TICS-411 Minería de Datos

Clase 8: Introducción al Aprendizaje Supervisado

Alfonso Tobar-Arancibia
alfonso.tobar.a@edu.uai.cl

Aprendizaje Supervisado

Introducción al Aprendizaje Supervisado

Los modelos Predictivos/Supervisados tienen la capacidad de predecir valores en datos no vistos.

Chip Huyen, Designing ML Systems (Profesora de Stanford)

> "Machine Learning Algorithms do not predict the future but encode the past, thus perpetuating the biases in the data and mode…"

Aprenden mediante un proceso de `entrenamiento` en un `train set` y evalúan su performance/rendimiento utilizando un `test set`.

Definiciones

Features

> También llamadas `variables` o `atributos`. Corresponden al ***input*** del Modelo y con el cuál el modelo ***aprende*** y ***predice***. Normalmente es representado mediante una Matriz denominada X.

Labels o Etiquetas

> Corresponden a las ***respuestas*** que el modelo necesita mapear para poder descubrir patrones de manera automática. Normalmente se representa mediante un vector denominado y.

Ejemplo

> Queremos generar un algoritmo de aprendizaje tal que dado un cierto set de datos predigamos si es que a un niño se le dará o no permiso para jugar.

Problema de Clasificación Binaria (Dos clases opuestas).

	Outlook	Temperatura	Humedad	Hay viento?	Juega?
1	Soleado	29	85	No	NO
2	Soleado	25	90	Sí	NO
3	Nublado	27	86	No	SÍ
4	Lluvioso	21	96	No	SÍ
5	Lluvioso	19	80	No	SÍ
6	Lluvioso	15	70	Sí	NO
7	Nublado	14	65	Sí	SÍ
8	Soleado	22	95	No	NO

Definición del Problema

$$h_\theta(X) = f(X, \theta)$$

- A $h_\theta(\cdot)$ la denominaremos `hipótesis` o simplemente `modelo`.
- X será nuestro set de `features` ($n \times m$ donde n es el número de observaciones y m el número de features).
- Cada fila de X corresponde a un vector x_i que representa una observación de nuestro set de features.
- θ corresponde a los parámetros del modelo *(existen modelos paramétricos y no paramétricos)*.
- Cada `algoritmo` tendrá su propio mapeo $f(\cdot)$ para tratar de *predecir* una etiqueta.

Tipos de Hipótesis

- Si $h_\theta(X)$ devuelve valores discretos (o categóricos) hablaremos de un modelo de `Clasificación`.
- Si $h_\theta(X)$ devuelve valores continuos hablaremos de un modelo de `Regresión`.

Tipos de Problemas

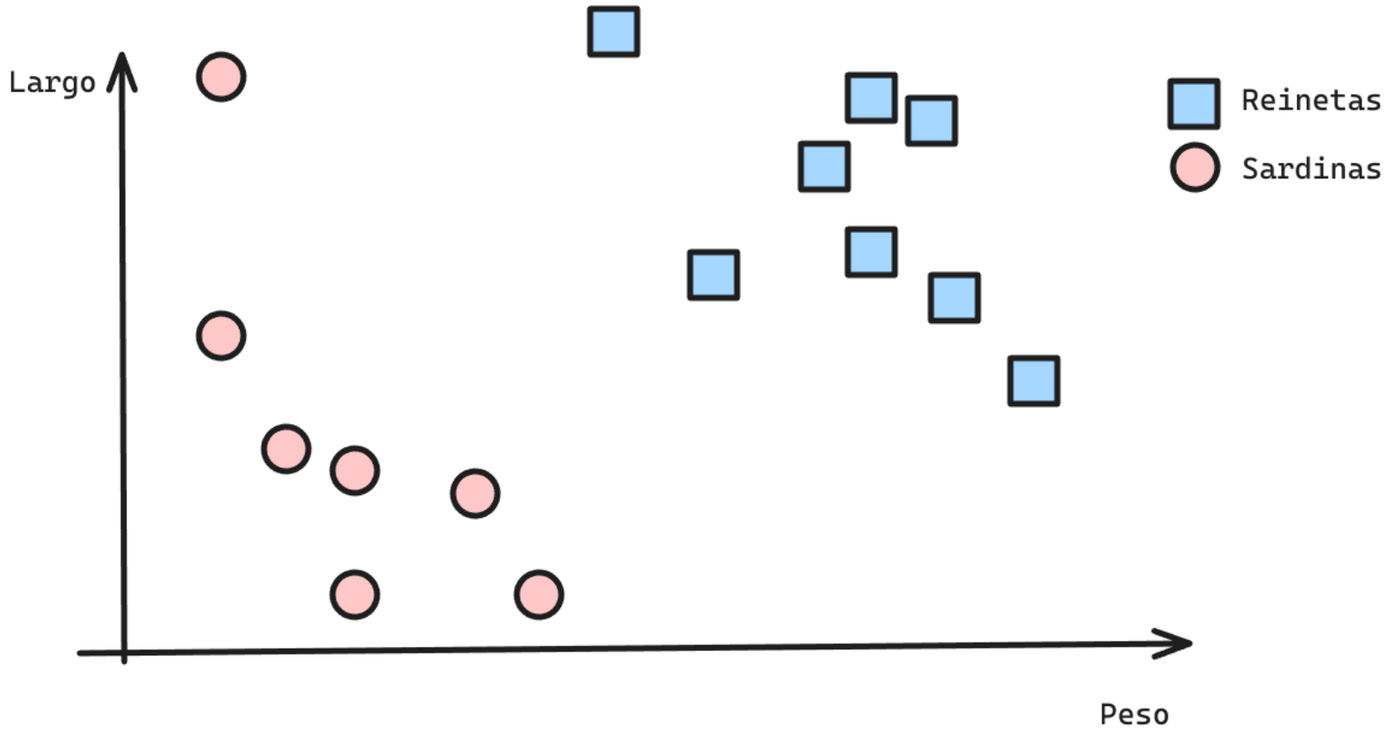
Clasificación:

- **Binaria**: La Clasificación es dicotómica, Perro o Gato, Sí o No, 1 o 0, Clase Positiva o Negativa.
- **Multiclase**: La clasificación puede tener más de 2 clases, pero sólo una es posible.
  - Ej: Perro, Gato o Canario; 0, 1, 2, 3, 4.
- **Multilabel**: La clasificación puede tener más de 2 clases, y más de una es posible a la vez.
  - Ej: Categorías de Libro: Puede ser Romance y Drama, Películas: Fantasía, Animación y Acción.

Regresión:

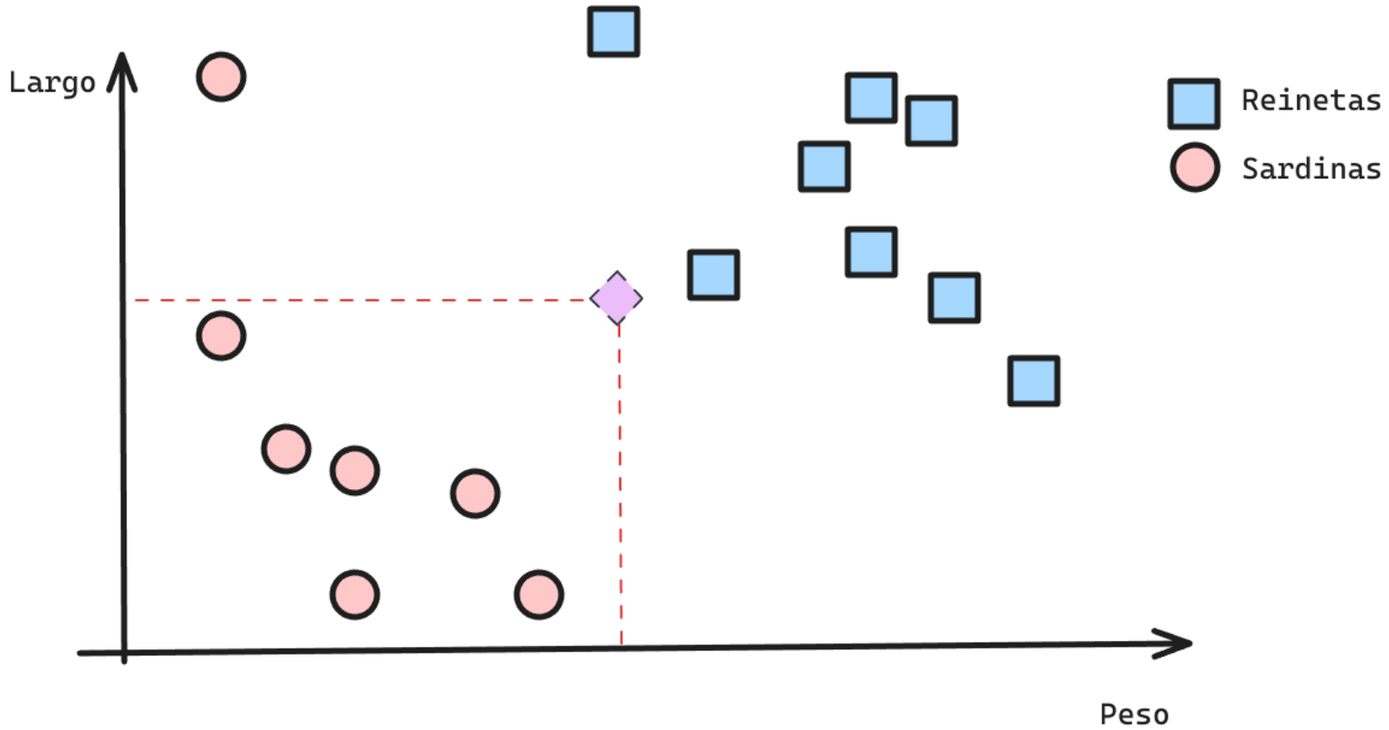
- **Simple**: Predigo sólo un valor. Ej: Predecir la Temperatura.
- **Multiple**: Predigo varios valores continuos a la vez.
  - Ej: Modelo para intentar estimar Temperatura y Humedad a la vez.
- **Forecast**: Donde se utilizan valores pasados para estimar valores futuros.
  - Dadas mis ganancias pasadas, estimar las futuras.

Clasificación: Intuición



> Supongamos el siguiente problema de clasificación. Tenemos un algoritmo, que dadas las variables `Largo` y `Peso` sean capaces de predecir si es que un Pez es una *Reineta* o una *Sardina*.

Clasificación: Intuición



- Queremos encontrar una `Regla de Decisión` (Decision Rule) que permita clasificar correctamente un punto nuevo.
- Distintos modelos son capaces de encontrar distintas reglas de decisión. Por lo tanto, sus predicciones pueden ser completamente distintas.

Clasificación: Detalles

Es importante mencionar que un modelo de clasificación puede generar:

- **Hard Predictions**: Es decir, la instancia a predecir es clase 0 o clase 1.
- **Soft Prediction**: Es decir, la instancia a predecir tiene una probabilidad p de pertenecer a la clase 1 y de $1 - p$ de pertenecer a la clase 0.

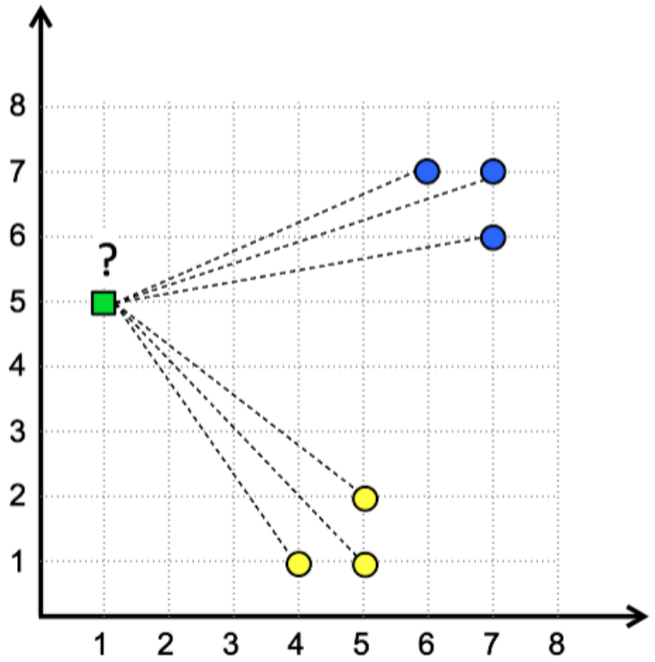
- Cuando se hace predicción binaria, lo común es usar un Threshold de 0.5 para elegir la clase. Es decir si $p < 0$ entonces clase 0, si $p \geq 0.5$ entonces clase 1.

En el caso de predicción multiclase o multilabel. Se calcula la probabilidad para cada clase. Por lo tanto se se asigna la clase de mayor probabilidad.

K-Nearest Neighbors

> El modelo de vecinos más cercanos, o KNN por sus siglas en Inglés es un modelo basado en `distancias`. Su `regla de decisión` se basa en imitar el comportamiento de sus K vecinos más cercanos por votación (para clasificación) o la media (para regresión).

K es un `hiperparámetro` de este modelo.



- Supongamos $K = 3$.
- Es decir, tomaremos los 3 vecinos más cercanos.

En general es una buena idea elegir vecinos impares. **¿Por qué?**

KNN: Paso 1 (Training Time)

Training Time

Corresponde al periodo donde el modelo aprende de los datos. Toma un patrón y ese modelo es utilizado para predecir.

> En el caso de un KNN **NO HAY APRENDIZAJE** en esta etapa.

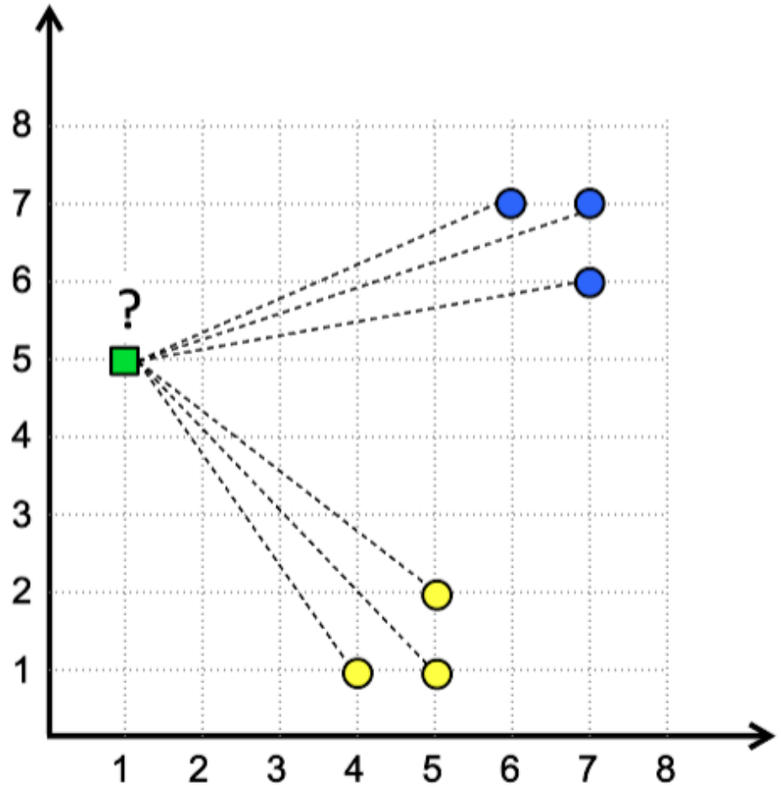
> Es considerado un modelo `no-paramétrico` ya que no aprende parámetros para realizar su predicción.

KNN: Paso 2 (Test Time)

Inference Time

Corresponde al periodo donde el modelo debe emitir una predicción.

En este caso, KNN calcula las distancias del punto a predecir (en verde) a todos los otros puntos existentes (proceso caro).



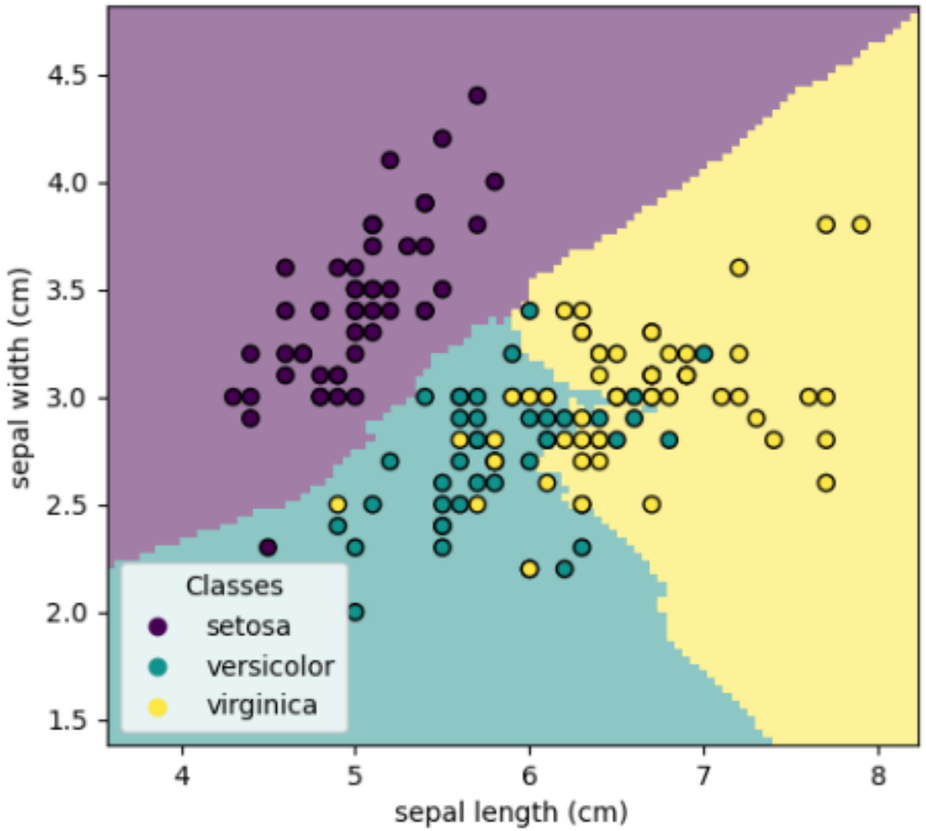
Distancia	Clase
5.00	1
5.66	1
5.00	1
5.39	2
6.08	2
6.32	2

Distancia	Clase
5.00	1
5.00	1
5.39	2
5.66	1
6.08	2
6.32	2

- La predicción corresponderá a la etiqueta mayoritaria por `votación`

- La predicción corresponderá a la etiqueta mayoritaria por `votación`.
- **¿Cuál sería una buena estrategia de predicción para un modelo de Regresión?**

Fronteras de Decisión



- Implicitamente, todo modelo de Machine Learning generará lo que se llama una Frontera de Decisión.
- Si un punto no visto cae dentro de su frontera entonces se le asigna dicha etiqueta.

Implementación Clasificación en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClasifier(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

Implementación Clasificación en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClasifier(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

Implementación Clasificación en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClasifier(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

Implementación Clasificación en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClasifier(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

Implementación Clasificación en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClasifier(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

Implementación Regresión en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsRegressor

knn_clf = KNeighborsRegressor(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

¿Cómo se encuentran las predicciones en un modelo de **Regresión**?

Implementación Regresión en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsRegressor

knn_clf = KNeighborsRegressor(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

¿Cómo se encuentran las predicciones en un modelo de **Regresión**?

Implementación Regresión en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsRegressor

knn_clf = KNeighborsRegressor(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

¿Cómo se encuentran las predicciones en un modelo de **Regresión**?

Implementación Regresión en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsRegressor

knn_clf = KNeighborsRegressor(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

¿Cómo se encuentran las predicciones en un modelo de **Regresión**?

Implementación Regresión en Scikit-Learn

```python
from sklearn.neighbors import KNeighborsRegressor

knn_clf = KNeighborsRegressor(n_neighbors = 5, metric="minkowski", p=2, n_jobs=-1)
knn_clf.fit(X, y)

# Predicción...
y_pred = knn_clf.predict(X)
```

- **n_neighbors**: K número de vecinos a utilizar. Por defecto 5.
- **metric**: Métrica de distancia. Por defecto “Minkowski”.
- **p**: Potencia de Minkowski: $p = 1$, Manhattan, $p = 2$ Euclideana. Por defecto $p = 2$.
- **n_jobs**: Corresponde a un parámetro interno para poder paralelizar los cálculos. Se recomienda utilizar -1 para utilizar todos sus cores.

¿Cómo se encuentran las predicciones en un modelo de **Regresión**?

KNN: Detalles Técnicos

ⓘ Fortalezas

- Modelo muy simple de implementar y entender.
- Muy eficiente en el ***aprendizaje***.

⚠ Debilidades

- Inferencia ineficiente: $O(mn^2)$.
- `Curse of Dimensionality`: A medida que el número de dimensiones del problema crece, se requiere un incremento exponencial en la cantidad de datos para asegurar que existen suficientes vecinos cercanos para cualquier punto.

Saionara!