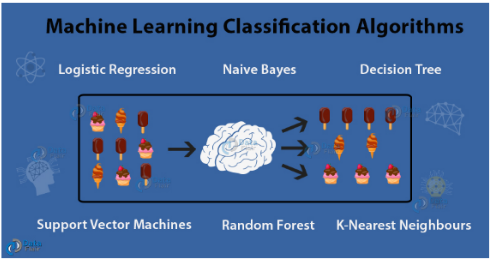
**Introducción a Clasificación KNN**

En las técnicas de aprendizaje automático, tenemos el **aprendizaje supervisado** (con una **variable objetivo a predecir;** necesitan ***input + output asociado***) y el **aprendizaje no supervisado (sin variable objetivo**;necesitan sólo ***input*)**.

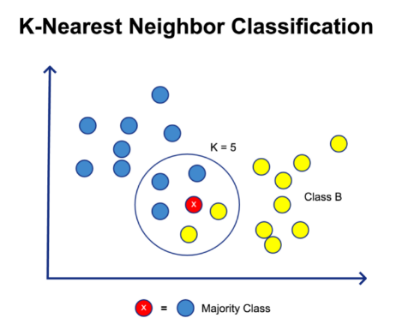
A su vez, dentro de aprendizaje supervisado hay dos tipos:

* **Regresiones** (aplicables cuando la **variable objetivo** es **continua**)
* **Clasificaciones** (aplicables cuando la **variable objetivo** es **categórica**). IE: Predicción de bajas de clientes, distinción de comentarios positivos o negativos en redes sociales; spam en correos electrónicos, diagnóstico de enfermedades, detección de fraudes en tarjetas de crédito.

Hay **diferentes** **modelos** con **diferentes soluciones** a **Problemas de Clasificación:**



**KNN – K Nearest Neighbours** *(Dime con quién andas y te dire quién eres).*



1. **Memoriza** la **ubicación** de cada miembro del *Train Dataset*, según los valores de sus *features*.
2. **Ubica** los **datos nuevos** en el espacio.
3. **Encuentra** los **k vecinos más cercanos** (siendo **k** un **hiperparámetro del modelo**)
4. Con cada uno de los **k vecinos más cercanos** recibe un “voto” de la clase a la que pertenece.
5. La **clase mayoritaria** entre los votos del caso 4 es la **predicción resultante**.

Entonces, lo que hace este modelo es crear un **espacio** **N dimensional**, donde cada dimensión corresponde a una de las n *features*, Se lo llama **algoritmo** **lazy**, ya que no aprende una función discriminante en la etapa de entrenamiento (no genera un modelo). Como usa **todas las variables al momento de predecir**, es **costoso en términos computacionales**. **Este costo aumenta** al **aumentar** el valor de **K**. Funciona con **Features continuas**, pero puede modificarse para funcionar con **features categóricas.** Pertenece a la clase de modelos llamada **Modelos IBL (Instance Based Learning)**.

Ejemplo: Usamos un modelo KNN para determinar si estamos frente a células benignas o cancerígenas.

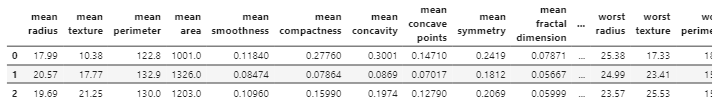
**En Python:**

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

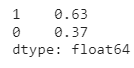
X = pd.DataFrame(cancer.data, columns = cancer.feature\_names)

X.head()



Y = pd.Series(cancer.target)

Pd.Series(y).value\_counts(normalize=True).round(2)



IE **En Python**:

**Creamos el modelo KNN:** from sklearn.neighbors import KNeighborsClassifier

Usamos el **parámetro n\_neighbors** para indicar la cantidad de vecinos más cercanos a considerar. Por ejemplo, 5: knn = KNeighborsClassifier(n\_neighbors=5)

Generamos los **datasets de train y test**: from sklearn.model\_selection import train\_test\_split

X\_train, y\_train, X\_test, y\_test = train\_test\_split(X, y, random\_state=2)

**Ajustamos** a los **datos de entrenamiento**: knn.fit(X\_train, y\_train)

**Predecimos** etiquetas para los **datos de test**: y\_pred = knn.predict(X\_test)

Calculamos el **accuracy del modelo**: from sklearn.metrics import accuracy\_score

Accuracy\_score(y\_test, y\_pred).round(2)



Como resultado nos dio que el modelo pudo clasificar correctamente el 92% de las observaciones del dataset *test*.

**Evaluación del Modelo**

¿Cómo sabemos cuál es el valor óptimo de K para la cantidad de vecinos más cercanos? Vamos a generar un modelo con una secuencia de valores desde 1 a 25y evaluaremos para cada uno de ellos el accuracy correspondiente. En este caso, como con la primer instancia el modelo pasa a conocer los datos de test, no podemos evaluar accuracy sobre los mismos; porque dejan de ser datos nuevos para el modelo. Lo que se hace entonces es reservar un conjunto de validación dentro de los datos de entrenamiento:

**En Python:**

X\_train\_train, X\_validation, y\_train\_train, y\_validation = train\_test\_split(X\_train, y\_train, random\_state = 2)

k\_range = list(range(1, 26))

scores = []

for k in k\_range:

knn = KNeighborsClassifier(n\_neighbors = k)

knn.fit(X\_train\_train, y\_train\_train)

y\_pred = knn.predict(X\_validation)

scpres.append(accuracy\_score(y\_validation, y\_pred))

plt.figure(figsize = (4,3))

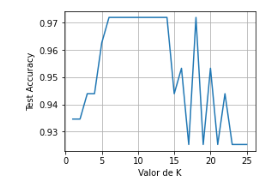
plt.plot(k\_range, scores)

plt.xlabels(‘valor de k’)

plt.ylabels(‘Test Accuracy’)

plt.grid()

plt.show()



Obtenemos los mejores valores de test para valores de k entre 6 y 14 y para valor de k igual 18.

l = [i for i, val in enumerate(scores) if val == max(scores)]

print(‘Mínimo K de todos los que tienen máximo score: ‘, min(l)+1)

print(‘Máximo K de todos los que tienen máximo score: ‘, max(l)+1)



Una forma más precisa de evaluar el modelo es con **Cross-Validation**:



Vamos a volver a generar una secuencia de valores para el hiperparámetro k entre 1 y 25. Y vamos a probar cada uno de estos valores tantas veces como *folds*le indiquemos al cross-validation:

**En Python**

from sklearn.model\_selection import cross\_val\_score, KFold

kf = KFold(n\_split = 5, shuffle = True, random\_state = 12) #*usa cross-validation con 5 folds*

scores\_para\_df = []

for I in range(1, 26):

model = KNeighborsCassifier(n\_neighbors=1i) #*inscancia el modelo en cada iteración con un hiperparámetro distinto.*

cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv = kf) #*cross\_val\_scores nos devuelve un array de 5 resultado, uno para cada partición que hizo CV.*

dict\_row\_score = {‘score\_medio’: np.mean(cv\_scores), ‘n\_neighbors’:i} #*creamos un diccionario con el valor de n\_neighbors y la media de los 5 scores recibidos para cada valor de n\_neighbors*

scores\_para\_df.append(dict\_row\_score) #*creamos una lista de diccionarios*

df\_scores = pd.DataFrame(scores\_para\_df) #*creamos un DataFrame a partir de la lista de diccionarios*

df\_scores.loc[df\_scores.score\_medio == df\_scores.score\_medio.max()] # *Identificamos el k que maximiza el score promedio*



Nos da que el K con mayor score promedio es 8, con un score promedio de 93,9%.

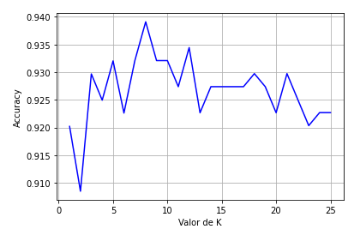
plt.plot(df\_scores[‘n\_neighbors’], df\_scores[‘score\_medio’], color=’b’)

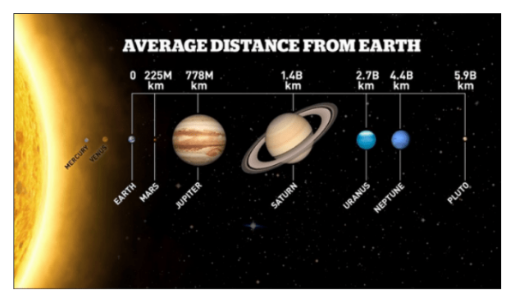
plt.xlabel(‘Valor de K’)

plt.ylabel(‘Accuracy’)

plt.grid()

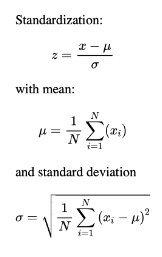
plt.show()



**Estándarización**: 

Necesitamos una **distancia** para poder medir la **cercanía entre las observaciones**. Cada *feature* del Dataset representa una dimensión; un eje, en el *espacio multidimensional* donde se encuentran las observaciones. No todas las *features* se encuentran en la misma escala. En caso de trabajar con hasta 3 dimensiones, son graficables, pero si las unidades de medida son distintas, entonces se generarán **distorsiones** al calcular las distancias entre puntos. Para evitar esto, hay que **estandarizarlas variables**, **eliminando sus distintas unidades de medida** y **evitando así distorsiones** debidas a diversas escalas.

**En Python:** Método **StandardScaler()**. Es un método que normaliza/estandariza cada columna de la matriz X **en forma individual**. Con media µ = 0 y std σ = 1.



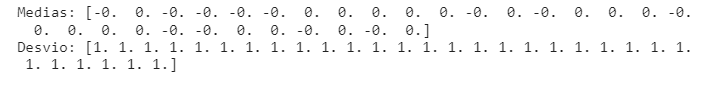
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

Print(‘Medias: ‘, np.mean(X\_train, axis = 0).round(2))

Print(‘Desvío: ‘, np.std(X\_train, axis = 0).round(2))



# Corroboramos que las medias son 0 y los std son 1, tal como indica la definición del método StandardScaler que debe hacer.

Ahora vamos a generar un nuevo modelo KNN con los datos normalizados:

from sklearn.model\_selection import cross\_val\_score, KFold

kf = KFold(n\_split=5, shuffle=True, random\_state = 12) # usamos cross-validation con 5 folds

scores\_para\_df = []

for i in range(1,26):

model = KNeighborsClassifier(n\_neighbors=i)

cv\_scores = cross\_val\_score(model, X\_train, y\_train, cv = kf)

dict\_row\_score = {‘score\_medio’:np.mean(cv\_scores), ‘n\_neighbors’:i}

scores\_para\_df.append(dict\_row\_score)

df\_scores = pd.DataFrame(scores\_para\_df)

df\_scores.loc[df\_scores.score\_medio == df\_scores.score\_medio.max()]



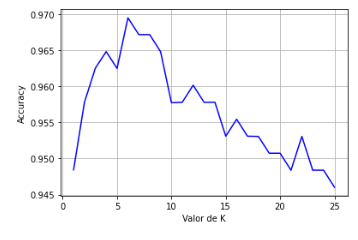
plt.plot(df\_scores[‘n\_neighbors’], df\_scores[‘score\_medio’], color = ‘b’)

plt.xlabel(‘Valor de K’)

plt.ylabel(‘Accuracy’)

plt.grid()

plt.show()

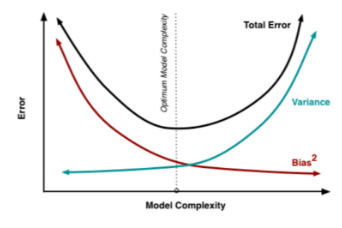


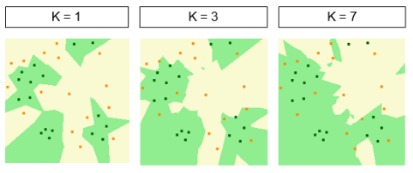
El valor K que da el mayor score promedio pasó a ser 6, y el score promedio mejoró a 0.97.

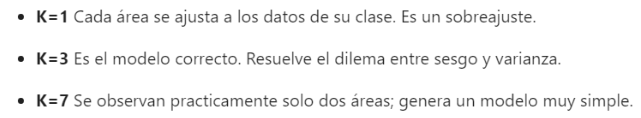
**Ajuste del Modelo**:

La capacidad de generalización de un modelo depende de qué tan bien se ajuste a los datos de entrenamiento:

* Con **K** **muy bajo**, va a dar un modelo muy complejo, con **alta varianza** y **bajo sesgo** (**overfit**).
* Con **K** **muy alto**, va a dar un modelo muy simple, con **baja varianza** y **alto sesgo** (**underfit**).

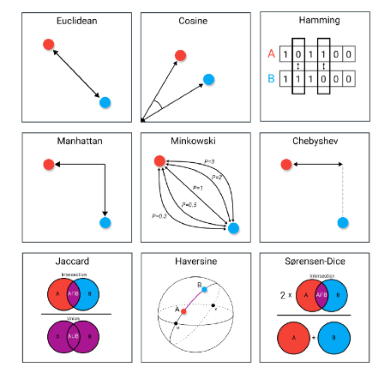






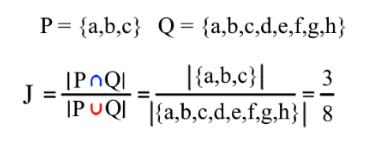
**Distancia:**

Hay muchas medidas de distancias:



**Métricas de Similaridad**: Forma de decir lo “parecidos” que son los datos entre sí. Se expresan usualmente con valores numéricos entre 0 y 1. 0 es baja similaridad. 1 es alta similaridad. Se pueden aplicar sobre datos numéricos, o alfabéticos, entre otros.

Un ejemplo de similaridad es el *Coeficiente de Jaccard*:



**En Python:**

def jaccard\_similarity(list1, list2):

s1 = set(list1)

s2 = set(list2)

return float(len(s1.intersection(s2))/ len(s1.union(s2)))

list1 = [‘dog’, ‘cat’, ‘cat’, ‘rat’]

list2 = [‘dog’, ‘cat’, ‘mouse’]

jaccard\_similarity(list1, list2)



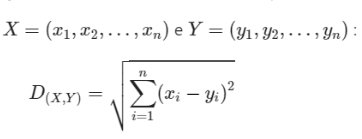
**Para que una métrica d sea una distancia**, debe:

1. Ser **no negativa**, d(p,q) >= 0 para todas las observaciones diferentes entre p y q.
2. **Simetría**. d(p,q) = d(q,p) para todo p,q.
3. **Desigualdad Triangular**: d(p,q) <= d(p,r) + d(r,q) para todo p, q, r.
4. **Igualdad**: d(p,q) = 0 sólo si p = q.

**Distancia Euclidea:**



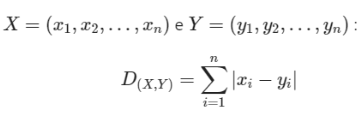
Es la distancia más común e intuitiva. Una forma de interpretarla es considerar uno de los puntos como el **centro de un círculo y** el otro como su **radio.** Trabaja bien con datos de **baja dimensionalidad**. Es la distancia que KNN usa por **default**.



**Distancia Manhattan:**

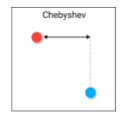


En lugar de un círculo, tenemos un cuadrado. **No se permiten movimientos diagonales para calcular la distancia**. Trabaja bien con datos de **alta dimensionalidad**. Siempre es mayor que la distancia Euclídea.

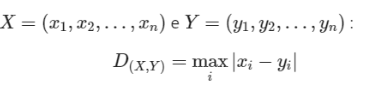


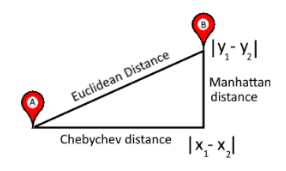
La distancia entre 2 puntos es la suma de las diferencias entre sus coordenadas.

**Distancia de Chebyshev:**



Máxima distancia en cualquiera de sus ejes. Se usa **solo en casos específicos**, donde estamos seguros de que dará buenos resultados.

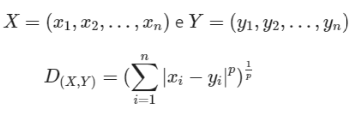




**Distancia de Minkowski:**



**Métrica en un espacio vectorial normalizado**. Es una generalización de las distancias anteriores:



Si p = 1 => **Distancia Manhattan**. Si p = 2 => **Distancia Euclideana**; si p = ∞. **Distancia Chebiyshev**.

**En Python:**

Vamos a aplicar diferentes medidas de distancia en el dataset de diagnósticos de cancer:

from sklearn.datasets import load\_breast\_cancer

cancer = load\_breast\_cancer()

X = pd.DataFrame(cancer.data, columns = cancer.feature\_names)

y = pd.Series(cancer.target)

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

# Vamos a usar la **distancia Euclídea** y evaluar el modelo:

knn =KNeighborsClassifier(metric=’euclidean’, n\_neighbors=5)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=2)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy\_score(y\_test, y\_pred).round(2)



# Vamos a usar la **distancia Manhattan** y evaluar el modelo:

knn =KNeighborsClassifier(metric=’manhattan’, n\_neighbors=5)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=2)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy\_score(y\_test, y\_pred).round(2)



# Vamos a usar la **distancia Chebyshev** y evaluar el modelo:

knn =KNeighborsClassifier(metric=’chebyshev’, n\_neighbors=5)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=2)

knn.fit(X\_train, y\_train)

y\_pred = knn.predict(X\_test)

accuracy\_score(y\_test, y\_pred).round(2)



**Conclusiones:**

El modelo **KNN** (K- Nearest Neighbors)es uno de los **algoritmos más sencillos** para **clasificar**.

1. **Permite Clasificar** a partir de la **cercanía entre observaciones** del **set de entrenamiento.**
2. **No genera un modelo** (***modelo Lazy***). Realiza la predicción usando **todas las observaciones**.
3. Hiperparámetro **K**. **Número de vecinos a considerar.**
4. Para **validar el modelo**, hay que **variar K y medir su performance**. Para esto conviene usar **cross-validation**.
5. La cercanía entre los datos se determina con las **medidas de distancia**. Existen distintos tipos de distancia, tales como la Euclídea, la Manhattan o la Chebyshev.