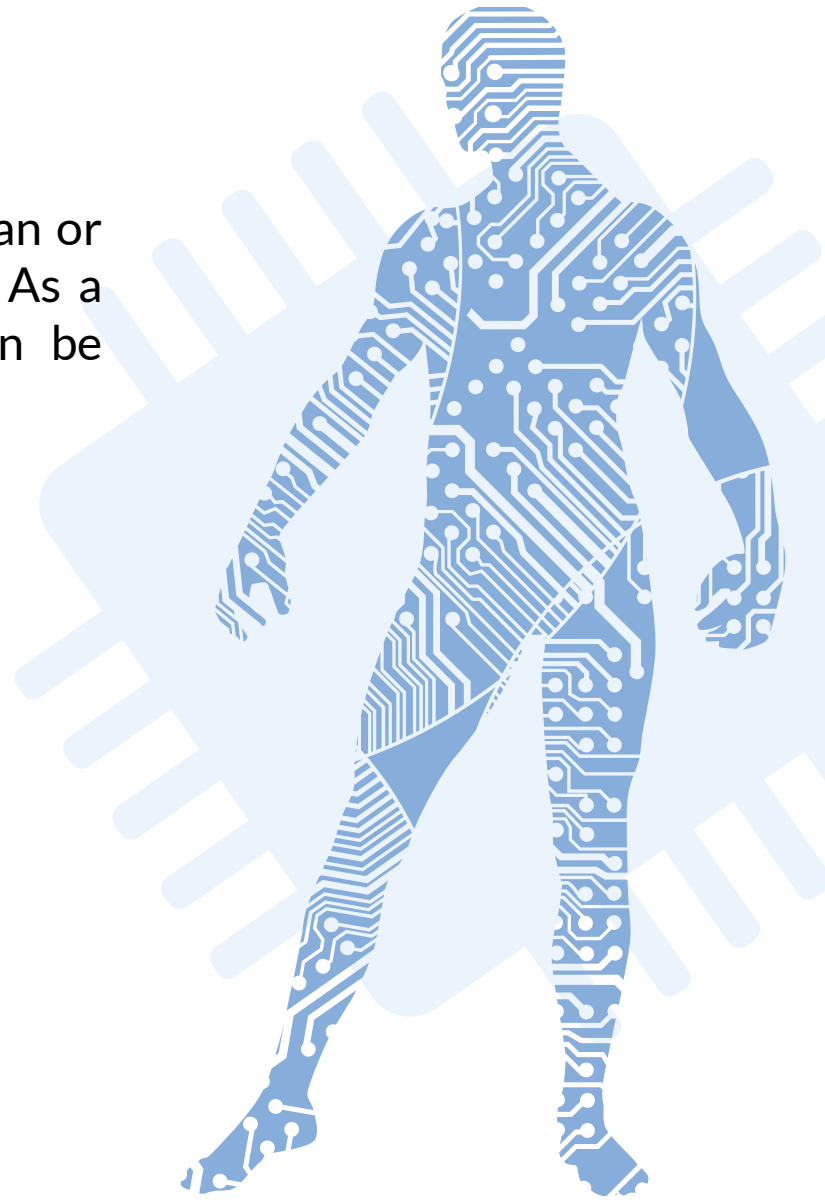
Distance Metrics

❑ **Hamming Distance:** This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result, it has also been referred to as the overlap metric. This can be represented with the following formula:

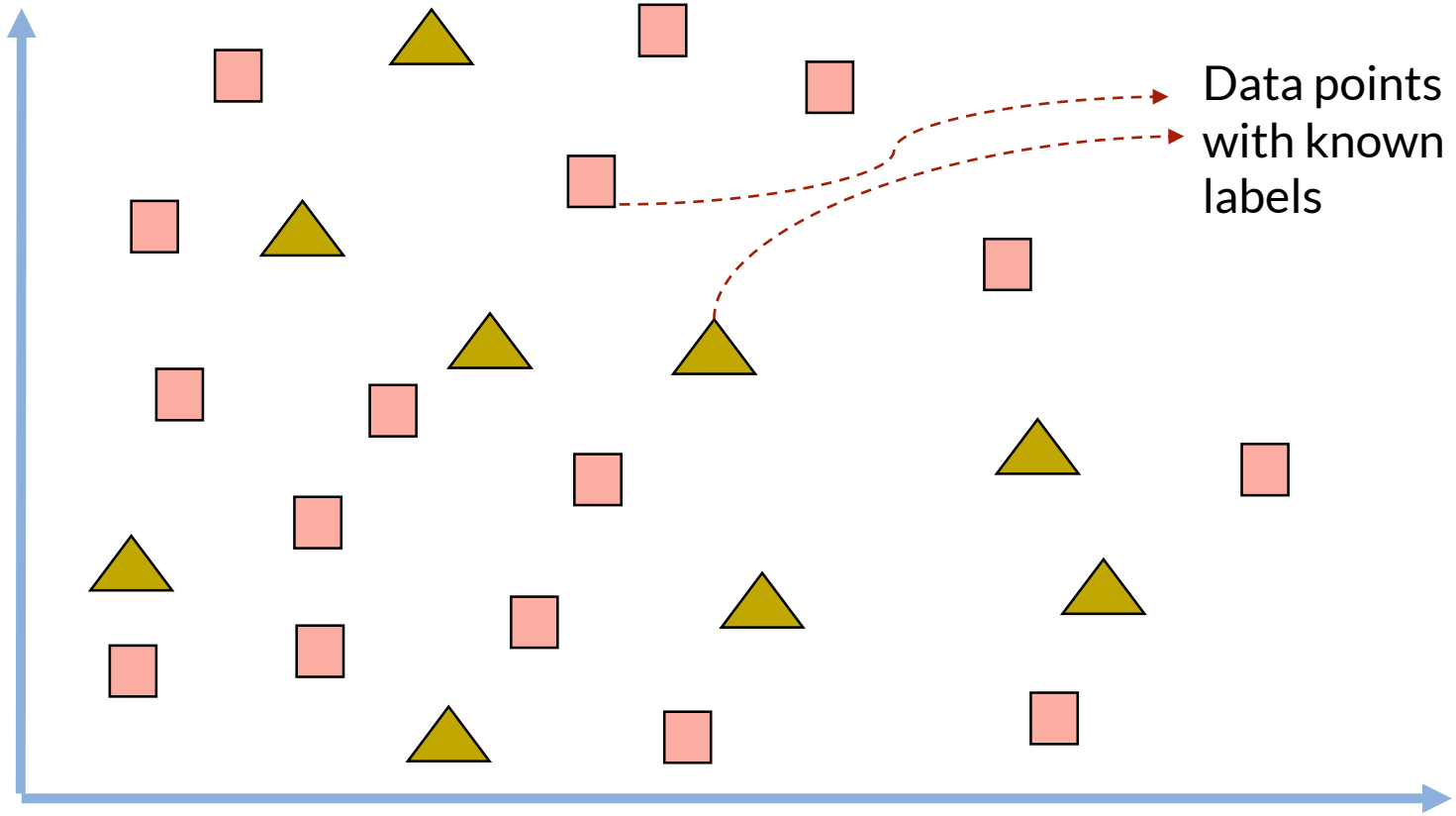
$$\text{Hamming Distance} = D_H = \left(\sum_{i=1}^{k} |x_i - y_i| \right)$$

$x=y \quad D=0$

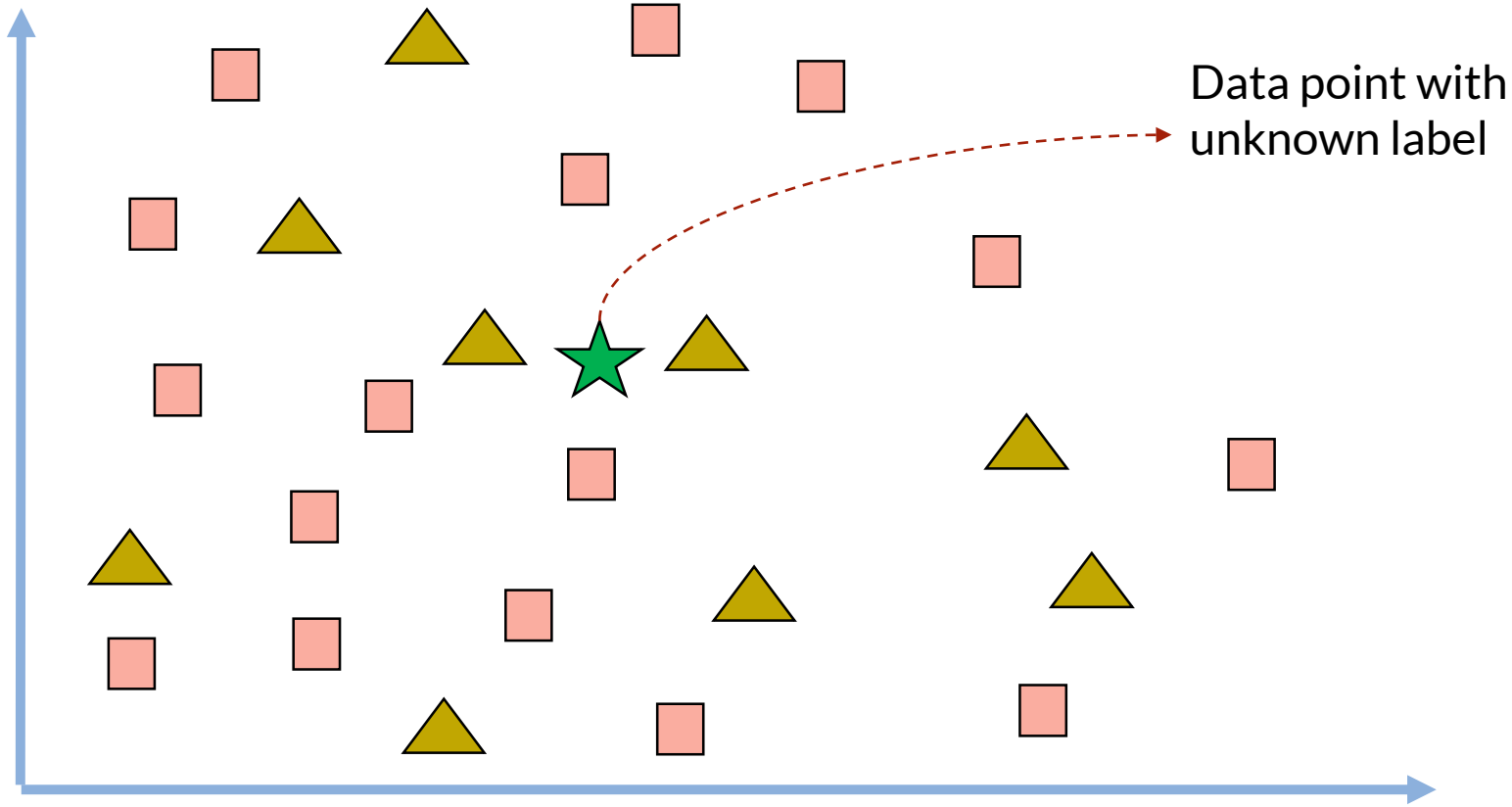
$x \neq y \quad D \neq 1$



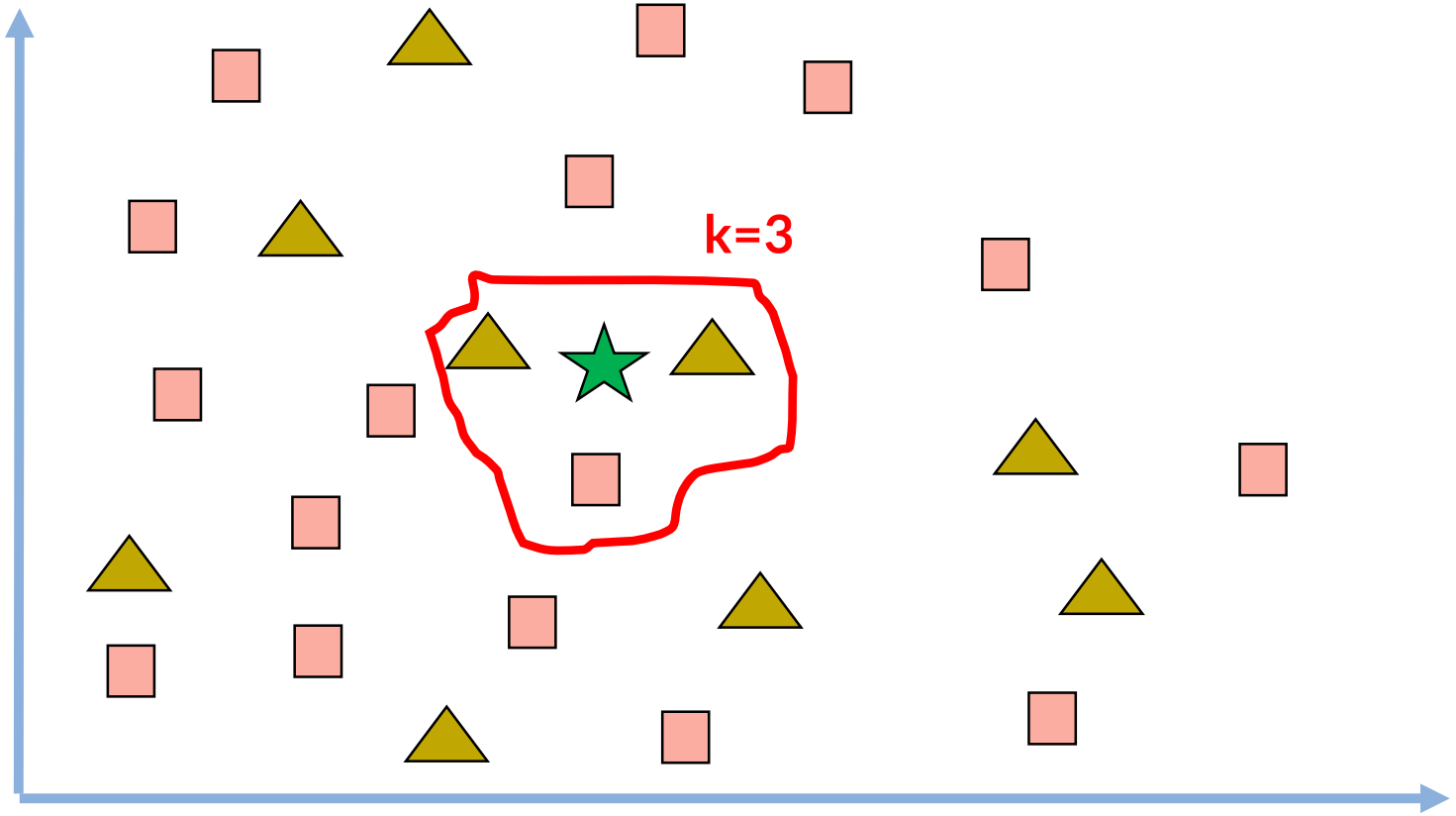
K-Nearest Neighbors (KNN)



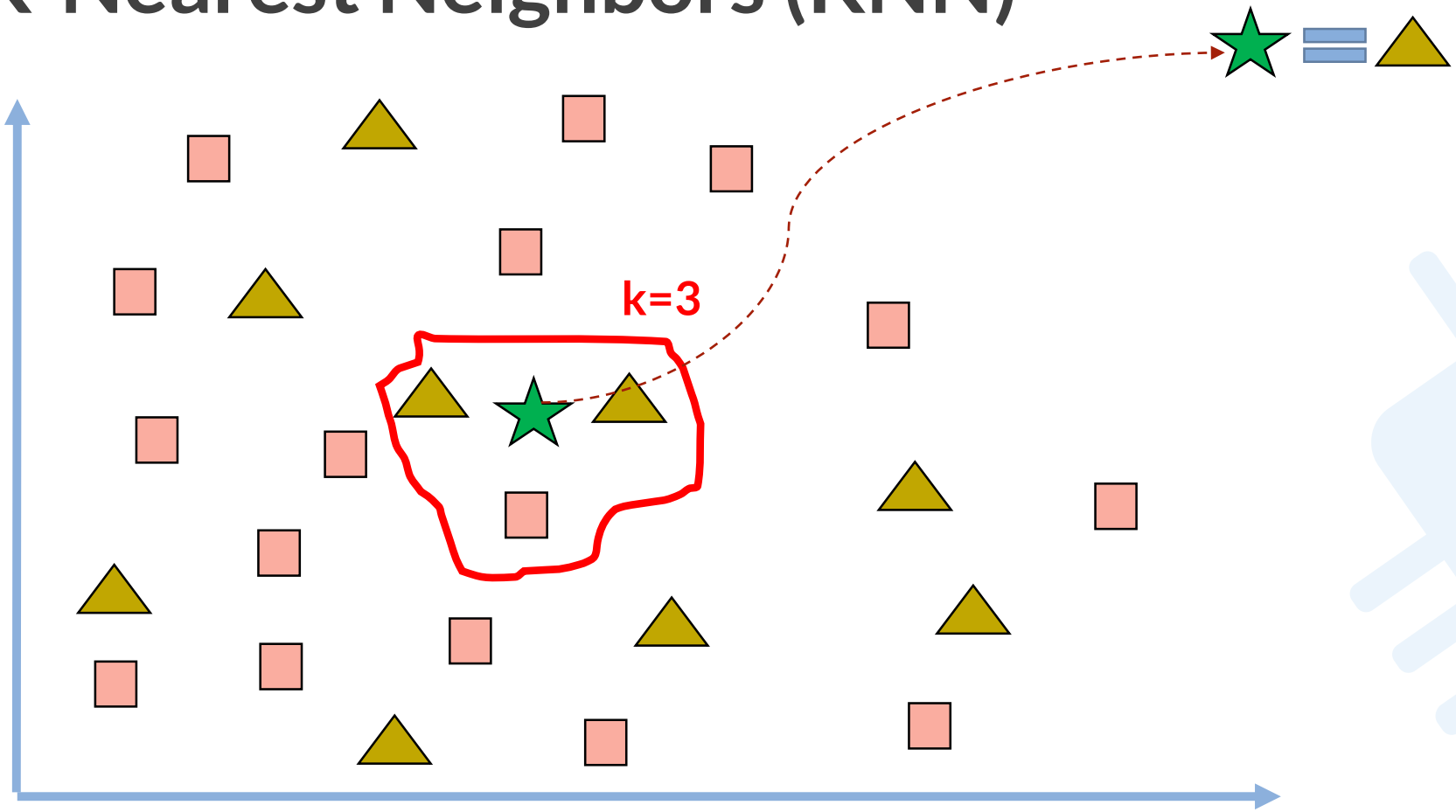
K-Nearest Neighbors (KNN)



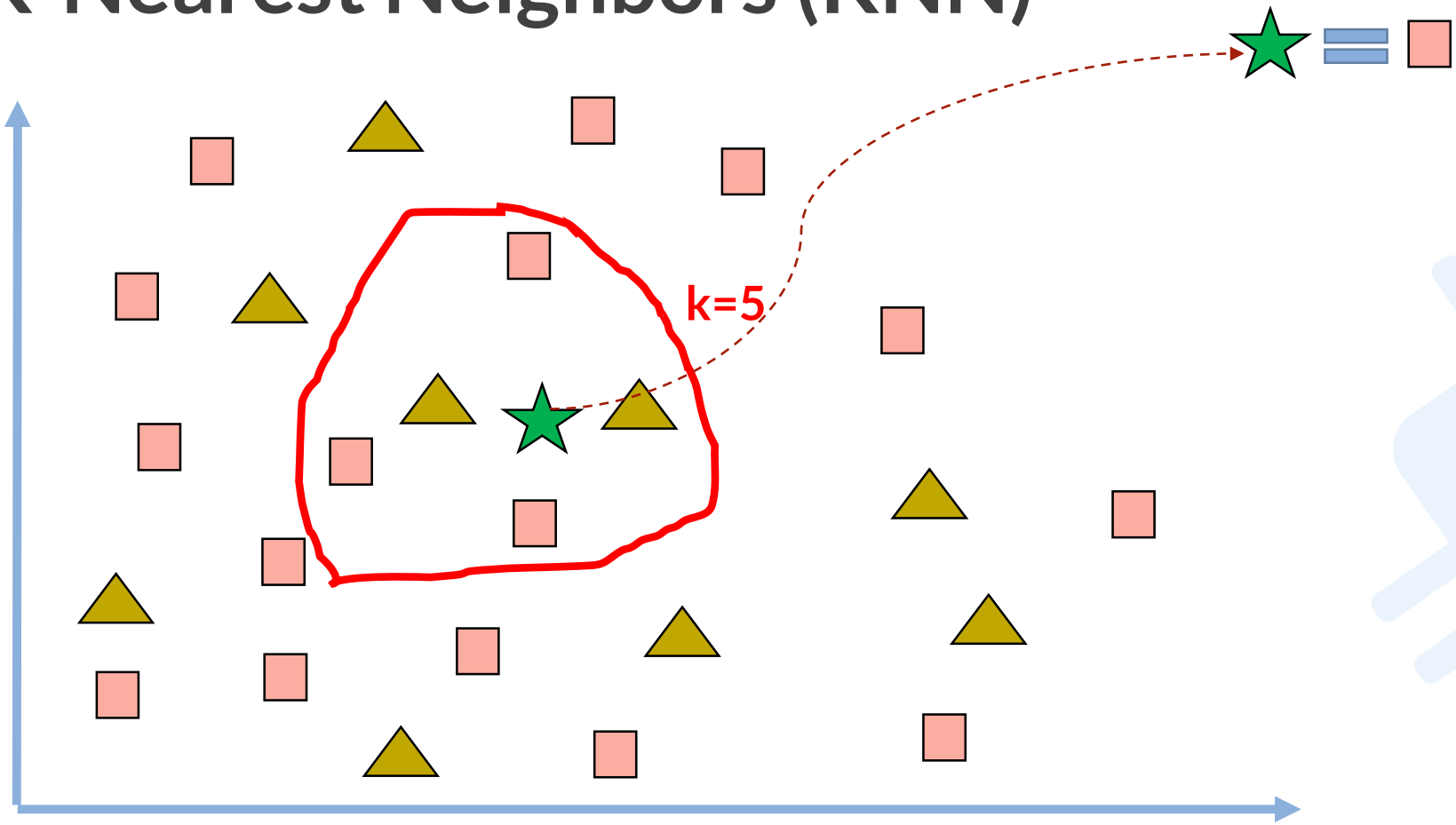
K-Nearest Neighbors (KNN)



K-Nearest Neighbors (KNN)



K-Nearest Neighbors (KNN)

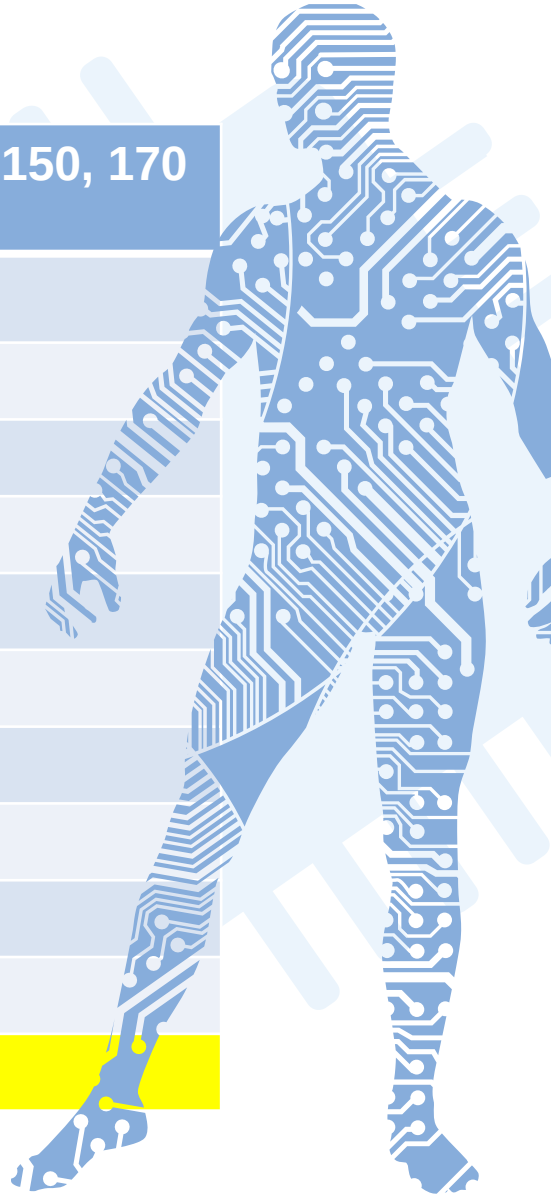


K-Nearest Neighbors (KNN)

Weight (x_1)	Height (x_2)	Size (y)
121	160	S
127.6	165	S
132	160	M
143	166	M
154	168	M
149.6	171	M
165	175	L
165	180	L
176	187	L
198	190	L
150	**170**	**?**

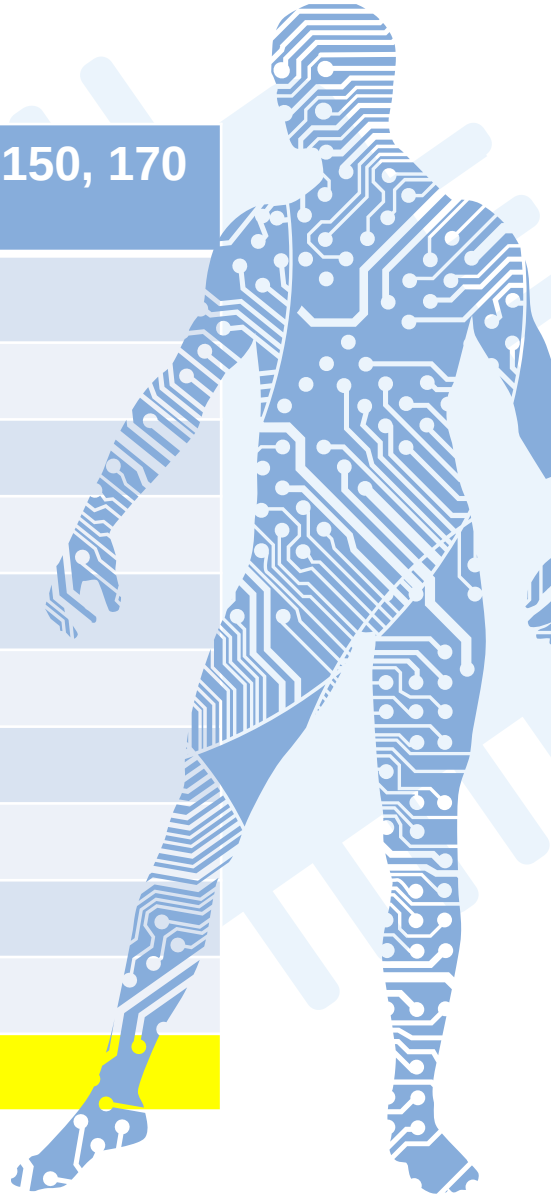
K-Nearest Neighbors (KNN)

Weight (x_1)	Height (x_2)	Size (y)	Compute Distance according to the point 150, 170
121	160	S	$\sqrt{(150-121)^2+(170-160)^2}$=30.675723
127.6	165	S	
132	160	M	
143	166	M	
154	168	M	
149.6	171	M	
165	175	L	
165	180	L	
176	187	L	
198	190	L	
150	**170**	**?**	



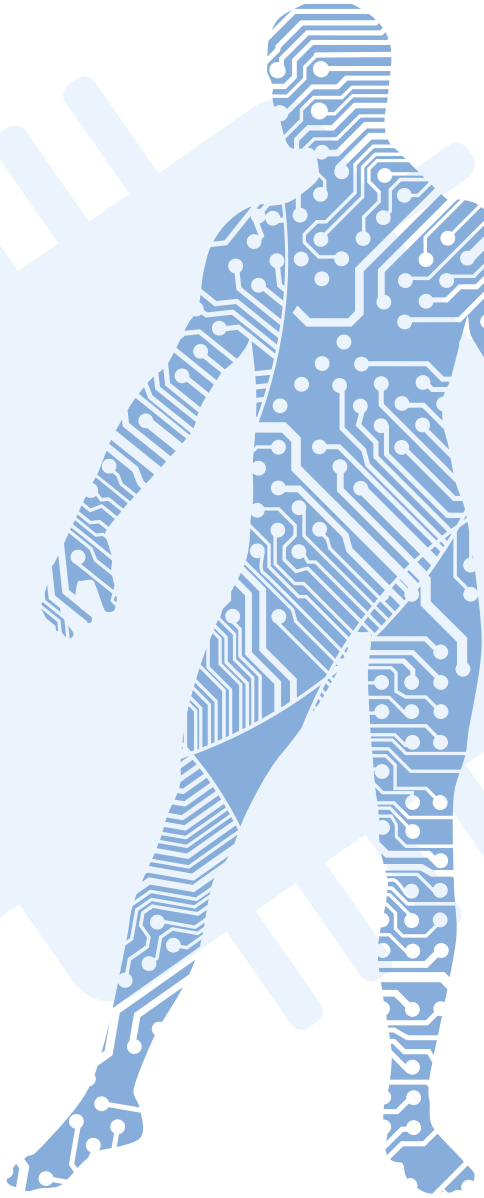
K-Nearest Neighbors (KNN)

Weight (x_1)	Height (x_2)	Size (y)	Compute Distance according to the point 150, 170
121	160	S	$\sqrt{(150-121)^2+(170-160)^2}$=30.675723
127.6	165	S	22.95125
132	160	M	20.59126
143	166	M	8.062258
154	168	M	4.472136
149.6	171	M	1.077033
165	175	L	15.81139
165	180	L	18.02776
176	187	L	31.06445
198	190	L	52
150	**170**	**?**	



K-Nearest Neighbors (KNN)

Weigh t (x_1)	Height (x_2)	Size (y)	Distance	Rank
121	160	S	30.675723	8
127.6	165	S	22.95125	7
132	160	M	20.59126	6
143	166	M	8.062258	3
154	168	M	4.472136	2
149.6	171	M	1.077033	1
165	175	L	15.81139	4
165	180	L	18.02776	5
176	187	L	31.06445	9
198	190	L	52	10
150	**170**	**?**		



K-Nearest Neighbors (KNN)

Weight (x_1)	Height (x_2)	Size (y)	Distance	Rank
121	160	S	30.675723	8
127.6	165	S	22.95125	7
132	160	M	20.59126	6
143	166	M	8.062258	3
154	168	M	4.472136	2
149.6	171	M	1.077033	1
165	175	L	15.81139	4
165	180	L	18.02776	5
176	187	L	31.06445	9
198	190	L	52	10
150	**170**	**?**		

k=3

K-Nearest Neighbors (KNN)

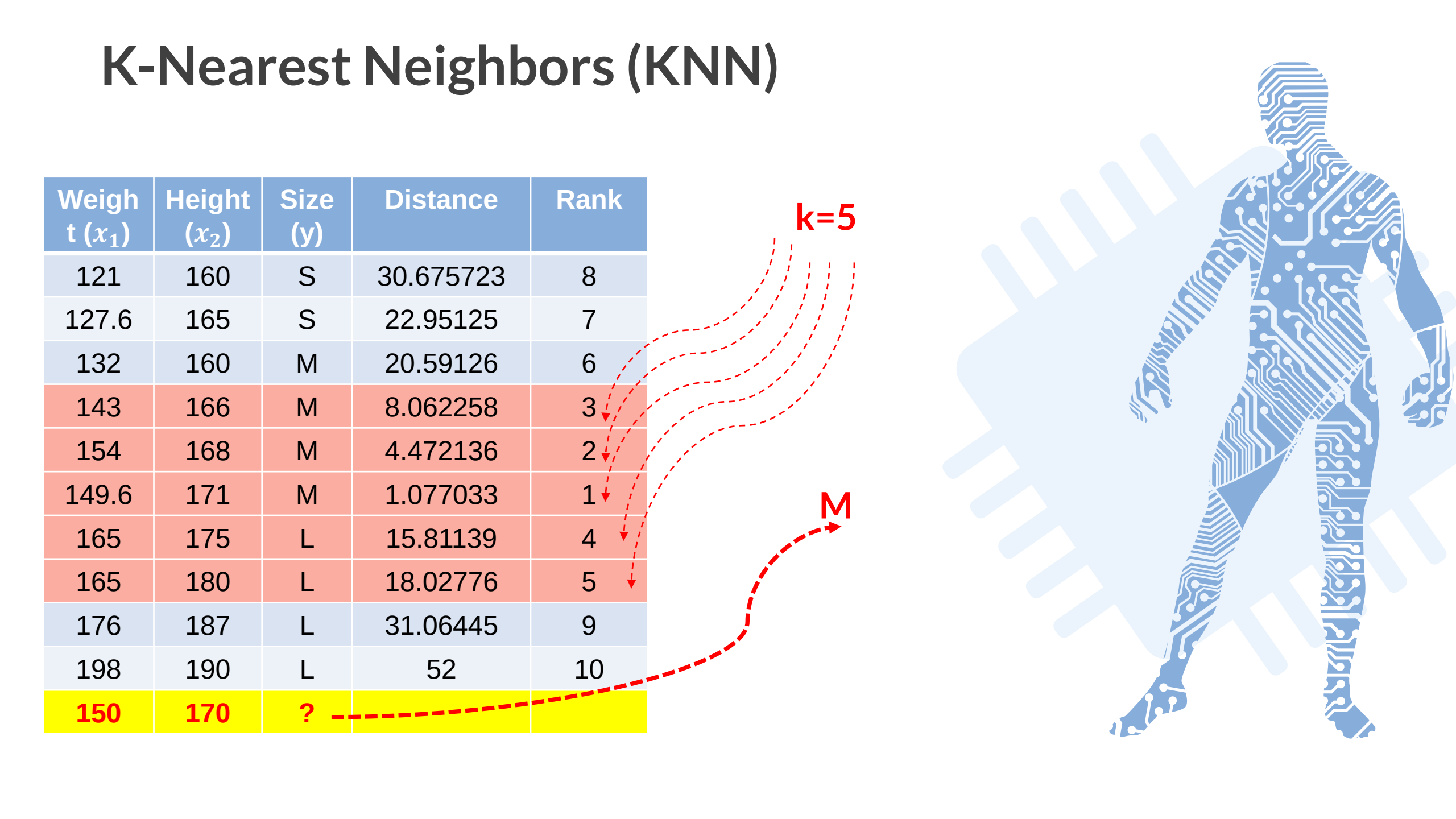
Weigh t (x_1)	Height (x_2)	Size (y)	Rank
121	160	S	8
127.6	165	S	7
132	160	M	6
143	166	M	3
154	168	M	2
149.6	171	M	1
165	175	L	4
165	180	L	5
176	187	L	9
198	190	L	10
150	**170**	**?**	

k=3

M

K-Nearest Neighbors (KNN)

Weigh t (x_1)	Height (x_2)	Size (y)	Distance	Rank
121	160	S	30.675723	8
127.6	165	S	22.95125	7
132	160	M	20.59126	6
143	166	M	8.062258	3
154	168	M	4.472136	2
149.6	171	M	1.077033	1
165	175	L	15.81139	4
165	180	L	18.02776	5
176	187	L	31.06445	9
198	190	L	52	10
150	**170**	**?**		

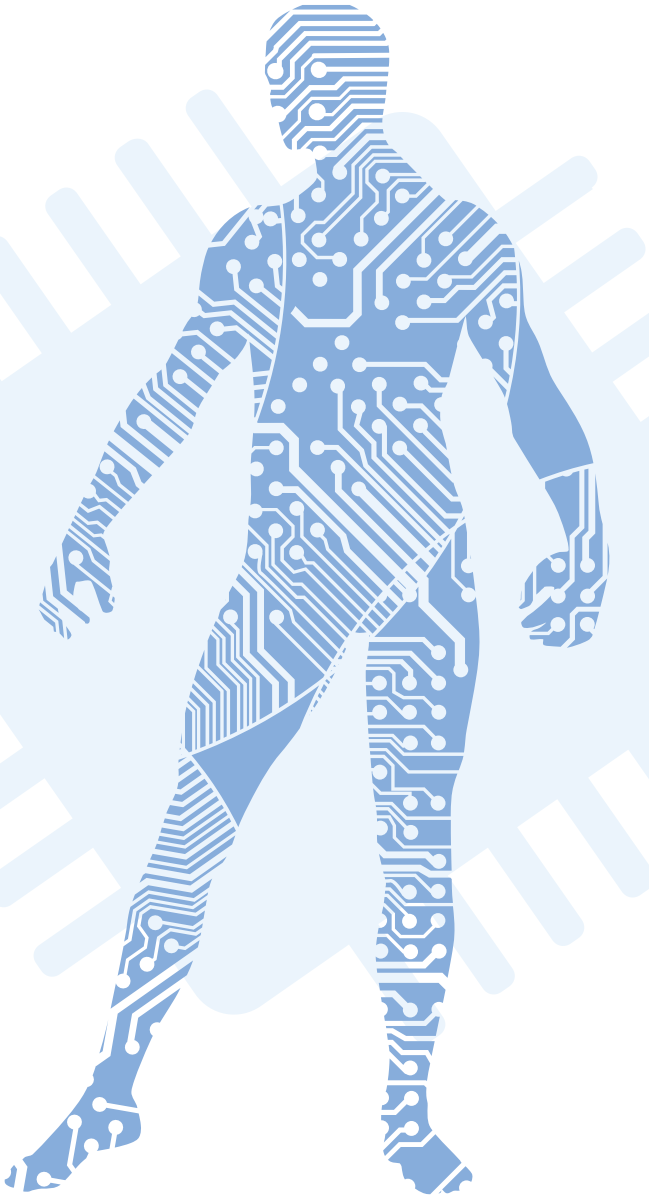


KNN (defining k)

❑ The k value in the k-NN algorithm defines how many neighbors will be checked to determine the classification of a specific query point.

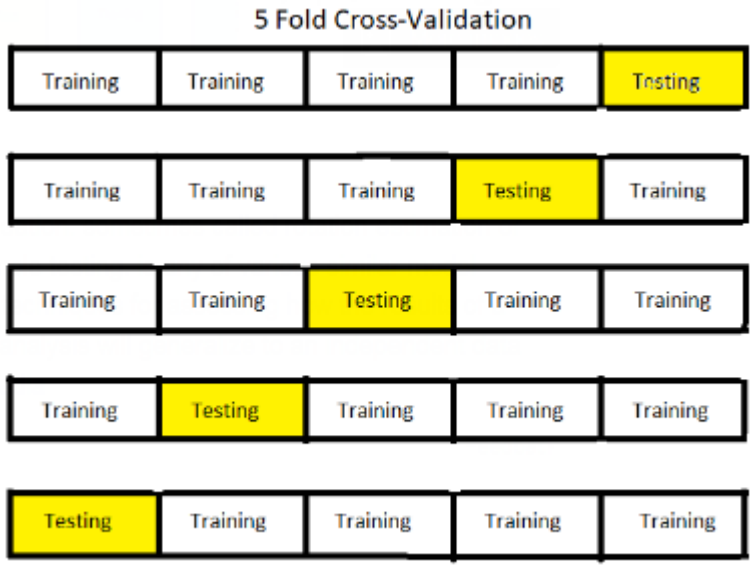
❑ The choice of k will largely depend on the input data as data with more outliers or noise will likely perform better with higher values of k.

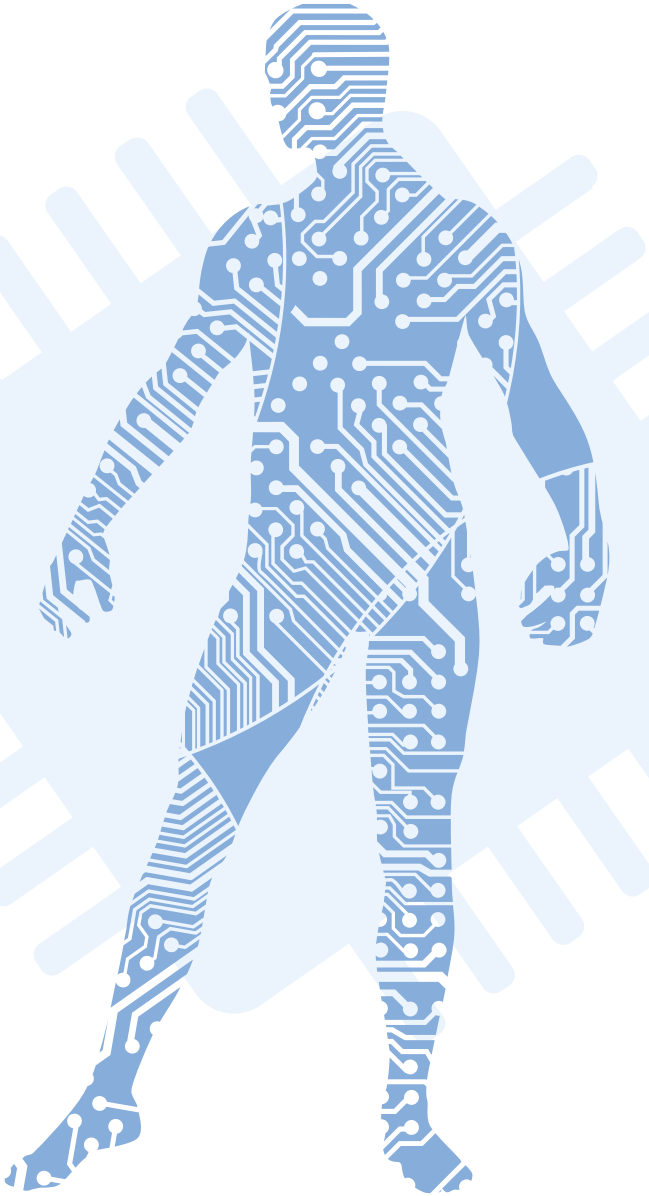
❑ Overall, it is recommended to have an odd number for k to avoid ties in classification, and cross-validation tactics can help you choose the optimal k for your dataset.



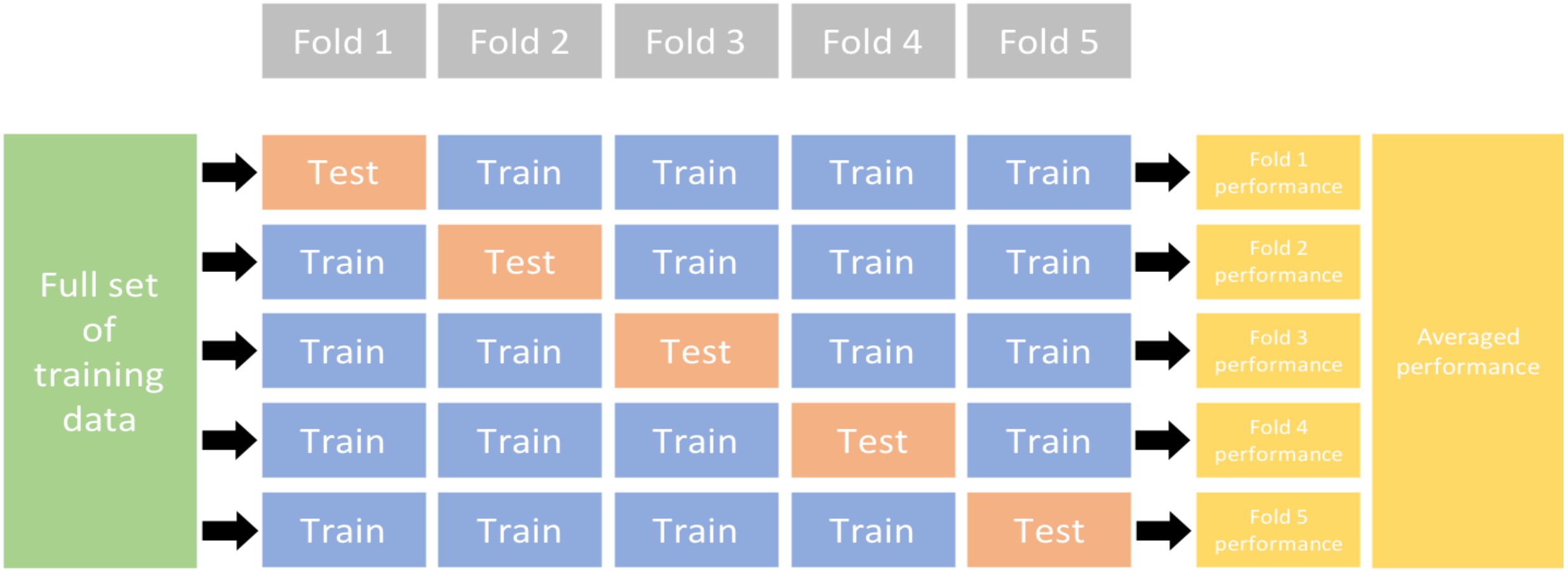
Cross-Validation

❑ **k-fold:** Partitions data into **k** randomly chosen subsets (or folds) of roughly equal size. One subset is used to validate the model trained using the remaining subsets. This process is repeated k times such that each subset is used exactly once for validation. The average error across all k partitions is reported as ε. This is one of the most popular techniques for cross-validation but can take a long time to execute because the model needs to be trained repeatedly. The image below illustrates the process.





Cross-Validation



Applications of KNN in Machine Learning

❑ **Data preprocessing:** Datasets frequently have missing values, but the KNN algorithm can estimate for those values in a process known as missing data imputation.

❑ **Recommendation Engines**: Using clickstream data from websites, the KNN algorithm has been used to provide automatic recommendations to users on additional content. a user is assigned to a particular group, and based on that group's user behavior, they are given a recommendation.

❑ **Healthcare:** KNN has also had application within the healthcare industry, making predictions on the risk of heart attacks and prostate cancer. The algorithm works by calculating the most likely gene expressions.

❑ **Pattern Recognition:** KNN has also assisted in identifying patterns, such as in text and digit classification. This has been particularly helpful in identifying handwritten numbers that you might find on forms or mailing envelopes.

Notes

Python

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
import numpy as np

# Load a sample dataset
iris = load_iris()
X, y = iris.data, iris.target

# Create a model
model = LogisticRegression(max_iter=200, random_state=42)

# Define the cross-validation strategy (e.g., K-Fold with 5 splits)
num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)

# Perform cross-validation
scores = cross_val_score(model, X, y, cv=kf, scoring='accuracy')

# Print the results
print("Cross-Validation Scores (Accuracy per fold):", scores)
print("Mean Accuracy:", np.mean(scores))
print("Standard Deviation of Accuracy:", np.std(scores))
```

Notes

```
#Using the linear CV model
from sklearn.linear_model import LassoCV

#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0001, 0.001,0.01, 0.1, 1, 10], random_state=0).fit(X_train, y_train)


#score
print(lasso_cv.score(X_train, y_train))
print(lasso_cv.score(X_test, y_test))
```


Thank You