Aprendizado de Máquina e Reconhecimento de Padrões

K-Nearest Neighbors (KNN)

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas

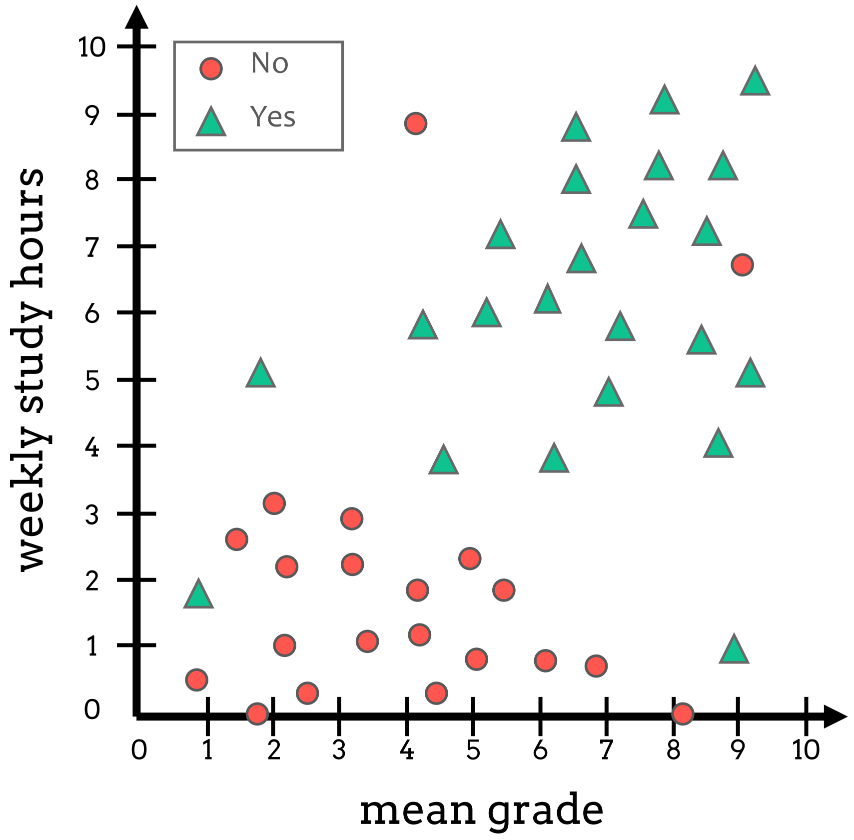
KNN

One of the simplest (**instance-based**) classification algorithms.

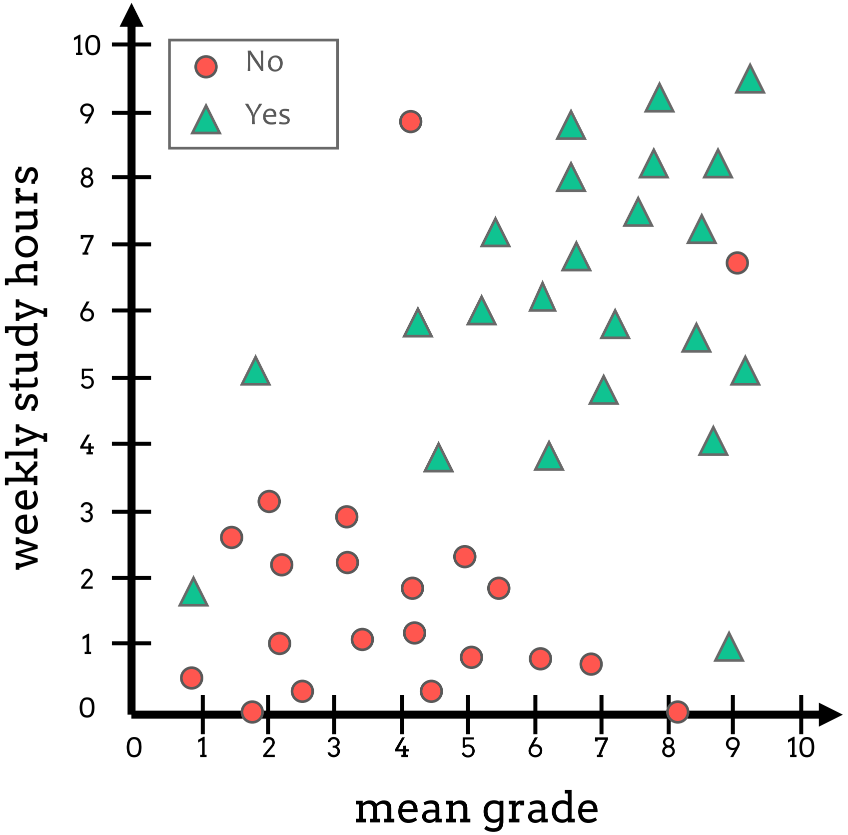
It works for binary or multiclass problems.

Training Set

mean grade	wekly study hours	approved at a university?
0.9	0.5	No
2.2	2	No
9.00	7.2	Yes
6.5	8.00	Yes
...	...	...

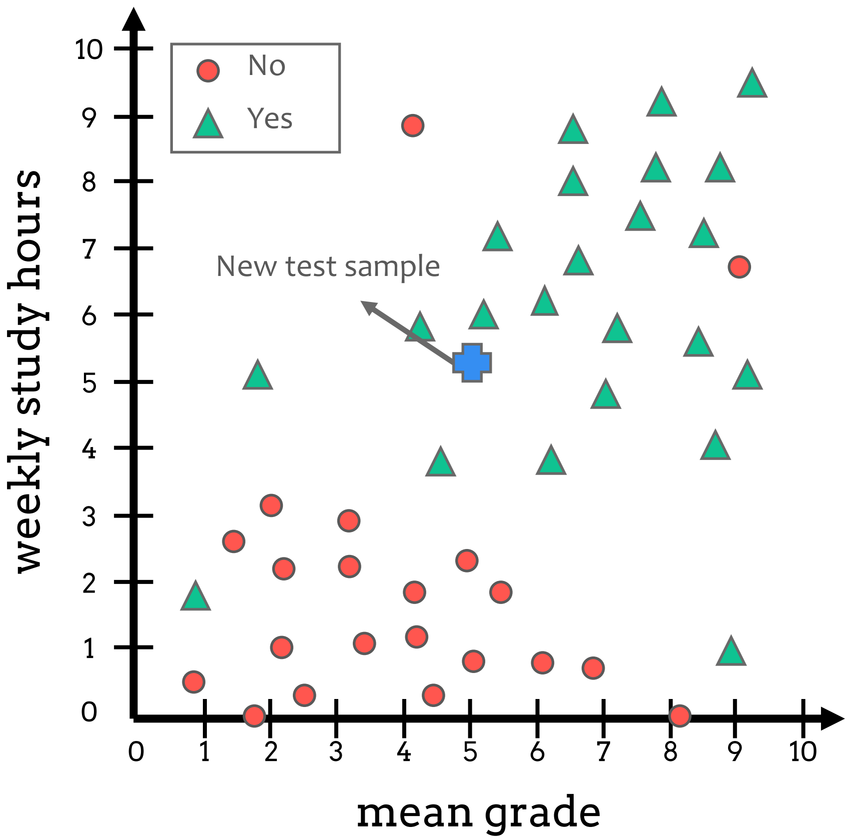


No Training



KNN **does not** learn any model.

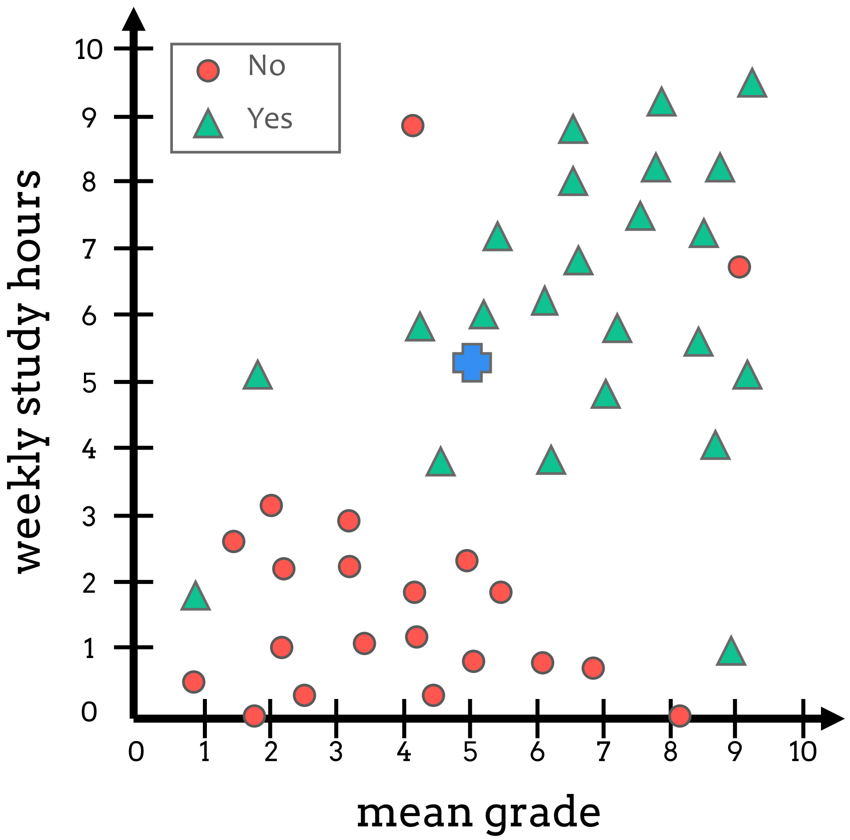
Classification



Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;

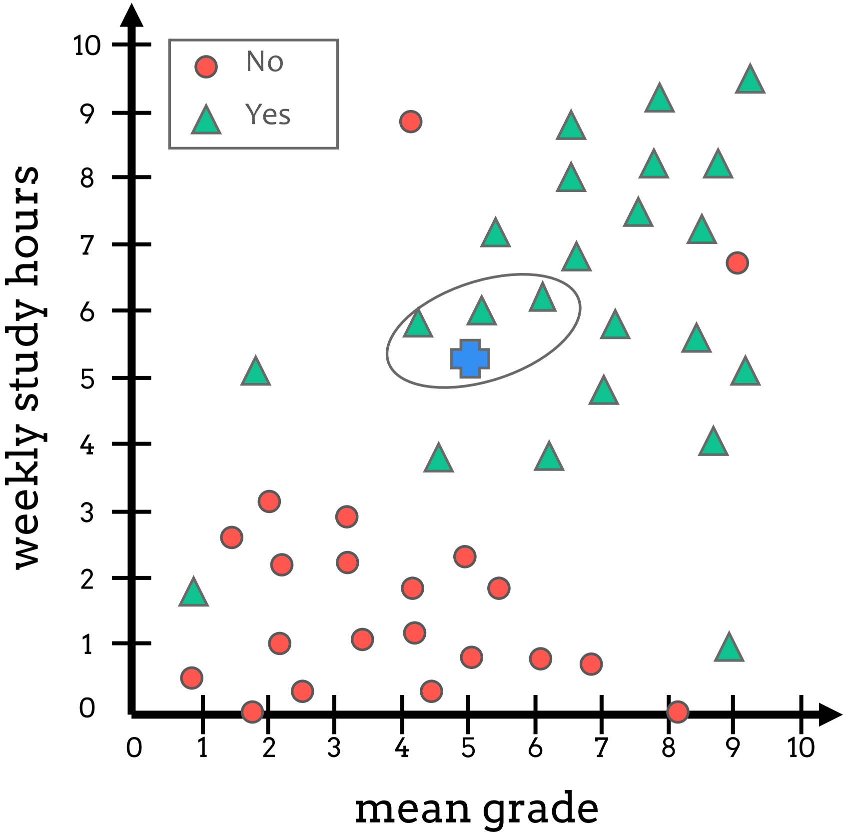


k = 3

Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);

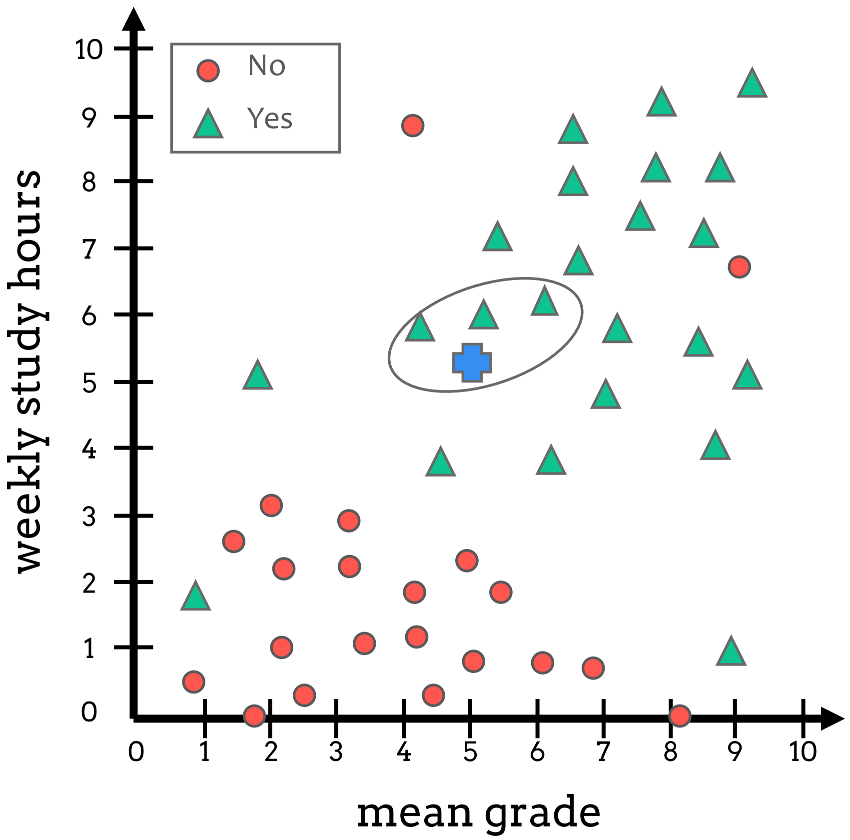


k = 3

Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.



k = 3

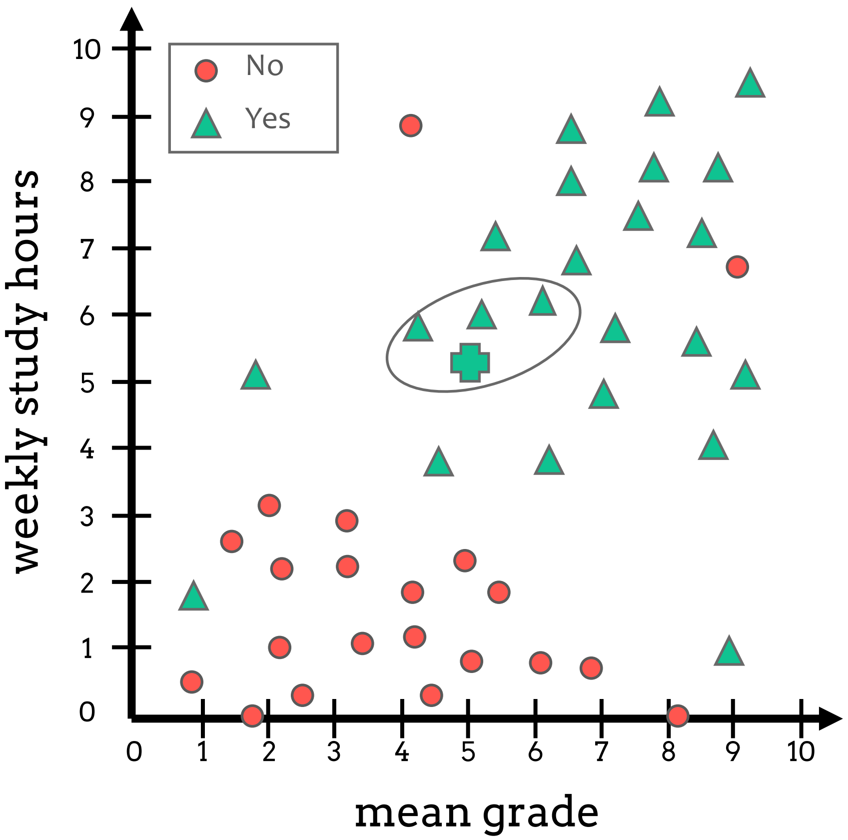
No = 0

Yes = 3

Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.
- **Step 4:** Assign the **new test instance** to **the most frequent class** (majority voting).



Yes

k = 3

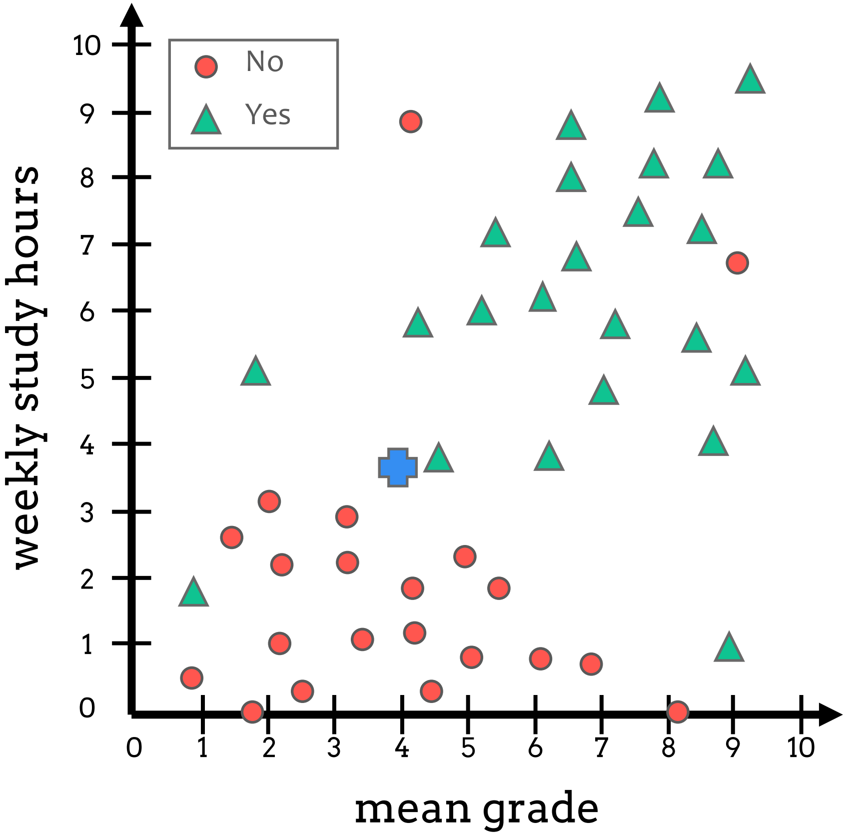
No = 0

Yes = 3

Classification

hyperparameter

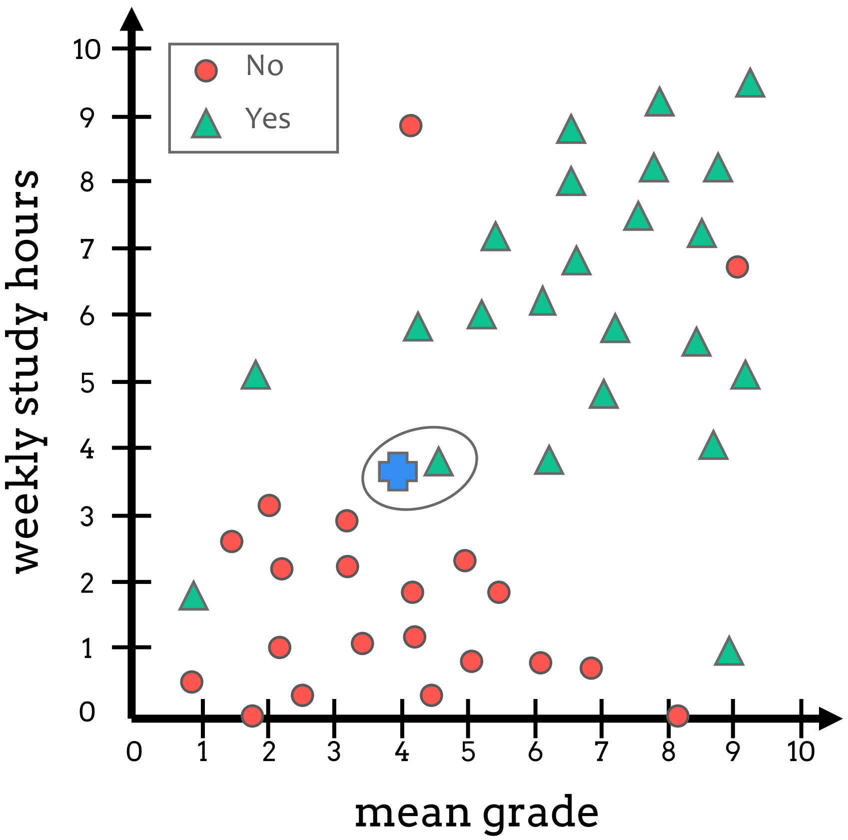
- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.
- **Step 4:** Assign the **new test instance** to **the most frequent class** (majority voting).



Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.
- **Step 4:** Assign the **new test instance** to **the most frequent class** (majority voting).



Yes

k = 1

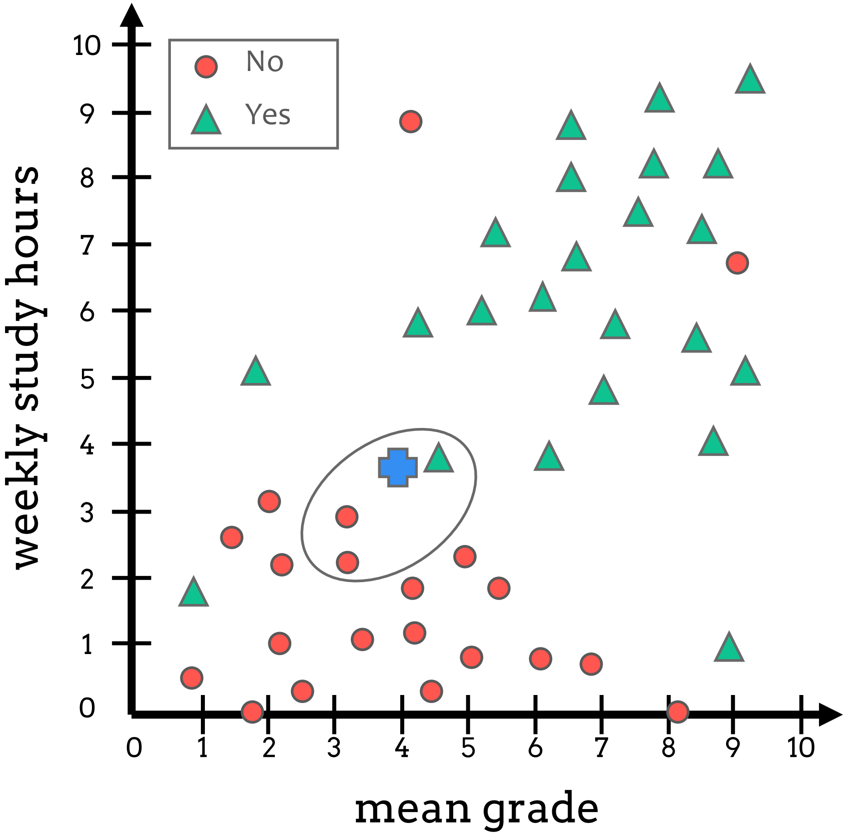
No = 0

Yes = 1

Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.
- **Step 4:** Assign the **new test instance** to **the most frequent class** (majority voting).



No

k = 3

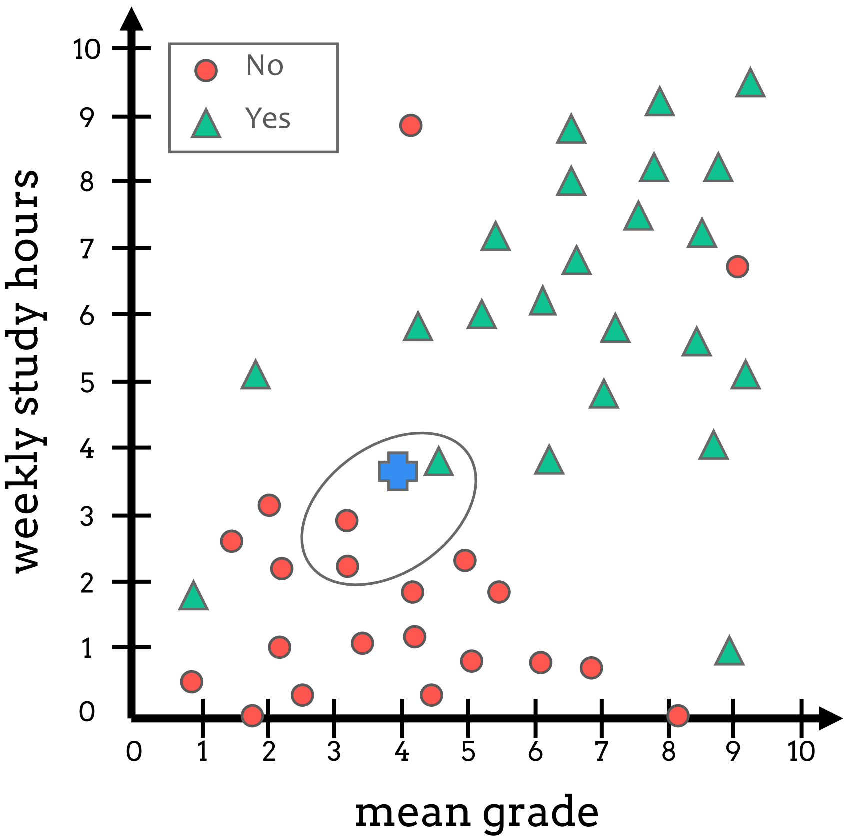
No = 2

Yes = 1

Classification

hyperparameter

- **Step 1:** Choose the number **K of neighbors**;
- **Step 2**: Take the K nearest neighbors of the **new instance**, according to a given **distance measure** (*e.g.*, Euclidean);
- **Step 3:** Among these K neighbors, **count** the number of training instances in each class/category.
- **Step 4:** Assign the **new test instance** to **the most frequent class** (majority voting).



No

k = 3

No = 2

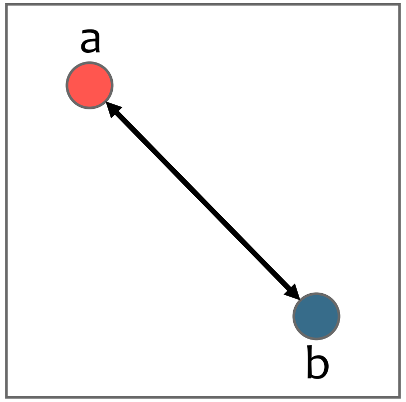
Yes = 1

`sklearn.neighbors`.KNeighborsClassifier

Common Distance Measures

Common Distance Measures

Euclidean

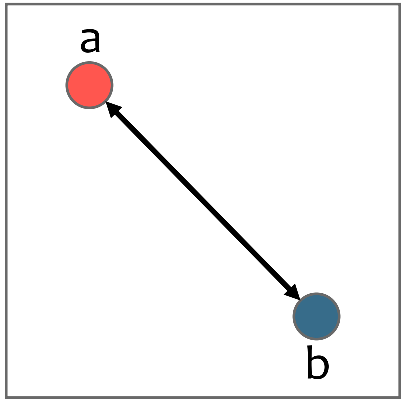


$$d(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

- Common distance measure;
- Suitable for low-dimensional data;

Common Distance Measures

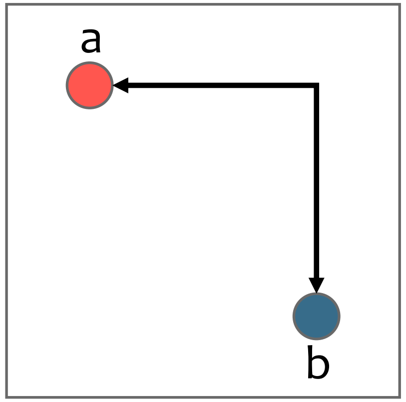
Euclidean



$$d(a,b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2}$$

- Common distance measure;
- Suitable for low-dimensional data;

Manhattan

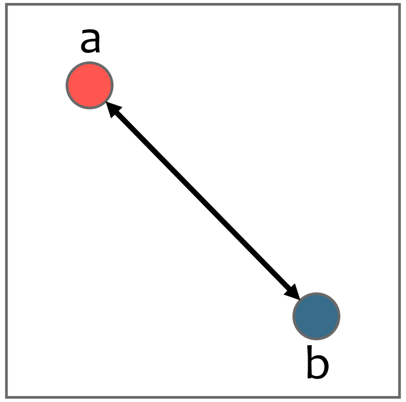


$$d(a,b) = \sum_{i=1}^{n} |b_i - a_i|$$

- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;

Common Distance Measures

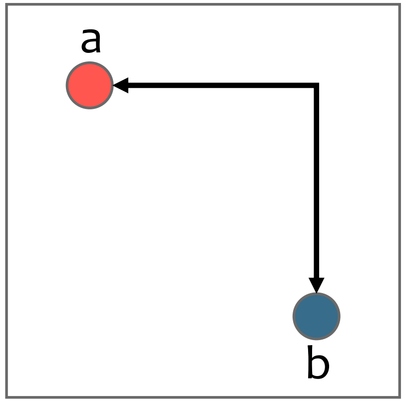
Euclidean



$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

- Common distance measure;
- Suitable for low-dimensional data;

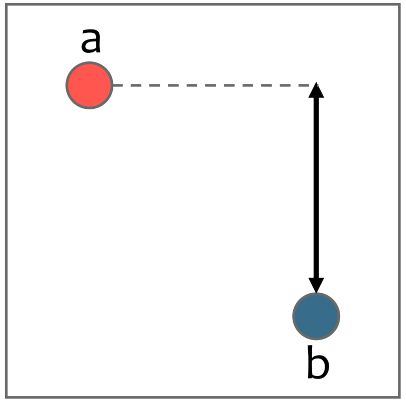
Manhattan



$$d(a,b) = \sum_{i=1}^{n}|b_i - a_i|$$

- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;

Chebyshev

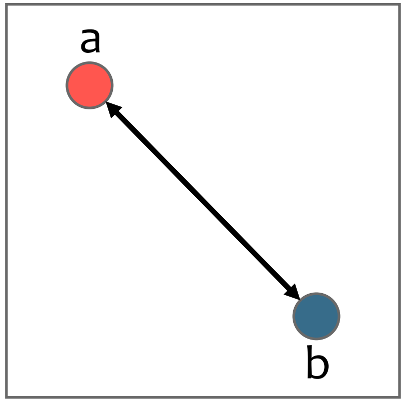


$$d(a,b) = \max_i(|b_i - a_i|)$$

- It can be used to extract the minimum number of moves needed to get from one square to another;
- Used in very specific use-cases, such as warehouse logistics;

Common Distance Measures

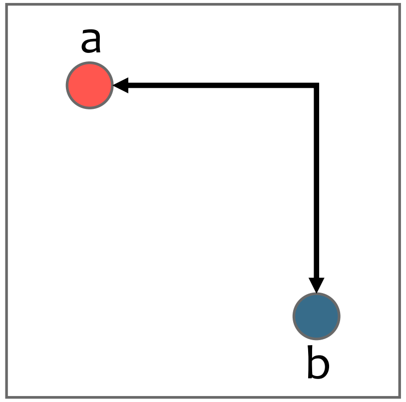
Euclidean



$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

- Common distance measure;
- Suitable for low-dimensional data;

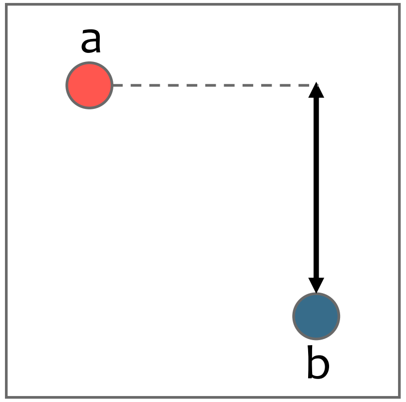
Manhattan



$$d(a,b) = \sum_{i=1}^{n}|b_i - a_i|$$

- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;

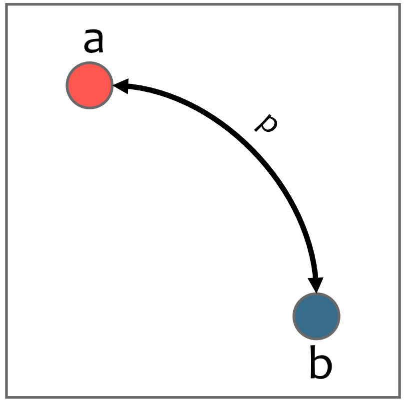
Chebyshev



$$d(a,b) = \max_i(|b_i - a_i|)$$

- It can be used to extract the minimum number of moves needed to get from one square to another;
- Used in very specific use-cases, such as warehouse logistics;

Minkowski

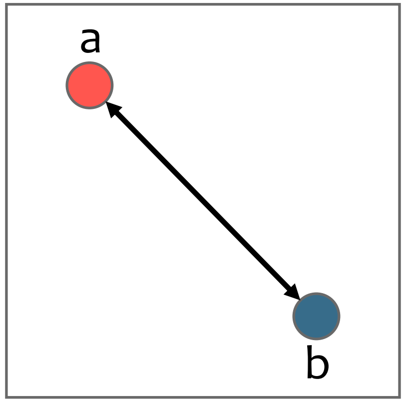


$$d(a,b) = \left(\sum_{i=1}^{n}|b_i - a_i|^p\right)^{\frac{1}{p}}$$

- Metric in a normed vector space;
- The upside to p is the possibility to iterate over it and find the distance measure that works best for your use case.
- p=1 → Manhattan distance
- p=2 → Euclidean distance
- p=∞ → Chebyshev distance

Common Distance Measures

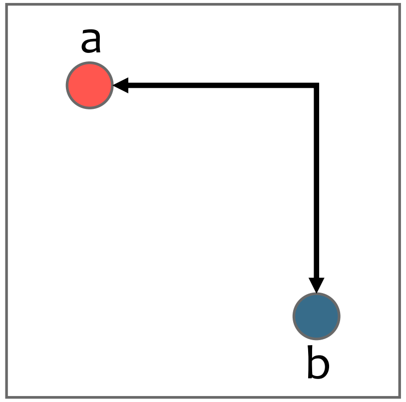
Euclidean



$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

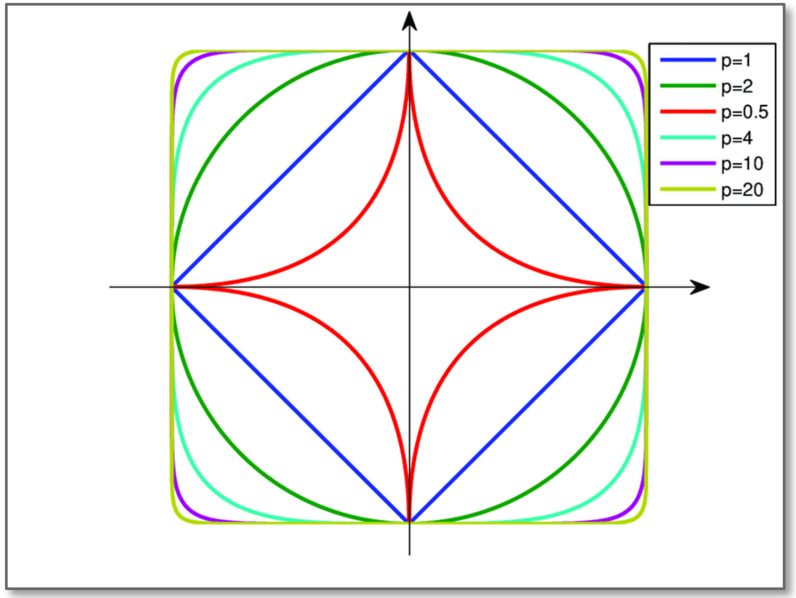
- Common distance measure;
- Suitable for low-dimensional data;

Manhattan



$$d(a,b) = \sum_{i=1}^{n}|b_i - a_i|$$

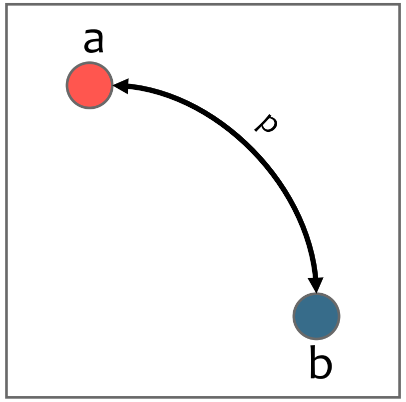
- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;



$$d(a,b) = \max_i(|b_i - a_i|)$$

- It can be used to extract the minimum number of moves needed to get from one square to another;
- Used in very specific use-cases, such as warehouse logistics;

Minkowski

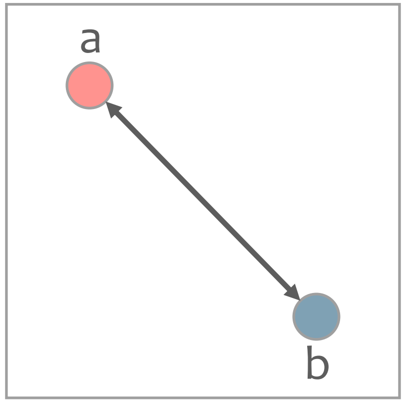


$$d(a,b) = \left(\sum_{i=1}^{n}|b_i - a_i|^p\right)^{\frac{1}{p}}$$

- Metric in a normed vector space;
- The upside to p is the possibility to iterate over it and find the distance measure that works best for your use case.
- p=1 → Manhattan distance
- p=2 → Euclidean distance
- p=∞ → Chebyshev distance

Common Distance Measures

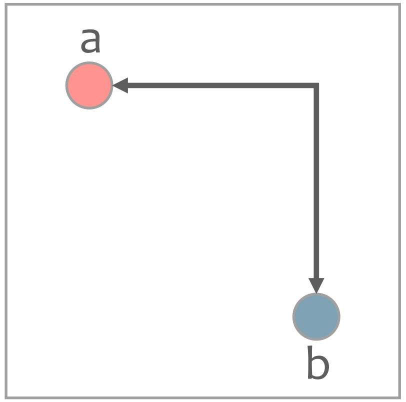
Euclidean



$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

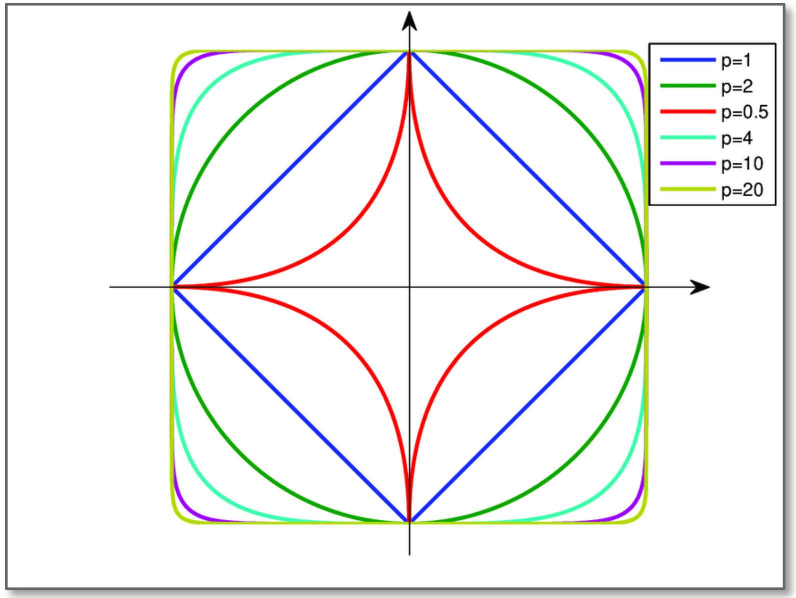
- Common distance measure;
- Suitable for low-dimensional data;

Manhattan



$$d(a,b) = \sum_{i=1}^{n}|b_i - a_i|$$

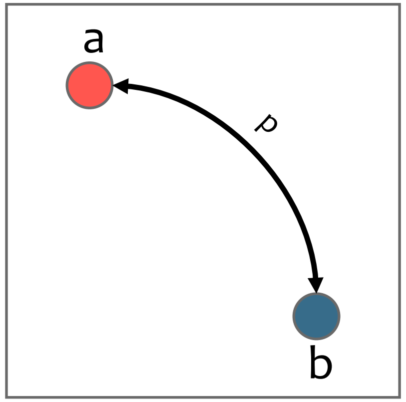
- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;



$$d(a,b) = \max_i(|b_i - a_i|)$$

- It can be used to extract the minimum number of moves needed to get from one square to another;
- Used in very specific use-cases, such as warehouse logistics;

Minkowski

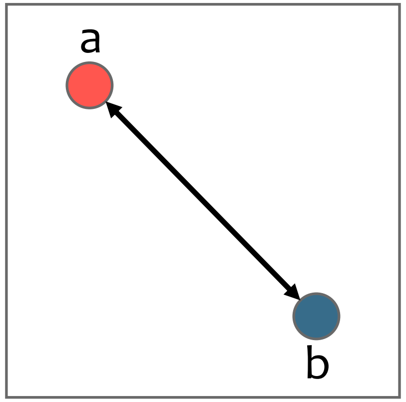


$$d(a,b) = \left(\sum_{i=1}^{n}|b_i - a_i|^p\right)^{\frac{1}{p}}$$

- Metric in a normed vector space;
- The upside to p is the possibility to iterate over it and find the distance measure that works best for your use case.
- p=1 → Manhattan distance
- p=2 → Euclidean distance
- p=∞ → Chebyshev distance

Common Distance Measures

Euclidean

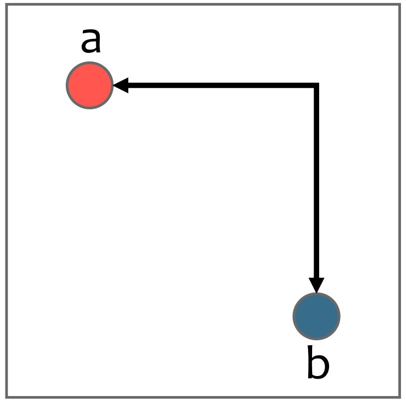


$$d(a,b) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

- Common distance measure;
- Suitable for low-dimensional data;

Typically, these metrics require **feature scaling**.

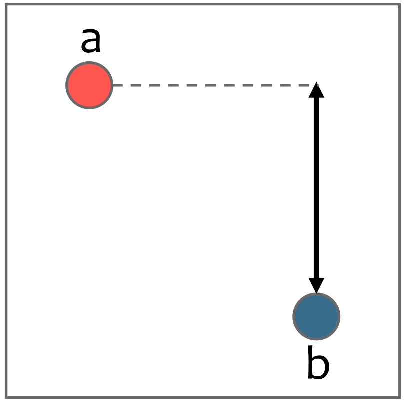
Manhattan



$$d(a,b) = \sum_{i=1}^{n}|b_i - a_i|$$

- Work quite well when your data has discrete and/or binary attributes;
- Work ok for high-dimensional data;
- Less intuitive than Euclidean distance;
- In general, give a higher distance value than Euclidean distance;

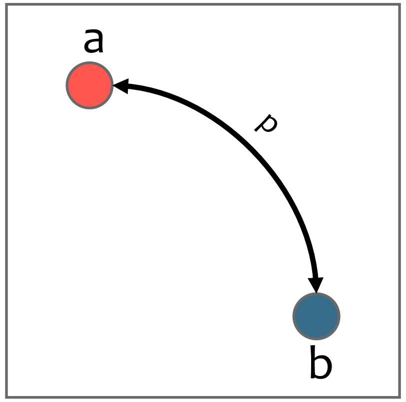
Chebyshev



$$d(a,b) = \max_i(|b_i - a_i|)$$

- It can be used to extract the minimum number of moves needed to get from one square to another;
- Used in very specific use-cases, such as warehouse logistics;

Minkowski

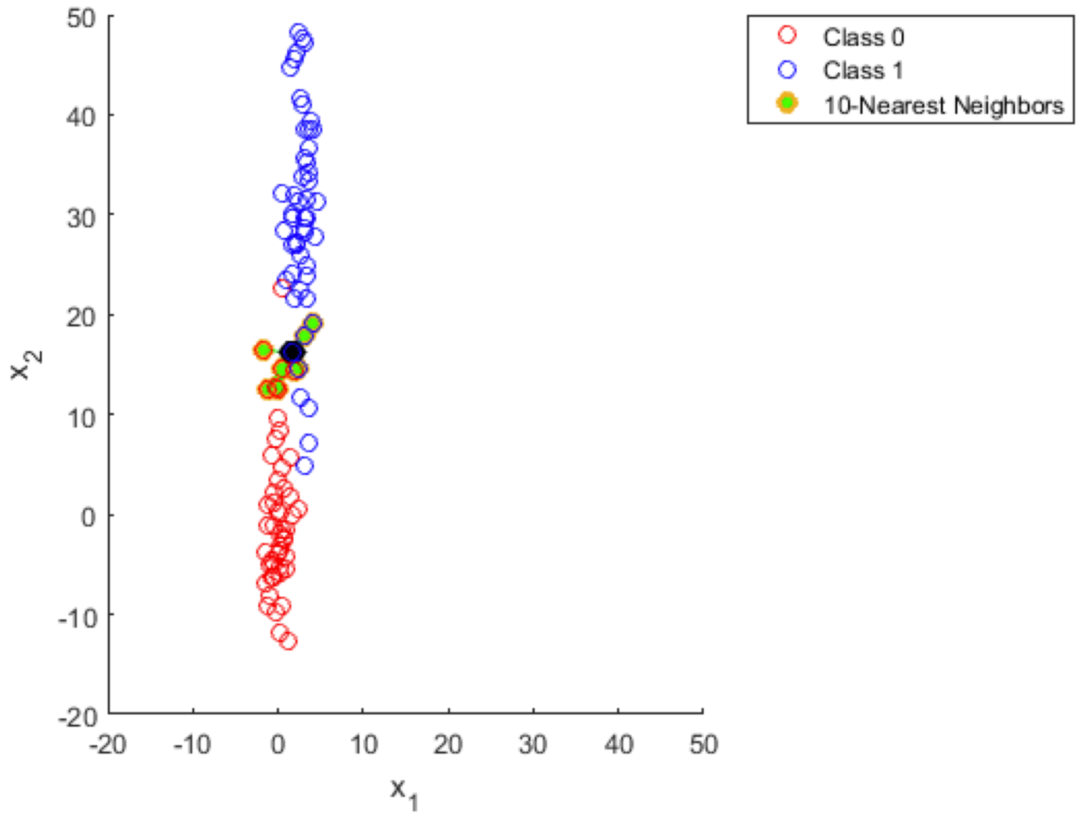


$$d(a,b) = \left(\sum_{i=1}^{n}|b_i - a_i|^p\right)^{\frac{1}{p}}$$

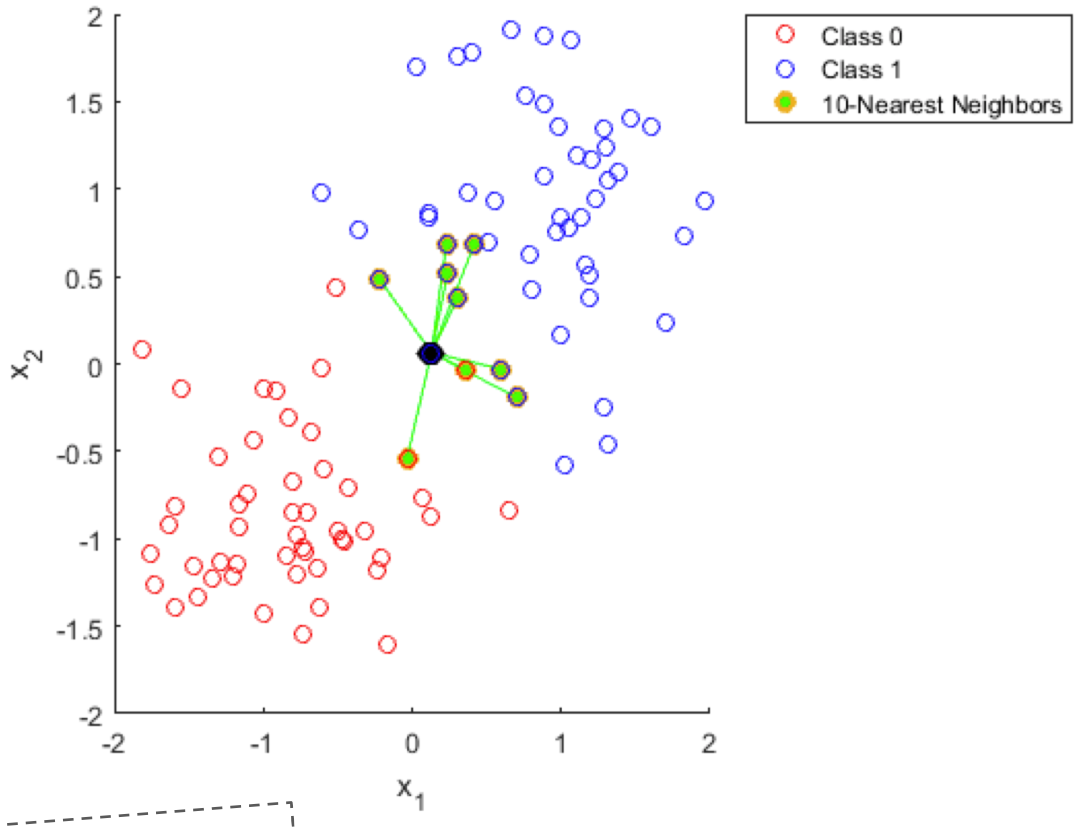
- Metric in a normed vector space;
- The upside to p is the possibility to iterate over it and find the distance measure that works best for your use case.
- p=1 → Manhattan distance
- p=2 → Euclidean distance
- p=∞ → Chebyshev distance

KNN with and without Feature Scaling

Data without Feature Scaling



Data with Feature Scaling



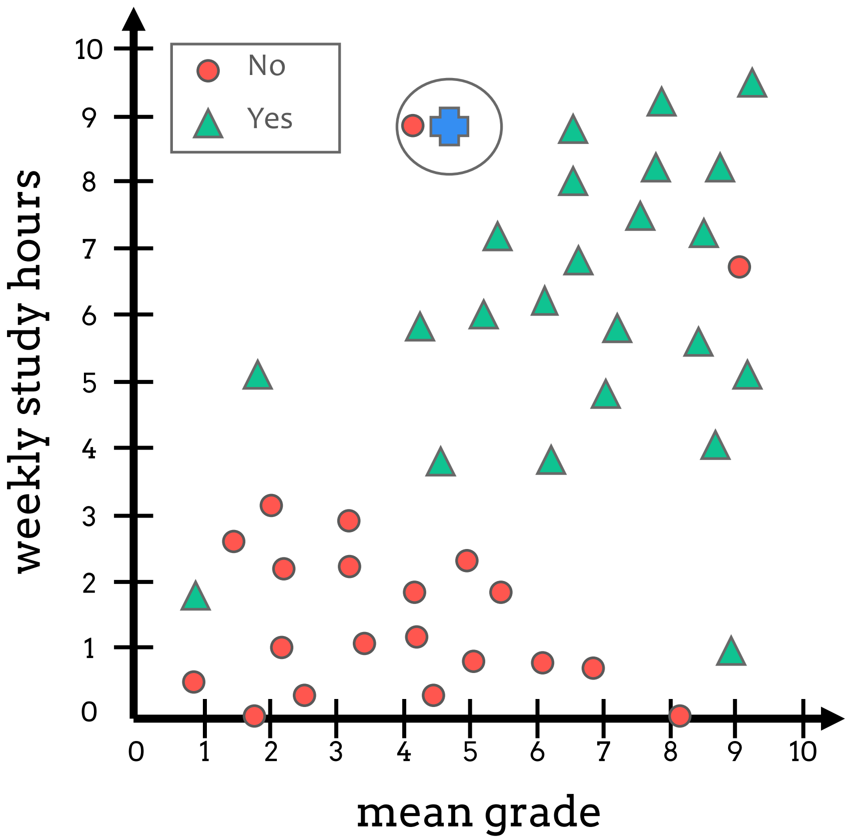
KNN works properly after **feature scaling** your data.

https://stats.stackexchange.com/a/287439

Choosing the value of K

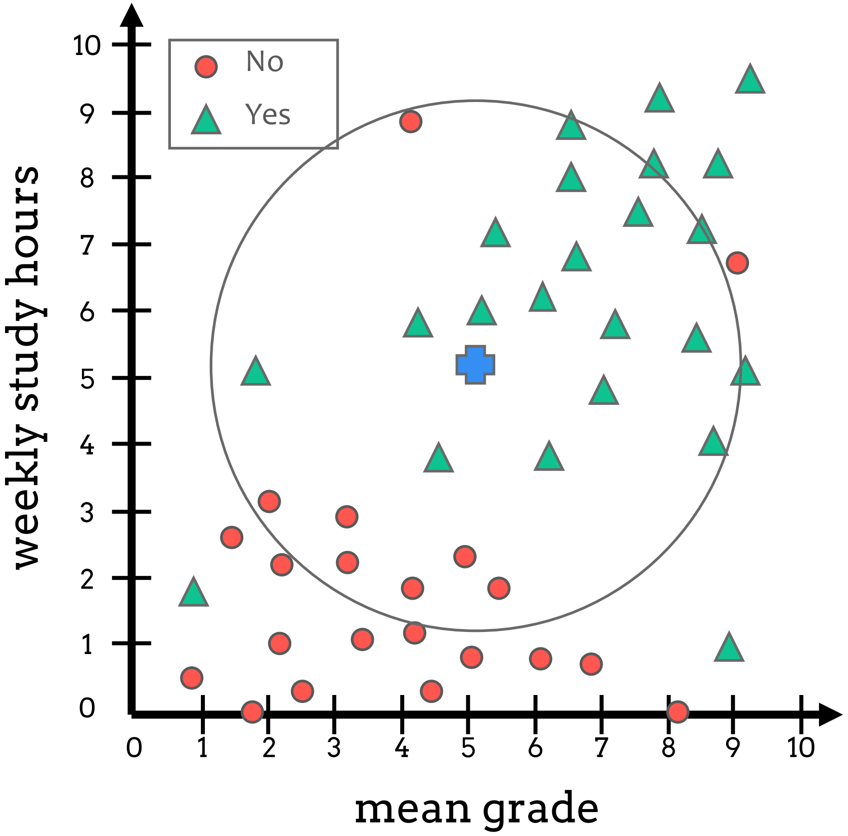
Choosing the value of K

- If **K is too small:**
  - KNN is sensitive to *outliers*;
  - KNN overfits the training data:
    - Higher error on different sets;



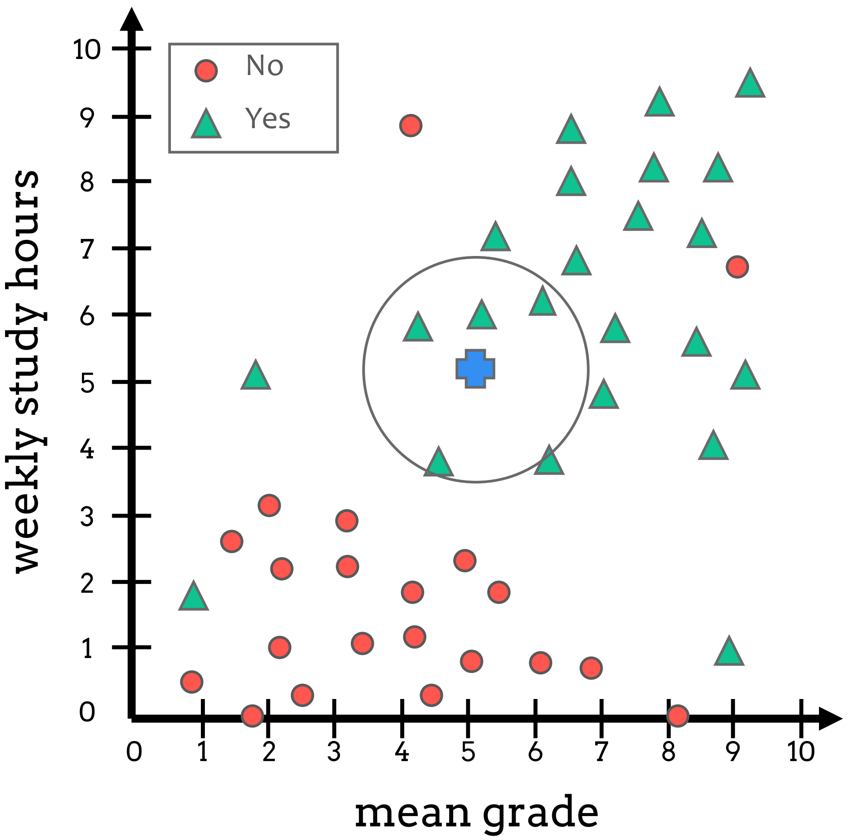
Choosing the value of K

- If **K is too small:**
  - KNN is sensitive to *outliers*;
  - KNN overfits the training data:
    - Higher error on different sets;
- If **K is too large:**
  - Neighbors can include samples from other classes;
  - KNN performs poorly on both train and validation set;



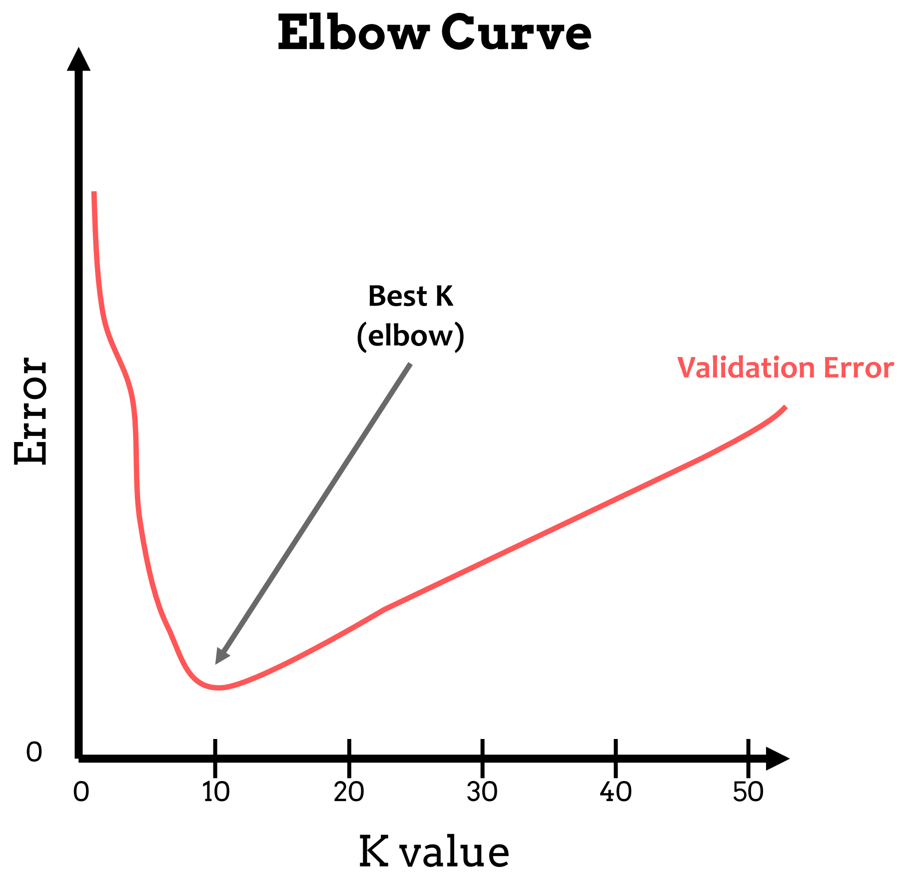
Choosing the value of K

- If **K is too small:**
  - KNN is sensitive to *outliers*;
  - KNN overfits the training data:
    - Higher error on different sets;
- If **K is too large:**
  - Neighbors can include samples from other classes;
  - KNN performs poorly on both train and validation set;
- **K** is a **hyperparameter** so we can **optimize it** for our problem;
  - For example, by **Cross-Validation Grid Search**



Choosing the value of K

- If **K is too small:**
  - KNN is sensitive to *outliers*;
  - KNN overfits the training data:
    - Higher error on different sets;
- If **K is too large:**
  - Neighbors can include samples from other classes;
  - KNN performs poorly on both train and validation set;
- **K** is a **hyperparameter** so we can **optimize it** for our problem;
  - For example, by **Cross-Validation Grid Search**
  - By plotting an **elbow curve**



KNN: Pros and Cons

Pros

1. Extremely easy to implement it;
2. It does not require training;
3. By **not** requiring **training** before making estimation/classifications, **new training samples** can be added without any problems (no models' retraining);
4. There is ***only* a single hyperparameter** required by KNN
    - Number of neighbors K
    - If we consider other distances, we can have more required hyperparameters (Minkowski, ...);

KNN: Pros and Cons

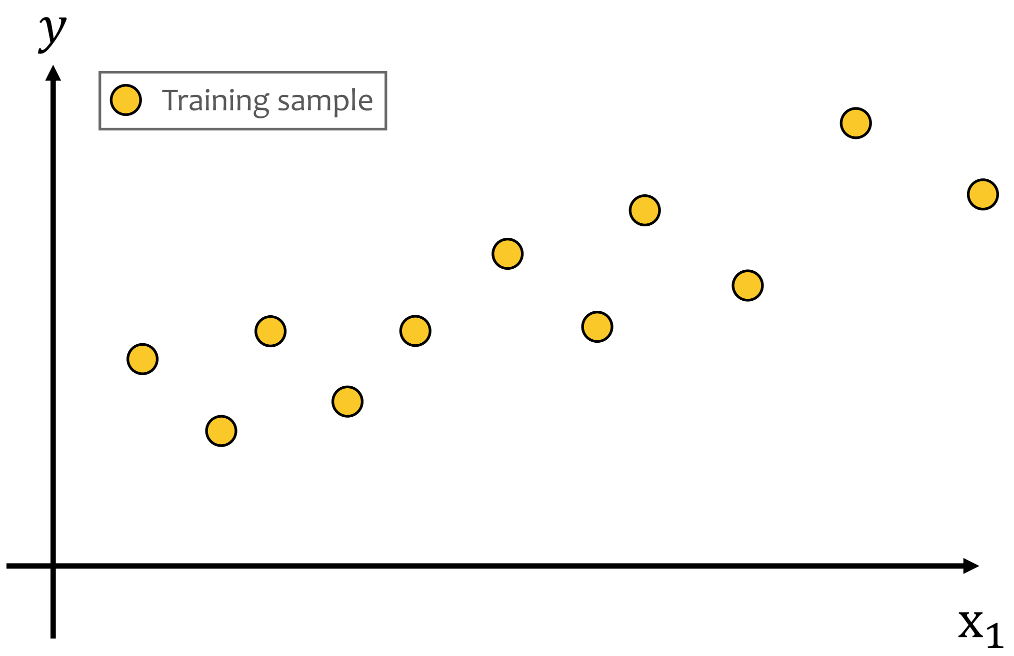
Pros

1. Extremely easy to implement it;
2. It does not require training;
3. By **not** requiring **training** before making estimation/classifications, **new training samples** can be added without any problems (no models' retraining);
4. There is ***only*** **a single hyperparameter** required by KNN
   - Number of neighbors K
   - If we consider other distances, we can have more required hyperparameters (Minkowski, ...);

Cons

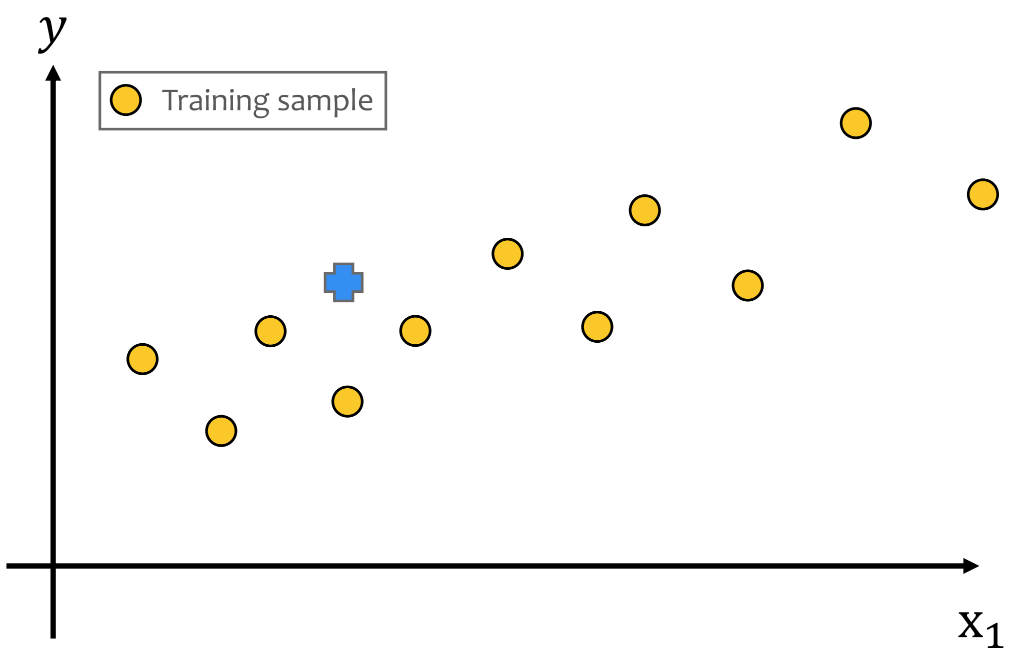
1. It does not work well for **high dimensionality data (the curse of dimensionality)**
2. The **prediction time** can be **high** if the size of the training set is large;

Variant: KNN Regression



Variant: KNN Regression

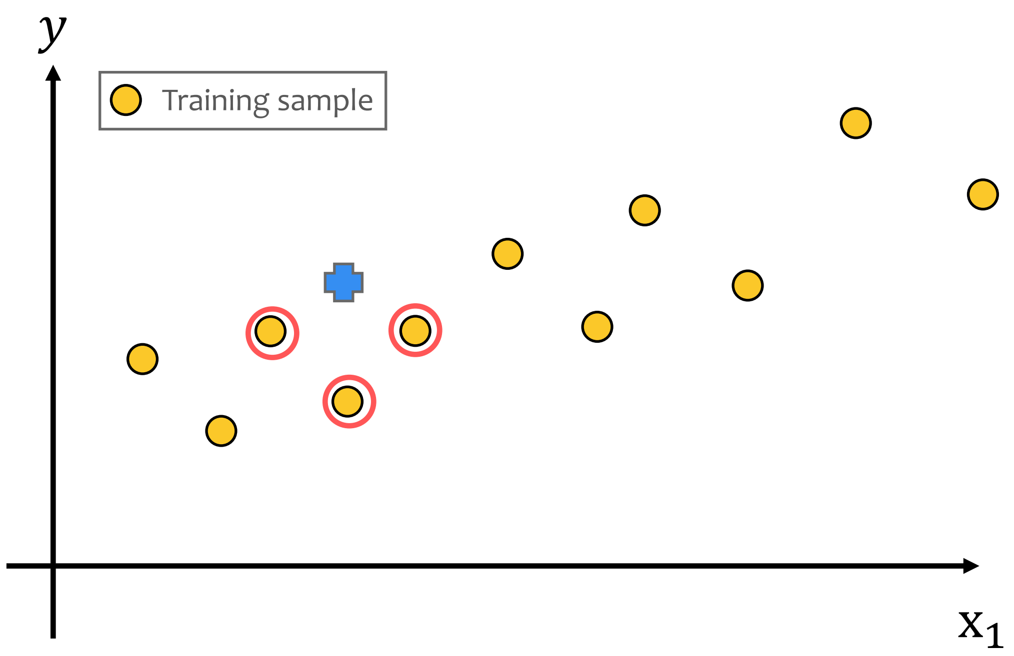
The **outcome (label / dependent variable)** of a **new test sample** is computed based on the **mean of the outcomes/labels** of its K nearest neighbors.



k = 3

Variant: KNN Regression

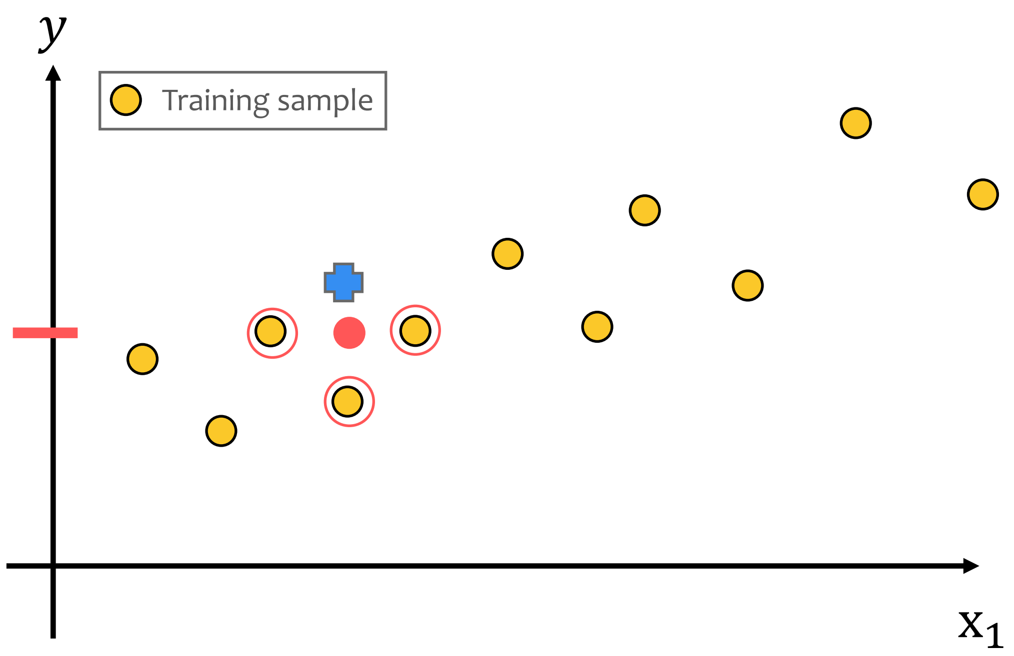
The **outcome (label / dependent variable)** of a **new test sample** is computed based on the **mean of the outcomes/labels** of its K nearest neighbors.



k = 3

Variant: KNN Regression

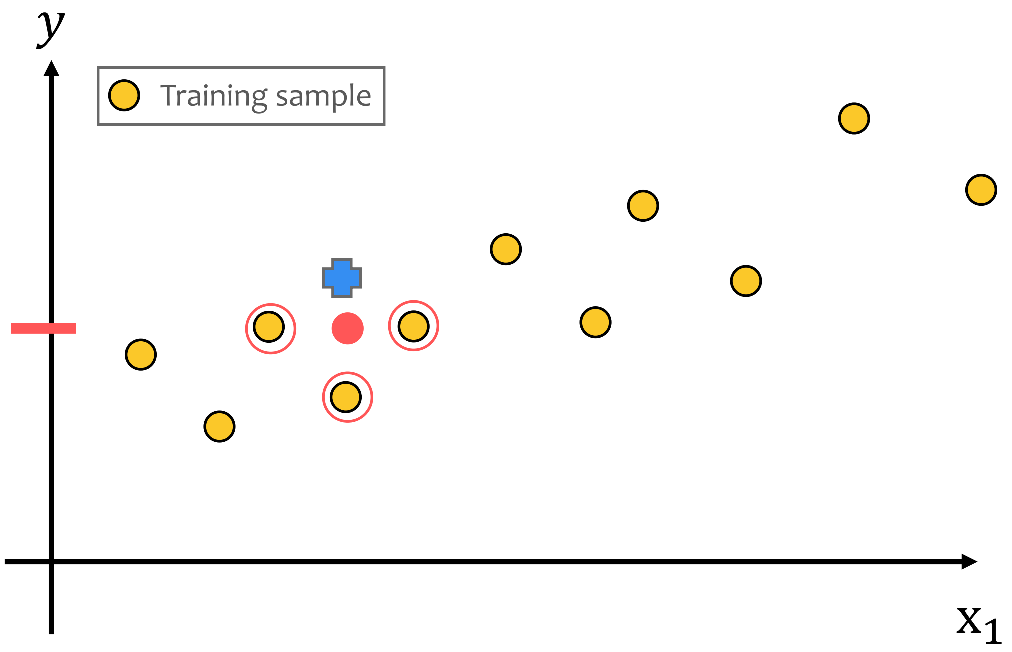
The **outcome (label / dependent variable)** of a new test sample is computed based on the **mean of the outcomes/labels** of its K nearest neighbors.



k = 3

Variant: KNN Regression

The **outcome (label / dependent variable)** of a **new test sample** is computed based on the **mean of the outcomes/labels** of its K nearest neighbors.



k = 3

`sklearn.neighbors.KNeighborsRegressor`

Aprendizado de Máquina e Reconhecimento de Padrões

K-Nearest Neighbors (KNN)

Prof. Dr. Samuel Martins (Samuka)
samuel.martins@ifsp.edu.br

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo
Campus Campinas