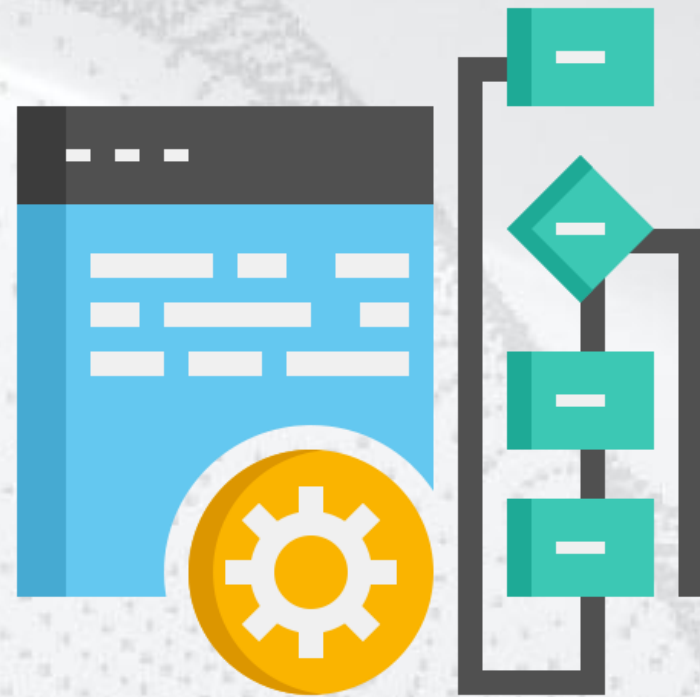


Módulo 5 – Aprendizaje de Máquina Supervisado

# K-Nearest Neighbor

Especialización en Ciencia de Datos

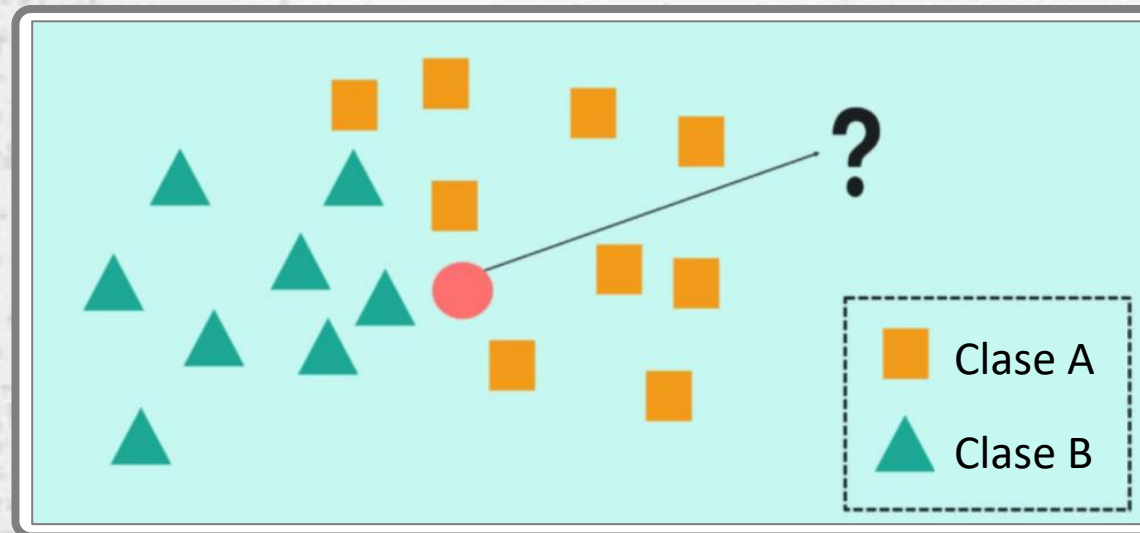
# K-Nearest Neighbor



# El Algoritmo K-NN

El algoritmo KNN (K-Nearest Neighbors) es un algoritmo de aprendizaje supervisado utilizado en problemas de clasificación y regresión. El objetivo del algoritmo KNN, es encontrar las  $k$  observaciones más cercanas en el conjunto de entrenamiento para una observación de prueba y predecir la clase o el valor de respuesta de la observación de prueba basándose en las observaciones cercanas.

El algoritmo KNN se basa en la idea de que las observaciones similares tienden a tener etiquetas similares o valores de respuesta similares. Es uno de los más simples de todos los algoritmos de aprendizaje de máquina, sin embargo, es uno de los top 10 algoritmos más utilizados en Data Mining.

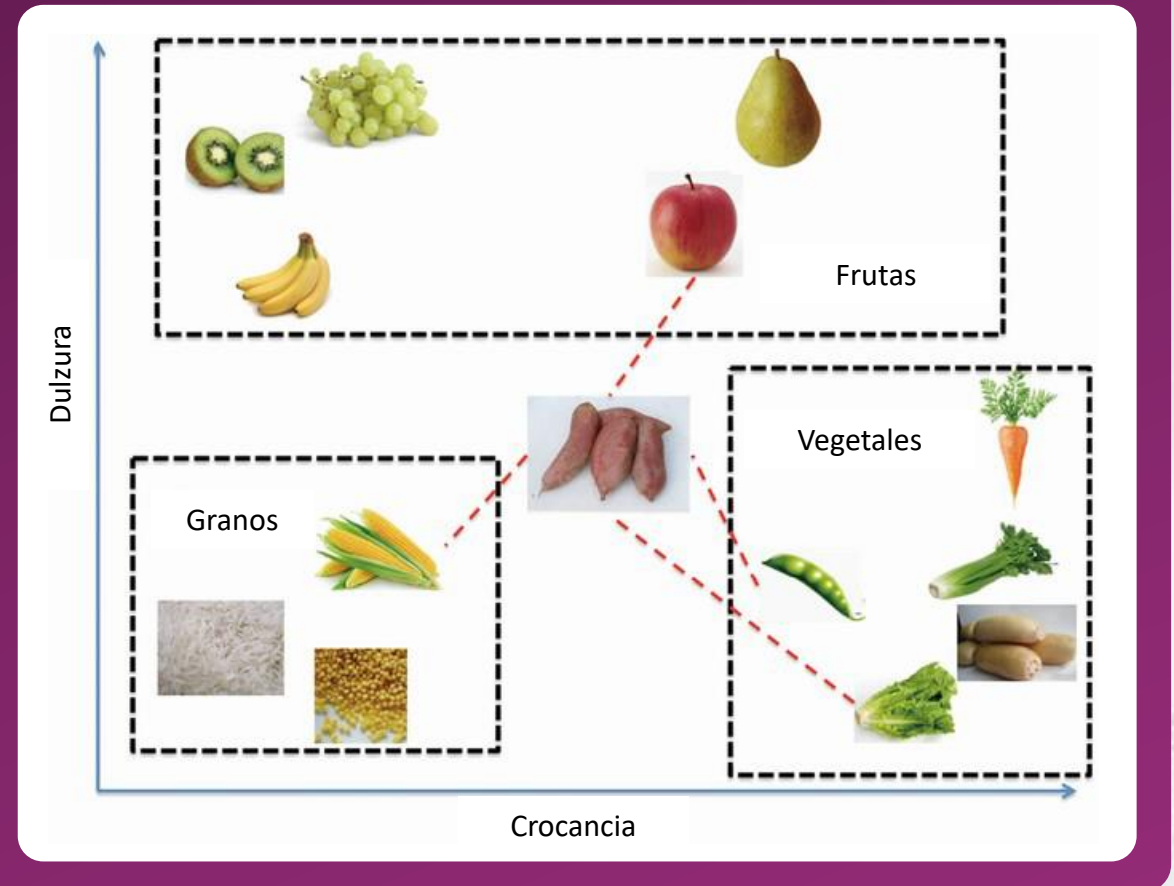




# Algoritmo K-NN

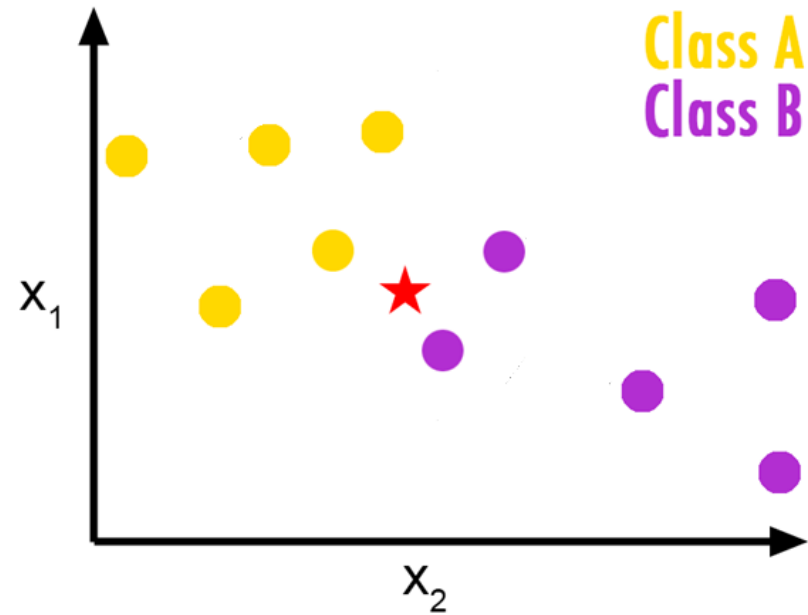
Un ejemplo podría ser un modelo de clasificación de productos naturales en base a su crocancia y su dulzor. Como se puede observar en la figura, existen tres clases de productos naturales: frutas, granos y vegetales. Cada uno de ellos caracterizados debidamente con sus niveles de dulzor y crocancia.

Entonces, si aparece un nuevo producto del cual desconozco su clasificación, y mido sus características de dulzor y crocancia. ¿Podré etiquetarlo como alguna de esas 3 categorías?



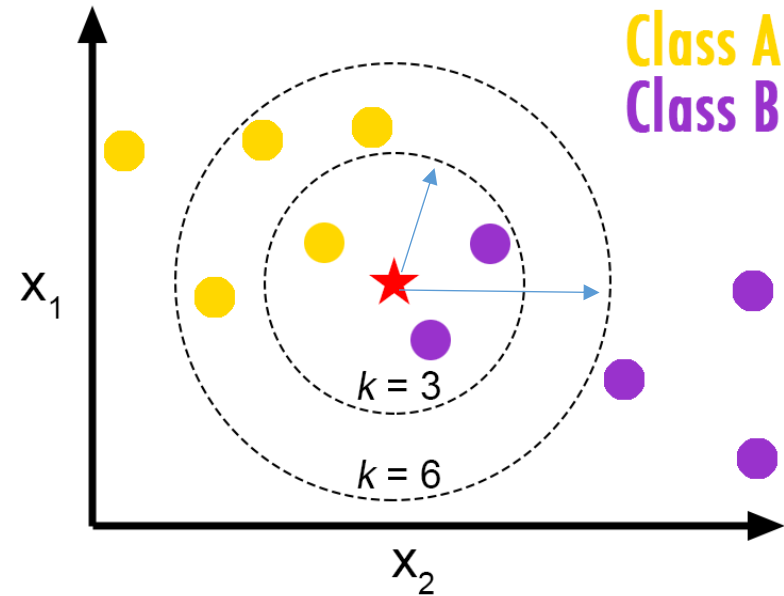
# Algoritmo K-NN

Suponga que los puntos amarillos y morados son ejemplos correctamente etiquetados, es decir, son parte de nuestro set de entrenamiento. Suponga ahora que queremos predecir qué clase de elemento es la estrella.



# Algoritmo K-NN

El algoritmo lo que hace es buscar, en una distancia a la redonda, una cantidad de vecinos  $K$  para posteriormente contabilizarlos. El algoritmo simplemente clasifica el punto como aquel que contabilizó la mayor cantidad de vecinos.



# ¿Cómo Funciona?

## **PASO 1:**

Elegir el parámetro K (número de vecinos).



## **PASO 2:**

Para un nuevo punto, tome los K vecinos más cercanos, de acuerdo a la distancia Euclidiana.



## **PASO 3:**

Dentro del grupo de K vecinos, cuente la cantidad de puntos de cada categoría.



## **PASO 4:**

Asigne al nuevo punto la categoría correspondiente a la que se contabilizó un mayor número de vecinos.



**MODELO FINALIZADO**

# Métricas de Distancia





# Distancia Euclideana

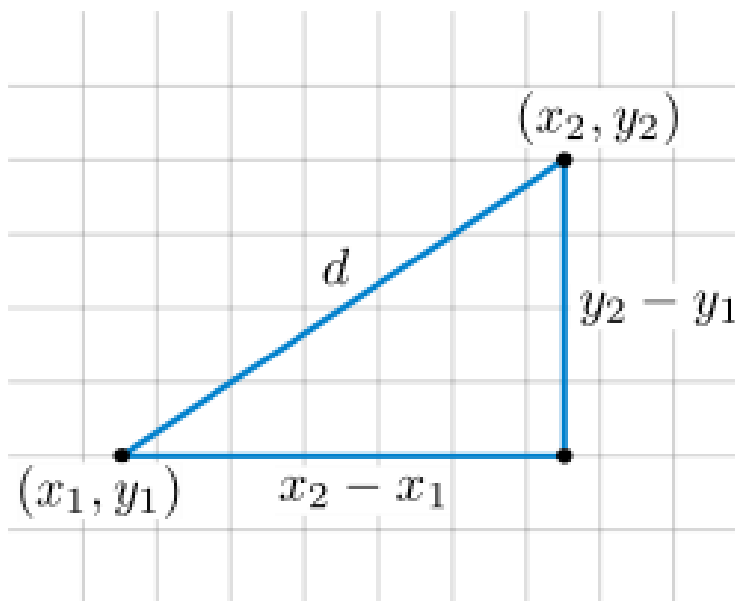
El algoritmo KNN utiliza métricas de distancia para determinar la cantidad de vecinos. La distancia Euclideana entre los puntos  $P=(p_1, p_2, \dots, p_n)$  y  $Q=(q_1, q_2, \dots, q_n)$ , del espacio euclidiano n-dimensional, se define como:

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}.$$

# Distancia Euclideana

Para un espacio con 2 dimensiones:

$$d_E(P_1, P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



# Distancia Manhattan

Conocida también como distancia de ciudad o métrica del taxista, es una métrica de distancia en la cual la distancia entre dos puntos es la suma de las diferencias (absolutas) de sus coordenadas.

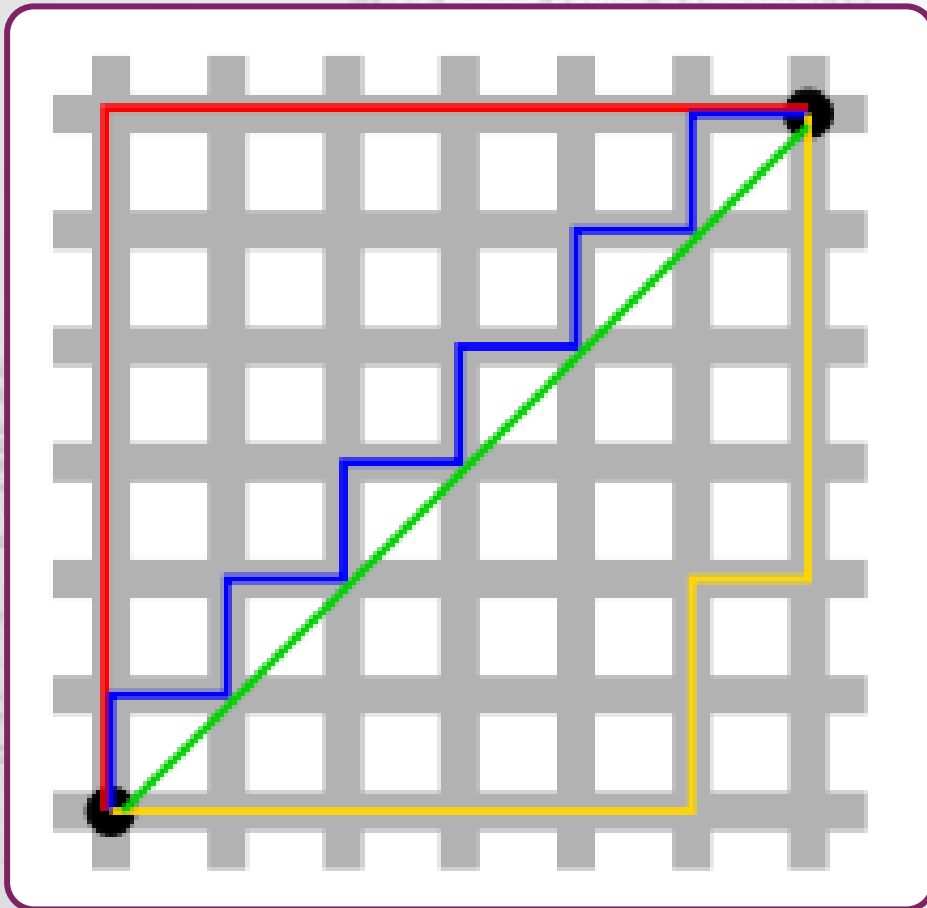
La distancia del taxista,  $d_1$ , entre dos vectores,  $p$  y  $q$ , en un espacio vectorial real  $n$ -dimensional con un sistema de coordenadas cartesianas fijo es la suma de las longitudes de las proyecciones del segmento de línea entre los puntos sobre el sistema de ejes coordenados.

$$d_1(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_1 = \sum_{i=1}^n |p_i - q_i|,$$

donde  $\mathbf{p} = (p_1, p_2, \dots, p_n)$  y  $\mathbf{q} = (q_1, q_2, \dots, q_n)$  son vectores.

# Distancia Manhattan

Por ejemplo, en el plano, la distancia del taxista entre  $(p_1, p_2)$  y  $(q_1, q_2)$  es  $|p_1 - q_1| + |p_2 - q_2|$





# Distancia Minkowski

La distancia Minkowski es una métrica en un espacio vectorial normado el cual puede ser considerado una generalización de la distancia Euclideana y la distancia Manhattan. Se le llamó así en honor al matemático alemán German Minkowski.

La distancia de orden  $p$  (donde  $p$  es un entero) entre dos puntos

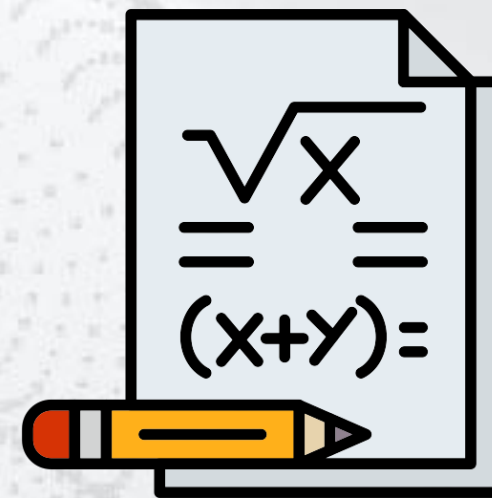
$$X = (x_1, x_2, \dots, x_n) \text{ and } Y = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$$

Se define como:

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

Cuando  $p=1$ , corresponde a la distancia Manhattan.

Cuando  $p=2$ , corresponde a la distancia Euclideana.



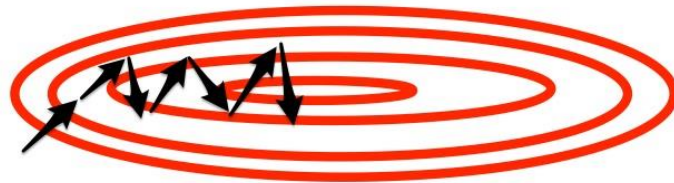
# Feature Scaling (Escalamiento de características)



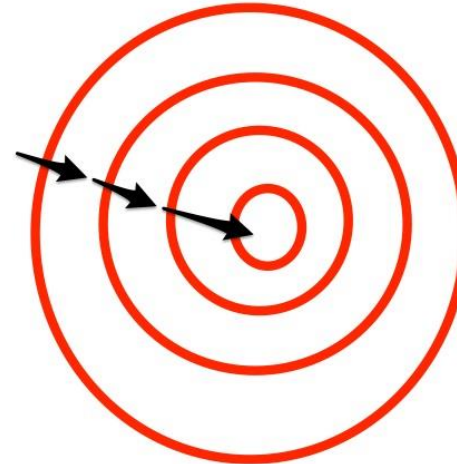
# Feature Scaling

Dado que los predictores pueden tener distintas unidades y escalas de medida, algunos algoritmos de Machine Learning podrían requerir el escalado de datos. Esto es necesario, por ejemplo, en aquellos algoritmos de clasificación que calculan la distancia entre puntos, para que no predomine el feature con valores más grandes.

Sin escalamiento de Features



Con escalamiento de Features




Otro motivo es que con los datos en una misma escala, la convergencia de la función de costo es más rápida.

# Feature Scaling

## Métodos más comunes para el escalamiento de features (características):

- Min Max Scaler.
- Standard Scaler.



Prev

Up

Next

scikit-learn 0.23.1

Other versions

Please cite us if you use the software.

API Reference

sklearn.base: Base classes and utility functions

sklearn.calibration: Probability Calibration

sklearn.cluster: Clustering

sklearn.compose: Composite Estimators

sklearn.covariance: Covariance Estimators

sklearn.cross\_decomposition: Cross decomposition

sklearn.datasets: Datasets

sklearn.decomposition: Matrix Decomposition

sklearn.preprocessing: Preprocessing and Normalization

The `sklearn.preprocessing` module includes scaling, centering, normalization, binarization methods.

User guide:

See the [Preprocessing data](#) section for further details.

<code>preprocessing.Binarizer(*[, threshold, copy])</code>	Binarize data (set feature values to 0 or 1) according to a threshold
<code>preprocessing.FunctionTransformer([func, ...])</code>	Constructs a transformer from an arbitrary callable.
<code>preprocessing.KBinsDiscretizer([n_bins, ...])</code>	Bin continuous data into intervals.
<code>preprocessing.KernelCenterer()</code>	Center a kernel matrix
<code>preprocessing.LabelBinarizer(*[, neg_label, ...])</code>	Binarize labels in a one-vs-all fashion
<code>preprocessing.LabelEncoder</code>	Encode target labels with value between 0 and <code>n_classes-1</code> .
<code>preprocessing.MultiLabelBinarizer(*[, ...])</code>	Transform between iterable of iterables and a multilabel format
<code>preprocessing.MaxAbsScaler(*[, copy])</code>	Scale each feature by its maximum absolute value.
<code>preprocessing.MinMaxScaler([feature_range, copy])</code>	Transform features by scaling each feature to a given range.
<code>preprocessing.Normalizer([norm, copy])</code>	Normalize samples individually to unit norm.
<code>preprocessing.OneHotEncoder(*[, categories, ...])</code>	Encode categorical features as a one-hot numeric array.
<code>preprocessing.OrdinalEncoder(*[, ...])</code>	Encode categorical features as an integer array.
<code>preprocessing.PolynomialFeatures([degree, ...])</code>	Generate polynomial and interaction features.
<code>preprocessing.PowerTransformer([method, ...])</code>	Apply a power transform featurewise to make data more Gaussian-like.
<code>preprocessing.QuantileTransformer(*[, ...])</code>	Transform features using quantiles information.

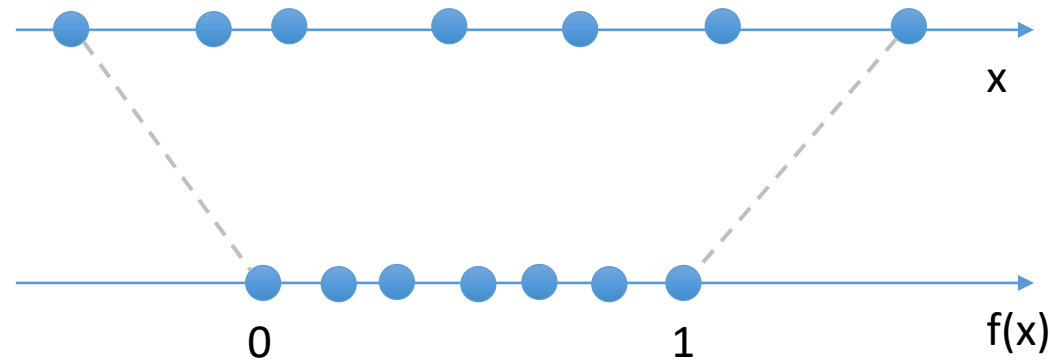
<https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing>



# Min-Max Scaler

El escalamiento MinMax transforma la escala de valores entre 0 y 1.

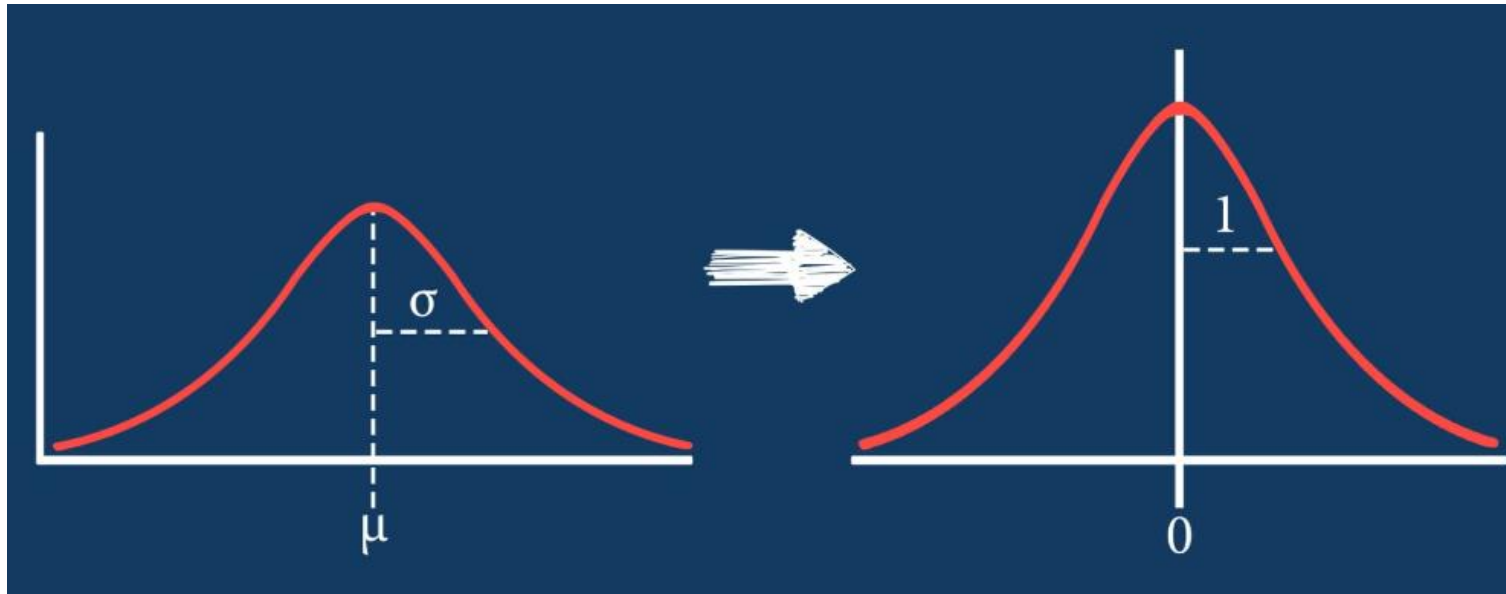
$$f(x) = \begin{cases} 0, & x < \min(x) \\ \frac{x - \min(x)}{\max(x) - \min(x)}, & \min(x) < x < \max(x) \\ 1, & x > \max(x) \end{cases}$$



# Standard Scaler (es el que yo llamo z-score)

El escalador estándar, centra los datos con media de 0 y desviación estándar de 1

$$Z = \frac{x - \mu}{s}$$



Valor de K

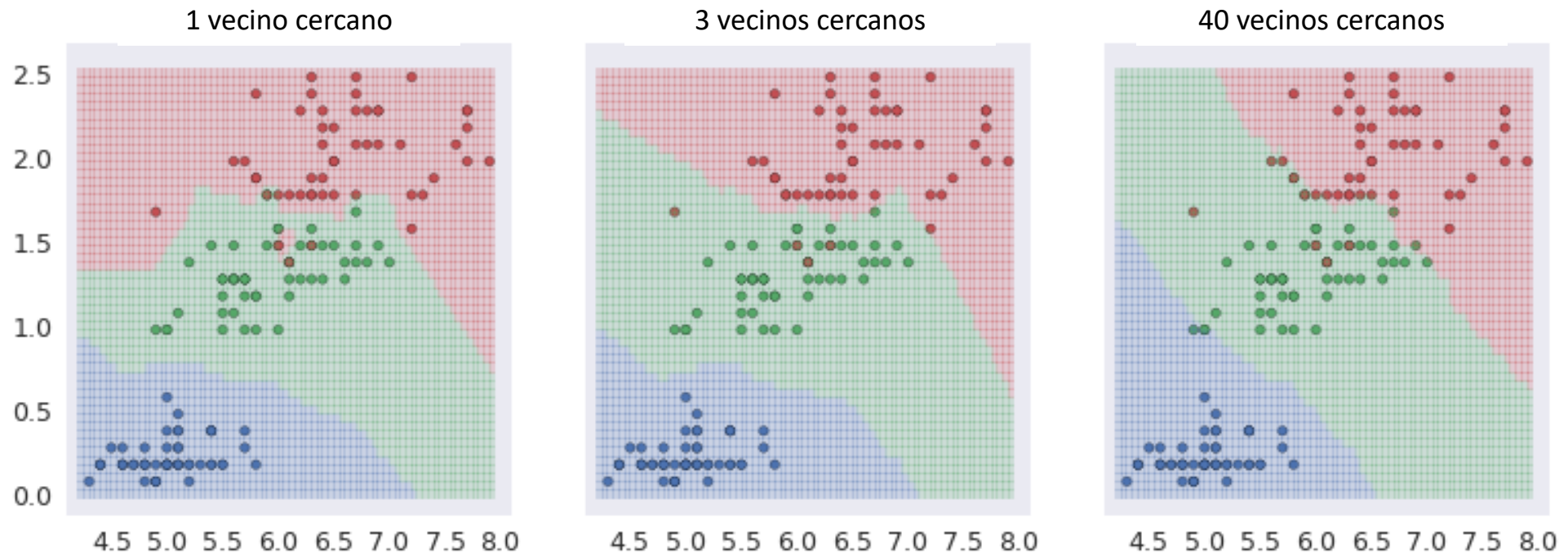


# Valor de K

Dependiendo del valor de K que se seleccione, el clasificador podría tener distinto performance en el set de datos. Nótese que en la medida que k se incrementa, los límites de decisión son más suaves.

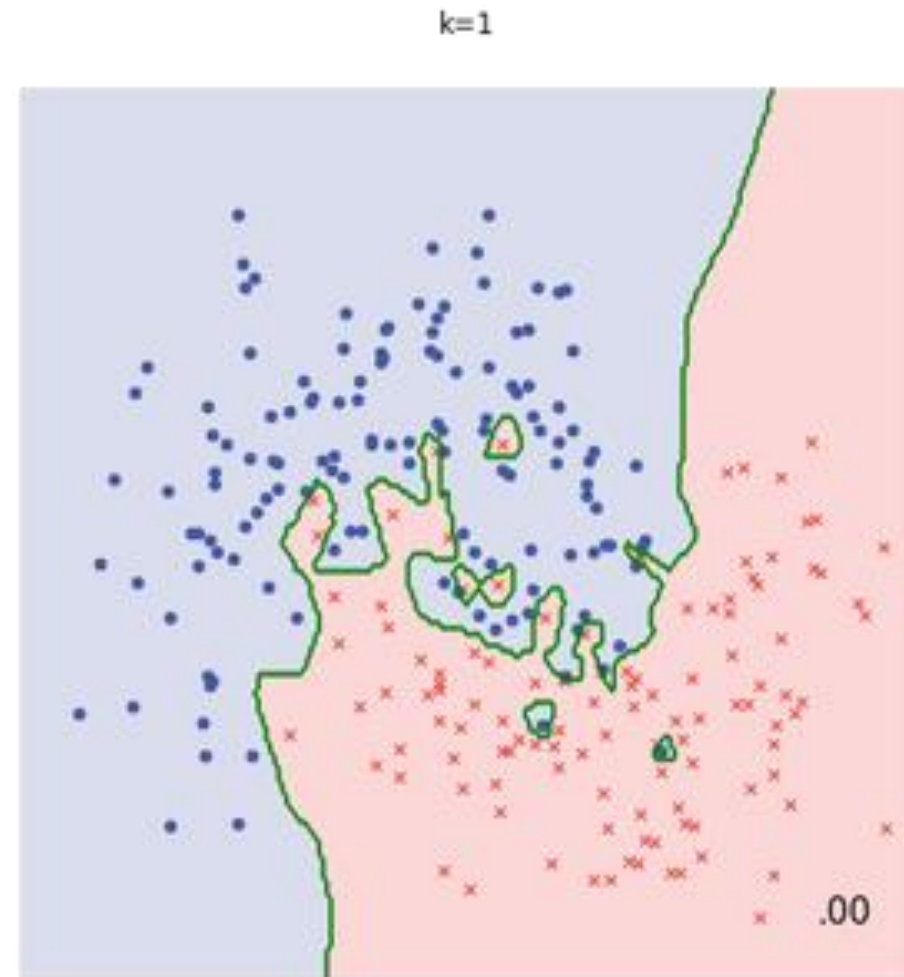
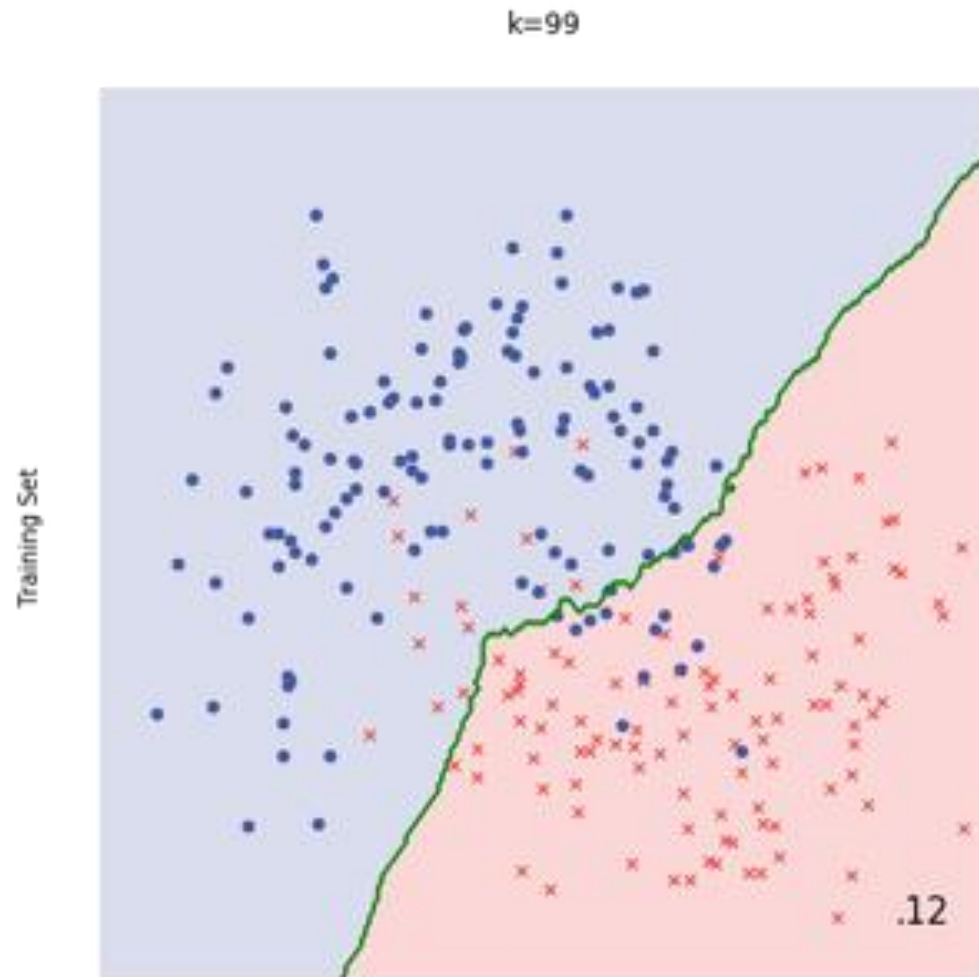
Un caso extremo sería  $k = \text{total de puntos}$ , en donde el límite de decisión sería completamente de la clase predominante.

## Clasificador K-ésimo vecino más cercano usando un número diferente de vecinos





# Valor de K



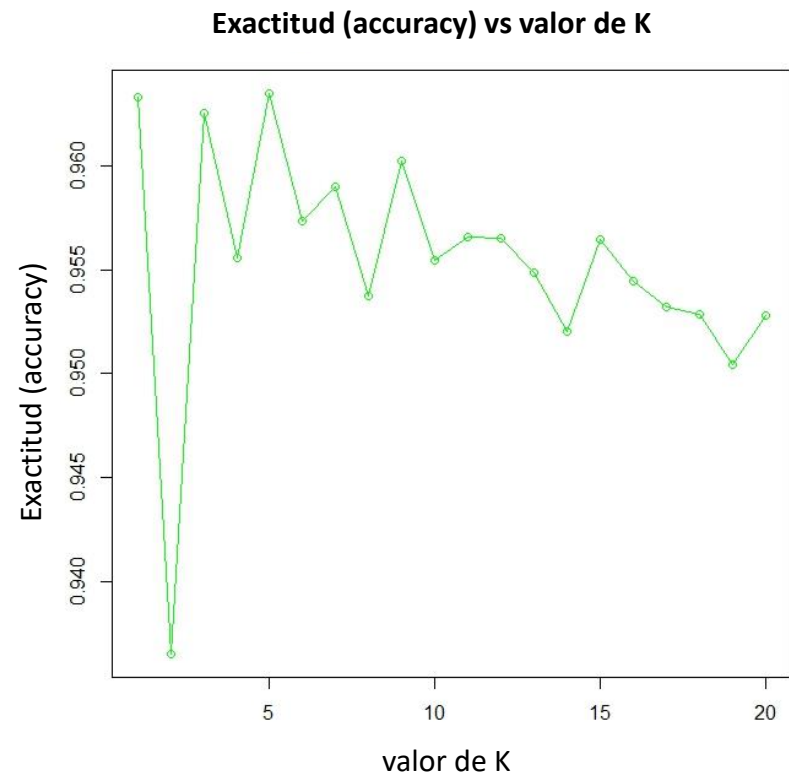
# El Valor de K

**No es sencillo determinar el valor de K, en KNN. Acá hay algunos criterios y recomendaciones:**

- No hay un método estructurado para encontrar el valor de K, hay que hacer varios intentos de “prueba y error”.
- Un valor pequeño de K significa que el ruido va a tener mayor influencia en el resultado (podría sobreajustar el modelo).
- Un valor muy grande de K haría que los límites de decisión sean más suaves, lo cual implica menor varianza pero mayor desvío. Por otro lado, es más intensivo en procesamiento también.
- Otra forma de elegir k es por medio de la validación cruzada, testeando varios valores de k y viendo cuál es el que tiene el mejor performance en el set de validación, que sería el que minimiza el error de validación.
- En general, una forma simple que se utiliza es seleccionar  $k = \sqrt{n}$ , en donde n corresponde a la cantidad de datos del set de entrenamiento.
- Utilizar un número impar si el número de clases es 2.

# Exactitud v/s K

No necesariamente incrementar el valor de K significa mejorar la exactitud de la predicción. Es importante ver el comportamiento de la exactitud para distintos valores de k.





## Ventajas

- 1. No asume una distribución de los datos:** el algoritmo KNN es un algoritmo no paramétrico, lo que significa que no hace suposiciones sobre la distribución de los datos. En cambio, utiliza los datos de entrenamiento para hacer predicciones sin imponer ninguna estructura particular en los datos.
- 2. Simplicidad:** el algoritmo KNN es fácil de entender y de implementar. Solo se necesita elegir una medida de distancia y un valor para  $k$ .
- 3. Puede ser utilizado para clasificación y regresión:** el algoritmo KNN puede ser utilizado tanto para problemas de clasificación como de regresión.

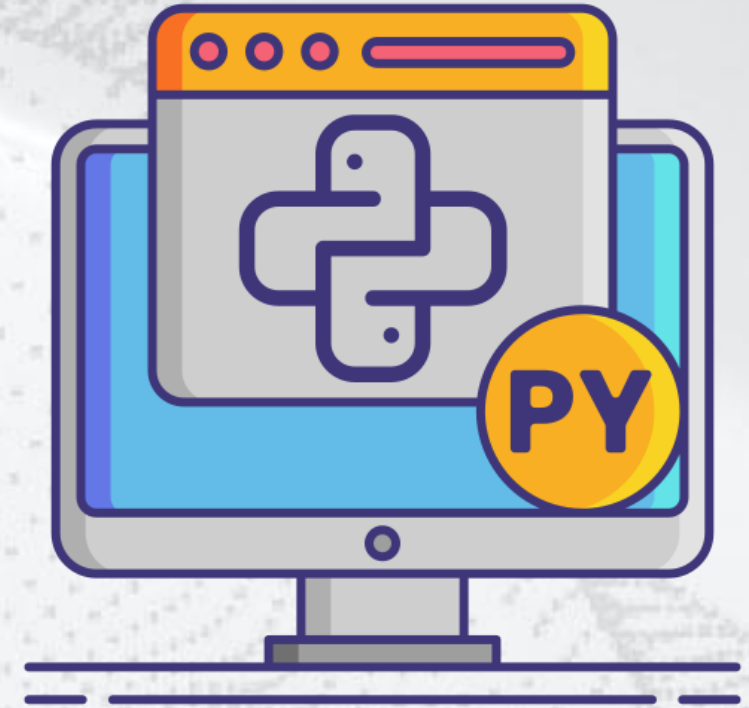


## Desventajas

- 1. Sensible a la escala de las variables:** ya que la distancia entre dos observaciones depende de la escala de las variables. Es importante escalar las variables antes de aplicar el algoritmo KNN.
- 2. Requiere almacenamiento de todo el conjunto de entrenamiento en memoria:** lo que puede ser problemático para grandes conjuntos de datos.
- 3. Elección del valor de  $k$ :** es importante en el algoritmo KNN, y a veces puede ser difícil encontrar el valor apropiado.
- 4. Puede verse afectado por la maldición de la dimensionalidad:** lo que significa que cuando el número de variables predictoras es grande, el espacio de búsqueda se vuelve muy grande y las observaciones pueden estar muy alejadas unas de otras en términos de distancia. Esto puede hacer que el algoritmo sea menos efectivo en la predicción.



# Implementación en Python



# Formulación del Modelo

Para este caso volveremos a utilizar el dataset del Titanic. Repetiremos los pasos habituales de carga, regularización del modelo y limpieza de datos. Recuerde que también debimos crear variables dummy y finalmente, seleccionar las variables que se incorporan al modelo.

## variables dummy

```
: train.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)
```

## Formulamos el modelo

```
: X = pd.get_dummies(train, drop_first=True).drop('Survived', axis=1)  
y = train['Survived']
```

```
: X.head()
```

```
:  
      Pclass  Age  SibSp  Parch   Fare  Sex_male  Embarked_Q  Embarked_S  
0         3  22.0     1     0  7.2500         1           0           1  
1         1  38.0     1     0 71.2833         0           0           0  
2         3  26.0     0     0  7.9250         0           0           1  
3         1  35.0     1     0 53.1000         0           0           1  
4         3  35.0     0     0  8.0500         1           0           1
```

# Feature Scalling

Dado que el algoritmo KNN trabaja en base a distancias Euclideanas, debemos escalar los valores de los atributos. Para esto, utilizaremos la clase StandardScaler perteneciente al package sklearn.preprocessing.

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
scaler.fit(X)
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

```
scaled_X = scaler.transform(X)
```

```
scaled_X[:3,:]
```

```
array([[ 0.82737724, -0.53383369,  0.43279337, -0.47367361, -0.50244517,  
        0.73769513, -0.30756234,  0.61930636],  
       [-1.56610693,  0.67489052,  0.43279337, -0.47367361,  0.78684529,  
       -1.35557354, -0.30756234, -1.61470971],  
       [ 0.82737724, -0.23165264, -0.4745452 , -0.47367361, -0.48885426,  
       -1.35557354, -0.30756234,  0.61930636]])
```

Nótese que StandardScaler nos retornó un array

# Feature Scalling

Ahora creamos un DataFrame a partir del arreglo de los atributos escalados.

```
X_sc = pd.DataFrame(scaled_X, columns=X.columns)
```

```
X_sc.head()
```

	Pclass	Age	SibSp	Parch	Fare	Sex_male	Embarked_Q	Embarked_S
0	0.827377	-0.533834	0.432793	-0.473674	-0.502445	0.737695	-0.307562	0.619306
1	-1.566107	0.674891	0.432793	-0.473674	0.786845	-1.355574	-0.307562	-1.614710
2	0.827377	-0.231653	-0.474545	-0.473674	-0.488854	-1.355574	-0.307562	0.619306
3	-1.566107	0.448255	0.432793	-0.473674	0.420730	-1.355574	-0.307562	0.619306
4	0.827377	0.448255	-0.474545	-0.473674	-0.486337	0.737695	-0.307562	0.619306



# Cross Validation (validación cruzada)

Como es habitual, dividimos nuestro dataset en test y train set.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_sc, y, test_size=0.33, random_state=101)
```

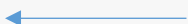
# Entrenamiento Algoritmo

Ahora entrenamos el algoritmo KNN. Para esto, creamos el objeto `KNeighborsClassifier`, ajustamos el modelo y realizamos predicciones sobre el set de test.

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X_sc, y, test_size=0.33, random_state=101)
```

```
knn = KNeighborsClassifier()
```

 Por defecto, utiliza k=5

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

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',  
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,  
                     weights='uniform')
```

```
predictions = knn.predict(X_test)
```

```
predictions
```

```
array([0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,  
       1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,  
       0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0,  
       1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,  
       1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1,  
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0,  
       0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0,  
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1,  
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,  
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,  
       1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0,  
       0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1,  
       1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0])
```

# Métricas de performance

Calculamos las métricas obtenidas del modelo entrenado anteriormente.

```
from sklearn.metrics import classification_report
```

```
print(classification_report(predictions, y_test))
```

	precision	recall	f1-score	support
0	0.85	0.79	0.82	180
1	0.71	0.77	0.74	115
avg / total	0.79	0.79	0.79	295

```
from sklearn.metrics import confusion_matrix
```

```
confusion_matrix(predictions, y_test)
```

```
array([[143, 37],  
       [ 26, 89]], dtype=int64)
```

# Determinando el valor de K

Para determinar el valor de K (hiperparámetro), realizaremos el entrenamiento con valores entre 1 y 50. Posteriormente sacaremos una medida del error y graficaremos los resultados.

```
error_rate = []  
  
for i in range(1,50):  
    knn = KNeighborsClassifier(n_neighbors=i)  
    knn.fit(X_train, y_train)  
    pred_i = knn.predict(X_test)  
    error_rate.append(np.mean(pred_i != y_test))
```

```
error_rate[:5]
```

```
[0.21694915254237288,  
 0.21694915254237288,  
 0.19322033898305085,  
 0.21694915254237288,  
 0.2135593220338983]
```

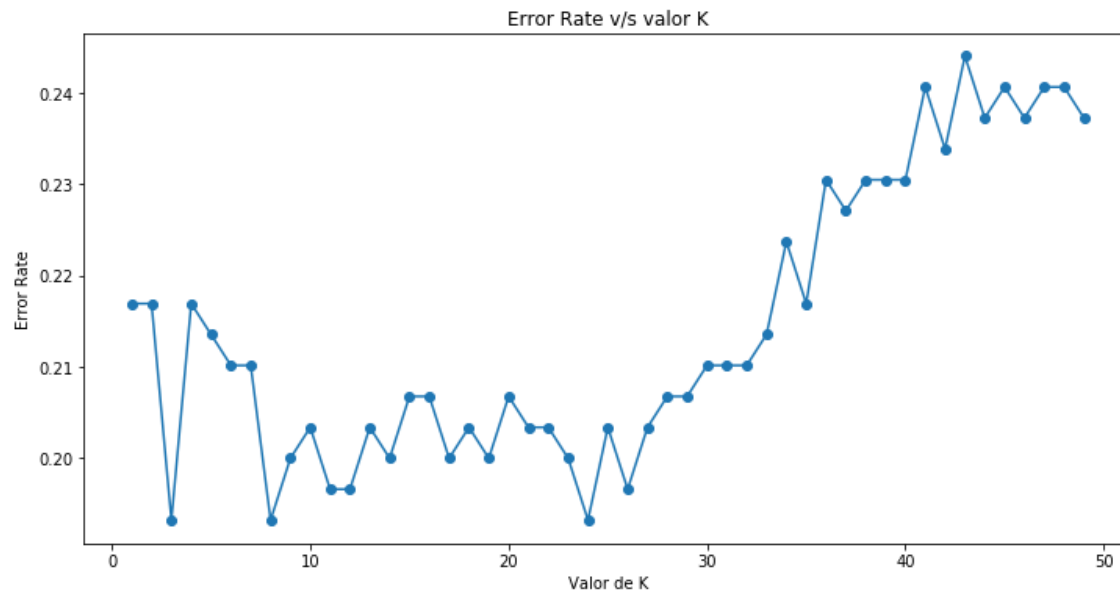


# Determinando el valor de K

Como podemos observar en el gráfico, los menores valores de error se obtuvieron con  $k=3$ ,  $k=8$  y  $k=24$ . Después, a medida que aumenta el valor de K, el error comienza a crecer.

```
plt.figure(figsize=(12,6))
plt.plot(range(1,50), error_rate, marker='o')
plt.title("Error Rate v/s valor K")
plt.xlabel("Valor de K")
plt.ylabel("Error Rate")
```

```
Text(0,0.5,'Error Rate')
```



# Modelo Final

Para formular el modelo final, entrenaremos el algoritmo con k=8 y calcularemos las métricas. Como se puede observar, el modelo mejoró su performance respecto al modelo inicial con k=5 por defecto.

```
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train, y_train)
predictions = knn.predict(X_test)
print(confusion_matrix(predictions, y_test))
print('\n')
print(classification_report(predictions, y_test))
```

```
[[156  44]
 [ 13  82]]
```

	precision	recall	f1-score	support
0	0.92	0.78	0.85	200
1	0.65	0.86	0.74	95
avg / total	0.84	0.81	0.81	295

The background of the slide features a light gray, wavy, dotted pattern that resembles a stylized landscape or a textured surface. A solid purple horizontal banner is positioned across the middle of the slide, containing the text "Dudas y consultas" in white.

# Dudas y consultas

Fin Presentación