

Aprendizaje Supervisado

Parte I: Clasificación

Clase 05

Breve Introducción a Machine Learning

Dr. Ramón Caraballo
SECIU Red Académica Uruguay
UDELAR



Objetivo

Reconocer:

- **Tipos de Aprendizaje supervisado**
- **Algoritmos específicos para Clasificación**
- **Underfitting vs Overfitting y Bias Vs Variance Tradeoff**
- **Evaluación de la performance del modelo: Métricas**

Aprendizaje Supervisado

Uso de datos etiquetados para el desarrollo del modelo de ML.

predictores/features
vars. independientes

respuesta/target
variable (dependiente)

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

species
setosa
setosa
setosa
setosa
setosa

Principales Usos

- **Automatizar tareas manuales que consumen mucho tiempo o son costosas**
Ej: Diagnósticos médicos
- **Predicciones sobre evolución futura**
Ej: ¿Un cliente hará clic en un anuncio o no?
- **Fuentes de datos etiquetados**
 - **Datos históricos con etiquetas**
 - **Experimentos para obtener datos etiquetados**
 - **Datos etiquetados de colaboración colectiva (crowd-sourced data)**

scikit-learn

Machine Learning in Python

[Getting Started](#)[Release Highlights for 1.2](#)[GitHub](#)

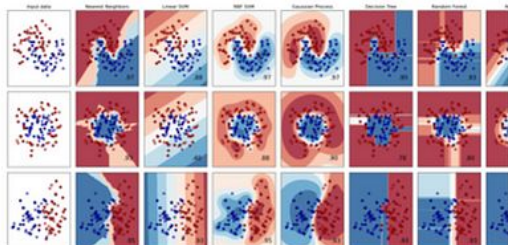
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.

Algorithms: SVM, nearest neighbors, random forest, and more...



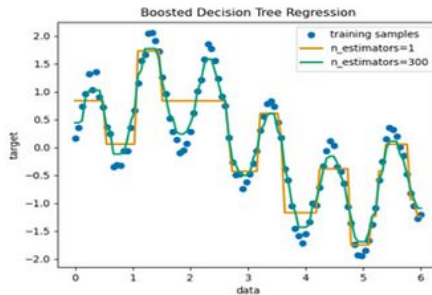
Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, nearest neighbors, random forest, and more...



Examples

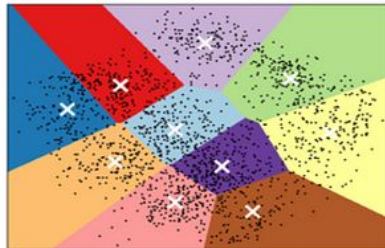
Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, and more...

K-means clustering on the digits dataset (PCA-reduced data)
Centroids are marked with white cross



Examples

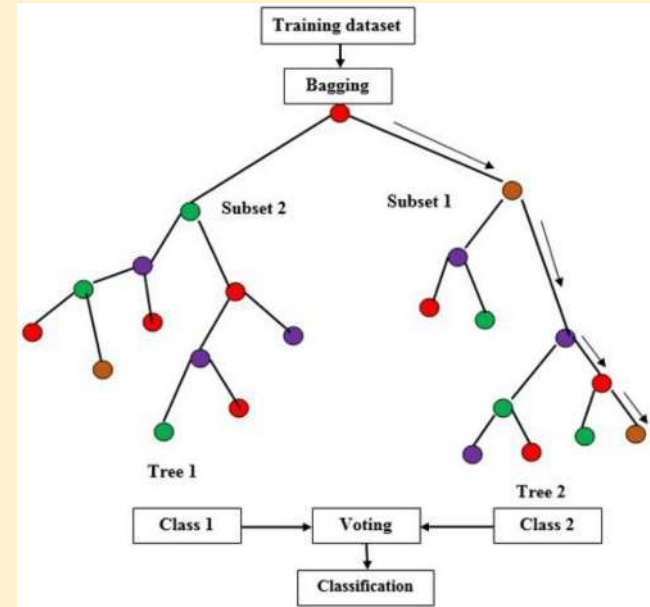
Clasificación

Categorización de un conjunto de ítems desconocidos en un conjunto discreto de clases o categorías.

La variable dependiente (target attribute) es una variable categórica.

Metodologías más comunes:

- **Regresión Logística**
- **Linear Discriminant Analysis (LDA)**
- **K-Nearest Neighbors (KNN)**
- **Decision Trees**
- **Random Forests**
- **Support Vector Machines (SVM)**
- **Naive Bayes**



Types of Data

Categorical or Qualitative

Nominal

- ✓ Letters
- ✓ Hair
- Colours
- ✓ Symbols
- ✓ Words
- ✓ Gender

Ordinal

Opinion

Agree, mostly agree,
neutral, mostly
disagree, disagree

Tumour Grade

1, 2, 3

Time of day

Morning, Noon, Night

Numerical or Quantitative

Discrete/Interval

Dates

200; 1000; 1500

Temperature

30°C; 45°C; 60°C

pH

1.2; 4.5; 7.2

IQ

80; 120; 140

Continuous/Ratio

Temperature range
Distance travelled
Time interval
Age range

Variables Categóricas

Definición

Representan datos que pueden ser clasificados en múltiples categorías pero que no pueden ordenarse o medirse. Cada categoría se puede representar por una etiqueta y los puntos son clasificados en función de propiedades cualitativas. Estas variables pueden ser subdivididas en variables ordinales, binarias o nominales

Ejemplos

Color de Pelo (Nominal)

E.g., Rubio, marron, negro, pelirrojo

Es estudiante (Binario):

Puede que sí o no.

Ranking (Ordinal):

Posiciones en una lista como ser Primero, Segundo, etc
representan variables ordinales

Categorical Variable Encoding Techniques

Los modelos de ML requieren variables numéricas

Find and replace

Replaces each matching occurrence of an old character in a string with a new character.

Label encoding

Assigns each label with a unique integer based on alphabetical ordering.

One-hot encoding

Converts each categorical variable into a column and assigns it a 1 or 0 value.

Tipos de Codificación

One-Hot-Encoding

Gender	Male	Female
Male	1	0
Female	0	1
Male	1	0
Male	1	0
Female	0	1
Male	1	0
Female	0	1
Female	0	1

Ordinal Encoding

original dataset			dataset with encoded labels		
x_1	x_2	y	x_1	x_2	y
5	8	calabar	5	8	0
9	3	uyo	9	3	2
8	6	owerri	8	6	1
0	5	uyo	0	5	2
2	3	calabar	2	3	0
0	8	calabar	0	8	0
1	8	owerri	1	8	1

LabelEncoder

```
{  
  "calabar" ----> 0  
  "owerri" ----> 1  
  "uyo" ----> 2  
}
```

Label, Ordinal & One-Hot-Encoding

Cuando usar cada uno?

Depende del dataset, el modelo usado y el número de variables.

Detalles a tener en cuenta:

Aplicamos One-Hot-Encoding cuando:

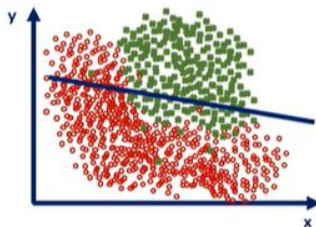
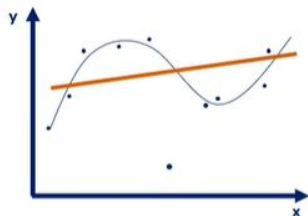
- La variable no es ordinal (p.ej. colores, tipos de ítems, etc)
- El número de variables es reducido de forma que no conlleva gran consumo de memoria

Aplicamos Label u Ordinal Encoding cuando:

- La variable es ordinal (i.e., tiene un orden inherente, p.ej: posiciones en un ranking, tamaños, etc)
- El número de categorías es muy grande y One-Hot-Encoding no es viable desde el punto de vista de eficiencia computacional y porque ignora el orden entre las categorías.

Overfitting vs Underfitting

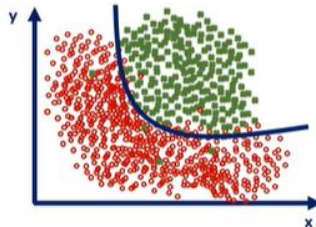
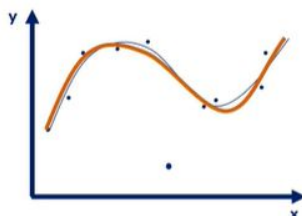
An **underfitted** model



Doesn't capture any logic

- High loss
- Low accuracy

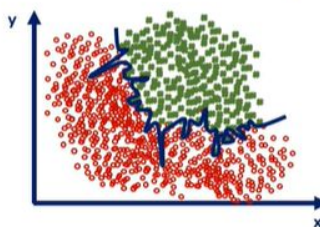
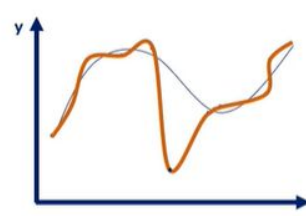
A **good** model



Captures the underlying logic of the dataset

- Low loss
- High accuracy

An **overfitted** model



Captures all the noise, thus "missed the point"

- Low loss
- Low accuracy

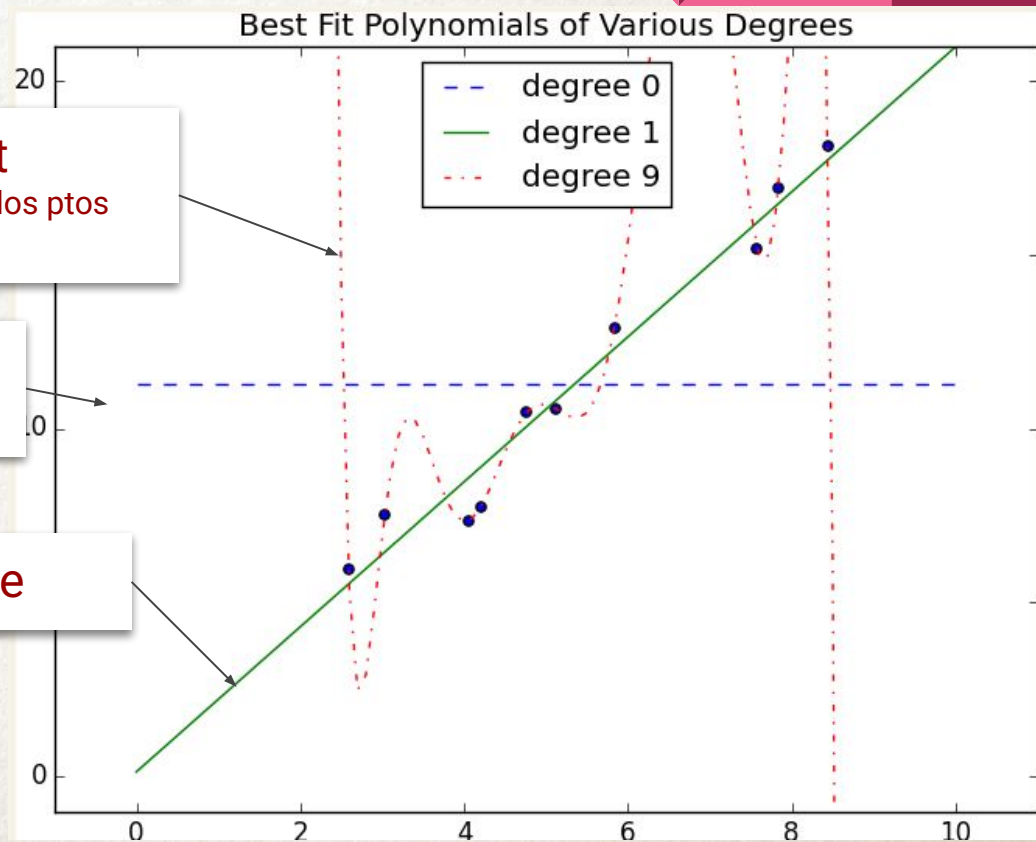
Bias-variance tradeoff: The balance between underfitting and overfitting

Overfitting vs Underfitting

polinomio gdo 9 \Rightarrow **overfit**
modela el ruido inclusive, ajusta todos los pto
pero extrapola mal con datos nuevos

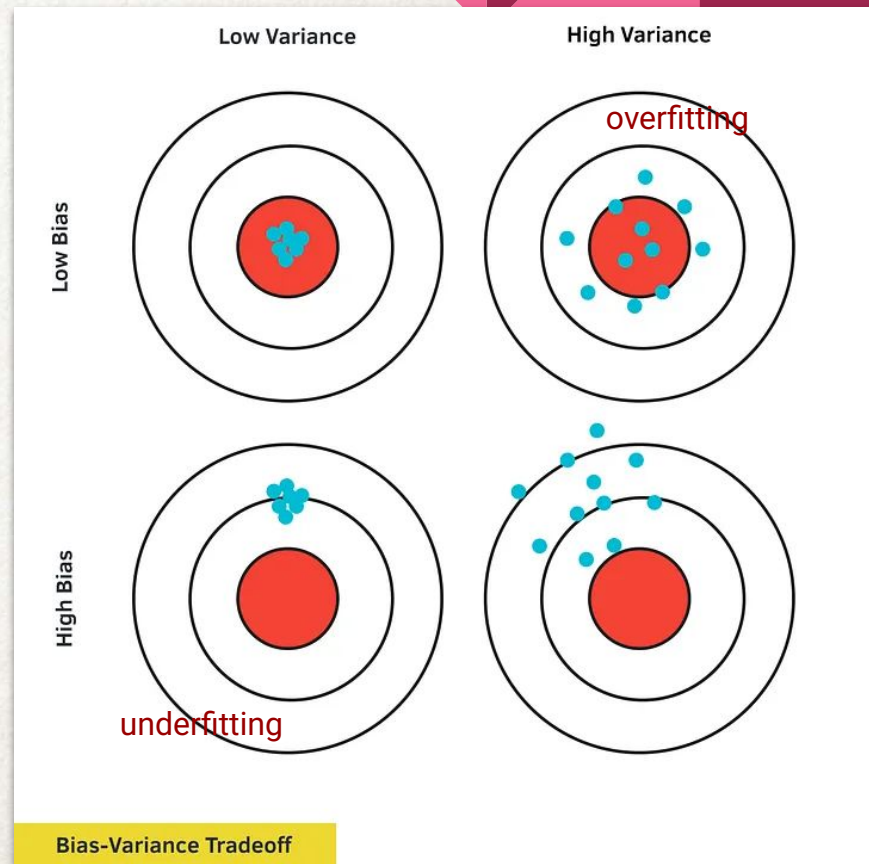
polinomio gdo 0 = cte \Rightarrow **underfit**
demasiado simple, no ajusta los datos de prueba

polinomio gdo 1 \Rightarrow **razonable**



Ajuste vs Bias-Variance

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> • High training error • Training error close to test error • High bias 	<ul style="list-style-type: none"> • Training error slightly lower than test error 	<ul style="list-style-type: none"> • Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> • Complexify model • Add more features • Train longer 		<ul style="list-style-type: none"> • Perform regularization • Get more data

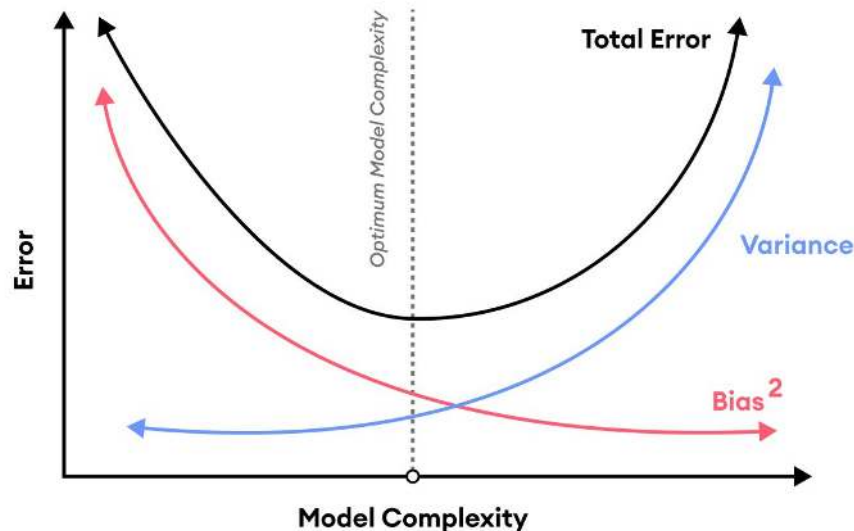


Bias-Varianza Tradeoff

Bias (sesgo): Es un error sistemático que ocurre en el modelo de aprendizaje automático como resultado de suposiciones erróneas al desarrollarlo.

La **varianza**, es un error de generalización, se produce debido a la excesiva sensibilidad del modelo a pequeñas variaciones en los datos de entrenamiento.

$$\text{Total Error} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



Previniedo Underfitting

- **Probar un modelo más complejo**
Agregue más variables
- **Incrementar el nro de ciclo (épocas) de training**
Aumentar el tiempo de entrenamiento del modelo
- **Eliminar ruido en los datos**
Eliminar valores extremos y dudosos
- **Reajustar los parámetros de regularización**
Estos parámetros mal ajustados son fuentes de overfitting también.
- **Probar otro modelo**
Si nada de lo anterior resulta

Previniedo Overfitting

- **Agregar más datos**

- **Early stopping**

En algoritmos iterativos detener el training cuando la performance comienza a deteriorarse.

- **Aumentación de datos**

Agregar datasets sintéticos o datos extra

- **Remoción de variables irrelevantes**

Estas variables no contribuyen mucho y meten ruido
(P.ej realizar un análisis de componentes principales (PCA), previo)

- **Regularización**

Agregar parámetros de penalización, podar arboles de decisión

- **Técnicas de ensemble: Boosting, Bagging**

Exactitud Predictiva

Imaginemos construir un modelo para emitir un juicio binario. ¿Es cierto correo electrónico spam?

Dado un conjunto de datos etiquetados y dicho modelo predictivo:

Cada punto de datos se encuentra en una de cuatro categorías:

Verdadero positivo: "El mensaje es spam y predijimos correctamente que era spam".

Falso positivo (error de tipo 1): "El mensaje no es spam, pero predijimos que sí lo era".

Falso negativo (error de tipo 2): "El mensaje es spam, pero predijimos que no lo era".

Verdadero negativo: "El mensaje no es spam y predijimos correctamente que no lo era".

Matriz de Confusión

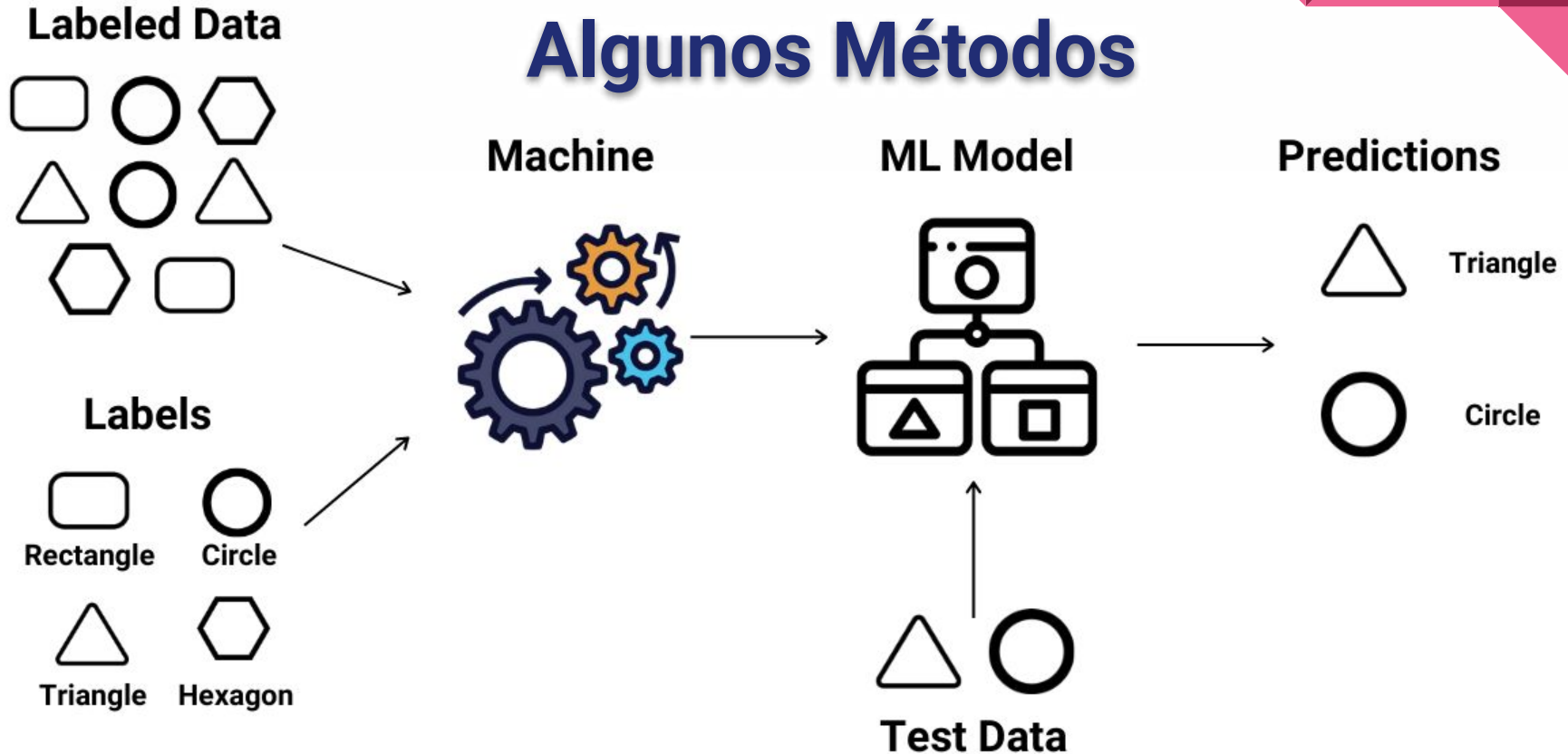
Modelo	Spam	No Spam
Predice Spam	Verdadero Positivo	Falso Positivo
Predice No Spam	Falso Negativo	Verdadero Negativo

predicho por el modelo


verdad conocida

Supervised Learning

Algunos Métodos



supervisados y no supervisados



● Data
— Linear Regression

```

graph TD
    Root[Colour = Red] -- Yes --> Model2010[Model = 2010]
    Root -- No --> ColourYellow[Colour = Yellow]
    Model2010 -- Yes --> Red1((Red))
    Model2010 -- No --> MileageYellow[Mileage = Yellow]
    MileageYellow -- Yes --> Red2((Red))
    MileageYellow -- No --> Yellow1((Yellow))
    ColourYellow -- Yes --> ModelNewer[Model = Newer]
    ColourYellow -- No --> Yellow2((Yellow))
    ModelNewer -- Yes --> Red3((Red))
    ModelNewer -- No --> Yellow3((Yellow))
  
```

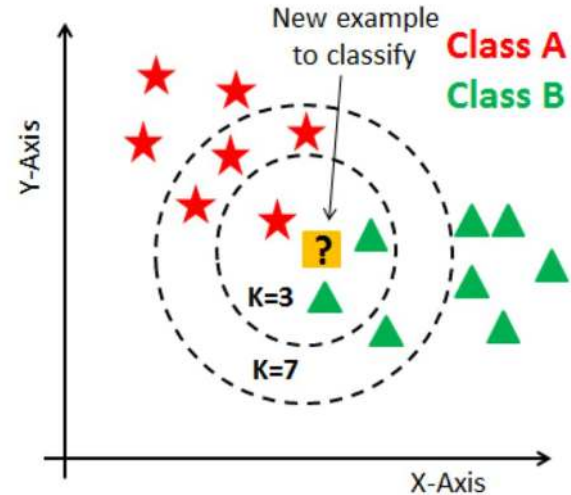

K-Nearest Neighbors

Fundamento

K-Nearest Neighbors (KNN) es un algoritmo de aprendizaje automático supervisado simple pero efectivo. Se puede utilizar tanto para problemas de clasificación como de regresión.

Idea básica: Predecir la etiqueta de un punto de datos:

- Observando los k puntos de datos etiquetados más cercanos
- Realizando una votación mayoritaria

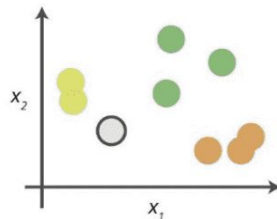


K-Nearest Neighbors

¿Cómo funciona?

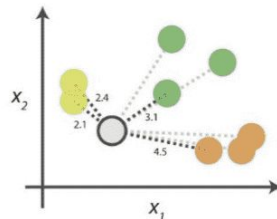
1. **Elegir el valor de K:** K representa el número de vecinos más cercanos a considerar.
2. **Calcular la distancia:** Se calcula la distancia entre el nuevo punto de datos y todos los demás puntos en el conjunto de entrenamiento. Se pueden utilizar varias métricas de distancia (por ejemplo, distancia euclidiana, distancia de Manhattan).
3. **Encontrar los K vecinos más cercanos:** Se identifican los K puntos más cercanos al nuevo punto de datos.
4. **Hacer la predicción:**
 - **Clasificación:** Se asigna la clase más común entre los K vecinos más cercanos al nuevo punto de datos.
 - **Regresión:** Se calcula el promedio (o la mediana) de los valores de los K vecinos más cercanos, y se asigna este valor al nuevo punto de datos.

0. Look at the data



Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

1. Calculate distances



Start by calculating the distances between the grey point and all other points.

2. Find neighbours

Point	Distance	
...	2.1	→ 1st NN
...	2.4	→ 2nd NN
...	3.1	→ 3rd NN
...	4.5	→ 4th NN

Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

3. Vote on labels

Class	# of votes	
	2	→ Class wins the vote! Point is therefore predicted to be of class .
	1	
	1	

Vote on the predicted class labels based on the classes of the k nearest neighbours. Here, the labels were predicted based on the k=3 nearest neighbours.

K-Nearest Neighbors

Características

- **Simple y fácil de entender.**
- **No paramétrico:** No hace suposiciones sobre la distribución de los datos.
- **Aprendizaje perezoso:** No construye un modelo explícito durante el entrenamiento. En cambio, almacena los datos de entrenamiento y realiza los cálculos en el momento de la predicción.
- **Sensible a la escala de los datos:** El escalado de características es importante para asegurar que todas las características contribuyan por igual al cálculo de la distancia.
- **La elección de K es crucial:** Un valor pequeño de K puede hacer que el modelo sea sensible al ruido, mientras que un valor grande de K puede llevar a un modelo sesgado.

Ventajas

- Fácil de implementar.
- Versátil (se puede utilizar para clasificación y regresión).
- Funciona bien con datos no lineales.

Desventajas

- Computacionalmente costoso para grandes conjuntos de datos.
- Sensible a datos irrelevantes y ruidosos.
- Requiere una elección cuidadosa de K.
- No funciona bien con datos de alta dimensionalidad.

El Dataset Iris

```
In [1]: from sklearn import datasets
```

```
In [2]: import pandas as pd
```

```
In [3]: import numpy as np
```

```
In [4]: import matplotlib.pyplot as plt
```

```
In [5]: plt.style.use('ggplot')
```

```
In [6]: iris = datasets.load_iris()
```

```
In [7]: type(iris)
```

```
Out[7]: sklearn.datasets.base.Bunch
```

```
In [8]: print(iris.keys())
```

```
dict_keys(['data', 'target_names', 'DESCR', 'feature_names', 'target'])
```

- Features:
 - Petal length
 - Petal width
 - Sepal length
 - Sepal width
- Target variable: Species
 - Versicolor
 - Virginica
 - Setosa



Scikit-Learn fit() & predict()

- Todos los modelos de aprendizaje automático se implementan como clases de Python.
 - Implementan los algoritmos de aprendizaje y predicción.
 - Almacenan la información obtenida de los datos.
- Entrenar un modelo con los datos ⇒ "adaptarlo" a los datos:
 - Método **.fit()**
- Para predecir las etiquetas de los nuevos datos:
 - Método **.predict()**

Análisis Exploratorio

```
In [9]: type(iris.data), type(iris.target)
```

```
Out[9]: (numpy.ndarray, numpy.ndarray)
```

```
In [10]: iris.data.shape
```

```
Out[10]: (150, 4)
```

```
In [11]: iris.target_names
```

```
Out[11]: array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

```
In [12]: X = iris.data
```

```
In [13]: y = iris.target
```

```
In [14]: df = pd.DataFrame(X, columns=iris.feature_names)
```

```
In [15]: print(df.head())
```

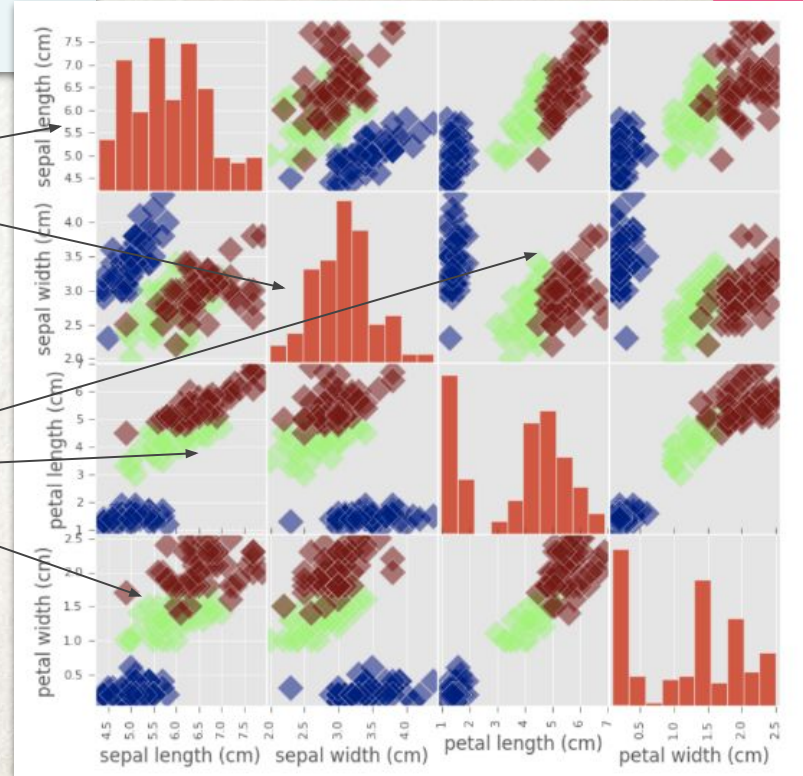
	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Análisis Exploratorio

```
In [16]: _ = pd.scatter_matrix(df, c = y, figsize = [8, 8],  
      ...:                      s=150, marker = 'D')
```

histogramas

scatter plots



K-Nearest Neighbors

Ejemplo: El Iris Dataset

```
In [1]: from sklearn.neighbors import KNeighborsClassifier
```

```
In [2]: knn = KNeighborsClassifier(n_neighbors=6)
```

```
In [3]: knn.fit(iris['data'], iris['target'])
```

```
Out[3]: KNeighborsClassifier(algorithm='auto', leaf_size=30,  
...: metric='minkowski', metric_params=None, n_jobs=1,  
...: n_neighbors=6, p=2, weights='uniform')
```

```
In [4]: iris['data'].shape
```

```
Out[4]: (150, 4)
```

```
In [5]: iris['target'].shape
```

```
Out[5]: (150,)
```


Prediciendo con KNN

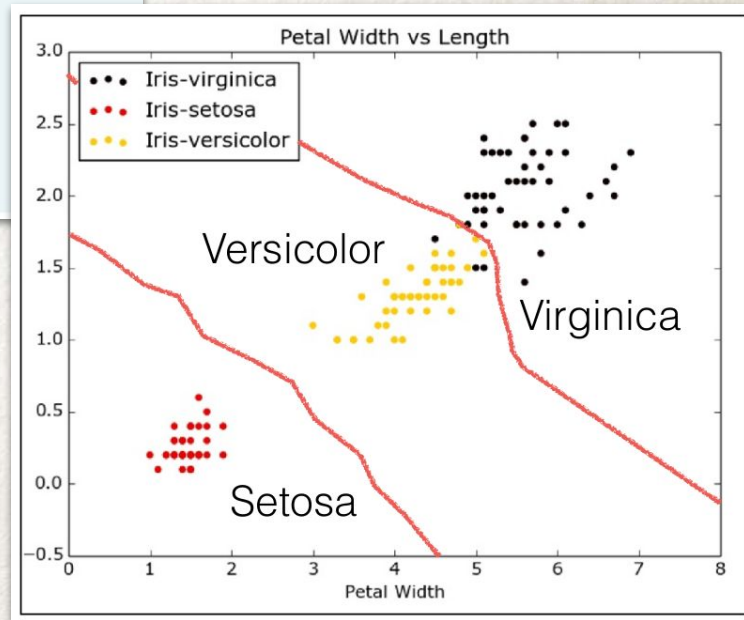
```
In [6]: prediction = knn.predict(X_new)
```

```
In [7]: X_new.shape
```

```
Out[7]: (3, 4)
```

```
In [8]: print('Prediction {}'.format(prediction))
```

```
Prediction: [1 1 0]
```



Distancias

- **Minkowski distance**

$$d(i, j) = \sqrt[q]{|x_{i1} - x_{j1}|^q + |x_{i2} - x_{j2}|^q + \dots + |x_{ip} - x_{jp}|^q}$$

$\xleftrightarrow{\text{1st dimension}} \quad \xleftrightarrow{\text{2nd dimension}} \quad \xleftrightarrow{\text{pth dimension}}$

- **Euclidean distance**

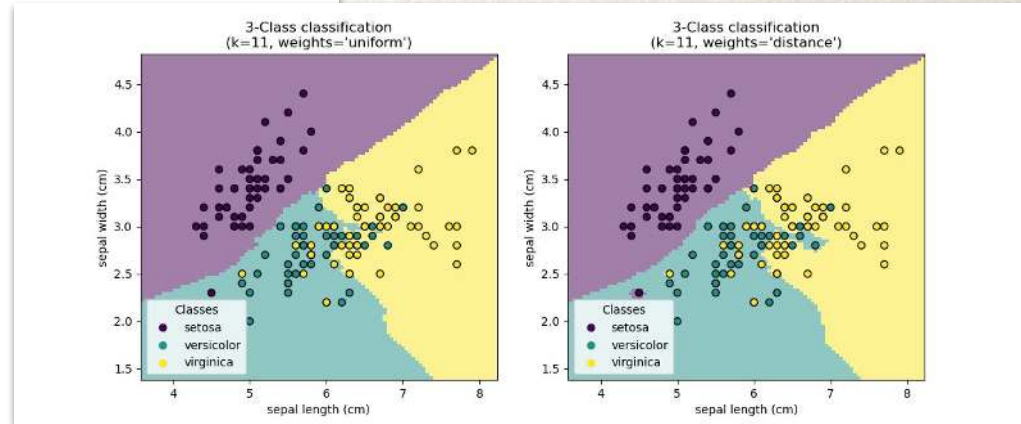
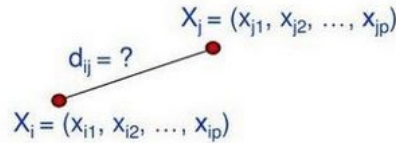
$$q = 2$$

$$d(i, j) = \sqrt{|x_{i1} - x_{j1}|^2 + |x_{i2} - x_{j2}|^2 + \dots + |x_{ip} - x_{jp}|^2}$$

- **Manhattan distance**

$$q = 1$$

$$d(i, j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + \dots + |x_{ip} - x_{jp}|$$



Normalización y Escalado de Características

¿Por qué es importante?

- Mejora el rendimiento del modelo
- Acelera la convergencia
- Aumenta la estabilidad

Técnicas Comunes

1. Normalización (Escalado Min-Max)

- Transforma los datos a un rango específico (generalmente 0 a 1)
- Fórmula: $X' = (X - X_{\min}) / (X_{\max} - X_{\min})$

2. Estandarización (Puntuación Z)

- Transforma los datos para que tengan una media de 0 y una desviación estándar de 1
- Fórmula: $X' = (X - \text{media}) / \text{desviación estándar}$

3. Escalado Robusto

- Menos sensible a los valores atípicos
- Utiliza la mediana y el rango intercuartílico (IQR)
- Fórmula: $X' = (X - \text{mediana}) / \text{IQR}$

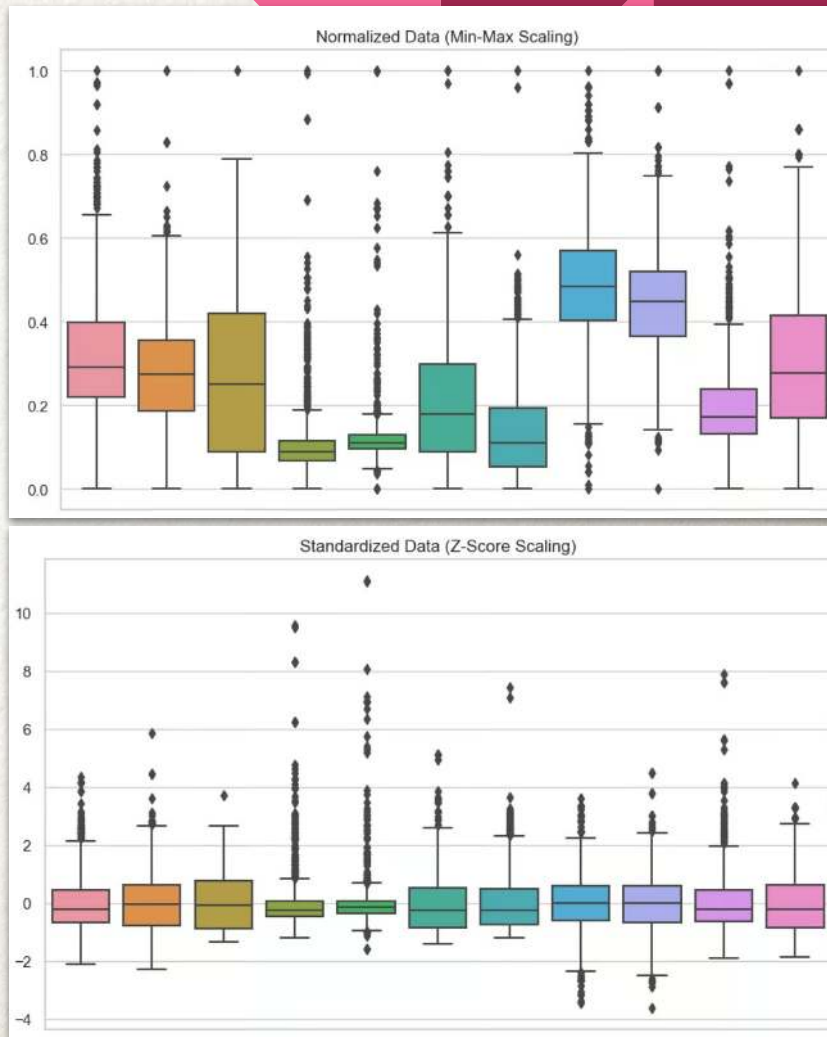
Cuando Usarlos

¿Cuándo usar qué técnica?

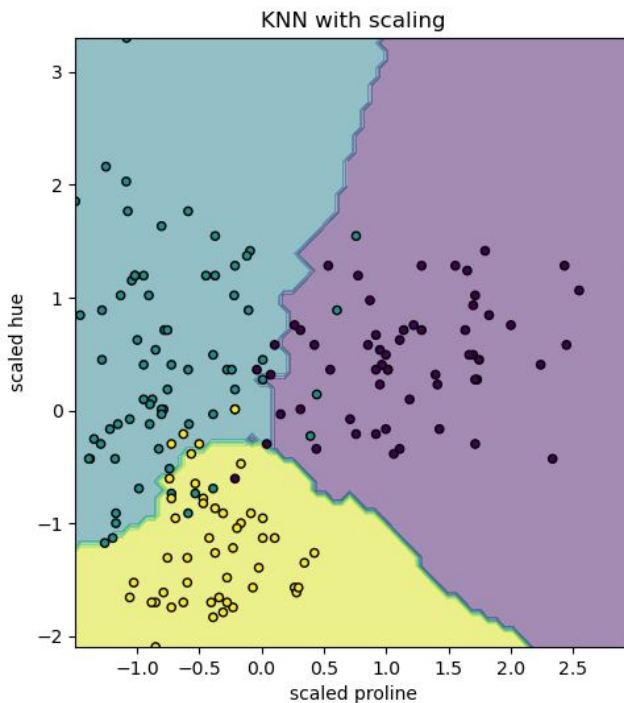
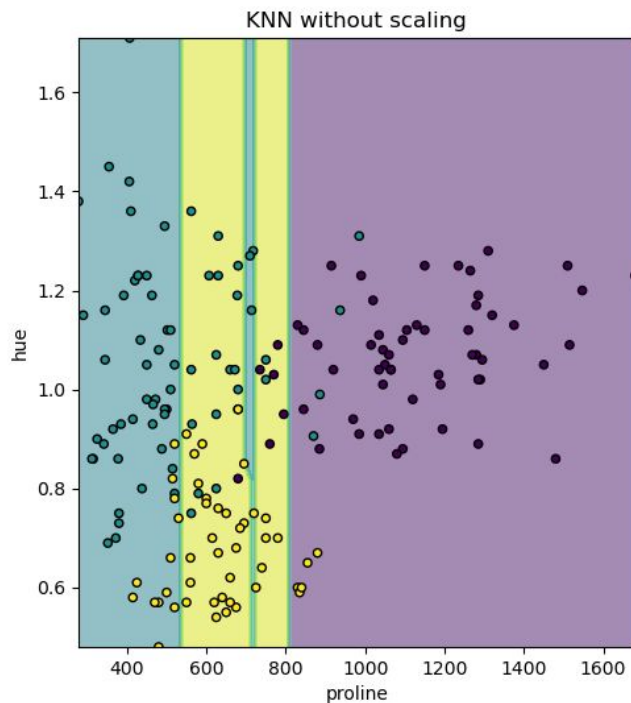
- **Normalización:** Cuando los datos tienen un rango acotado o no están distribuidos normalmente.
- **Estandarización:** Cuando los datos tienen una distribución normal o gaussiana.
- **Escalado Robusto:** Cuando hay valores atípicos significativos.

Consideraciones clave

- Aplicar el escalado después del train-test splitting
- Elegir la técnica adecuada
- Manejar las características categóricas por separado



Importancia de Escalar /Normalizar Features



Performance del Modelo

En la clasificación, la precisión la sensibilidad y el score F1 son métricas de uso común.

- ¿Qué datos se deben utilizar para calcular la precisión?
 - ¿Qué tan bien funcionará el modelo con nuevos datos?
-
- **Exactitud:** Proporción de predicciones correctas.
 - **Matriz de Confusión:** Tabla de rendimiento del modelo (verdaderos/falsos positivos/negativos).
 - **Precisión:** Proporción de positivos predichos correctamente.
 - **Sensibilidad/Recall:** Proporción de positivos reales predichos correctamente.
 - **Score F1:** Media armónica de precisión y sensibilidad
 - **AUC-ROC:** Capacidad del modelo para distinguir entre clases (0.5: aleatorio, 1: perfecto).

Métricas

Exactitud (Accuracy):

- La proporción de predicciones correctas del total de casos.
- Útil cuando las clases están **balanceadas**.
- Fórmula: $(TP+TN) / (TP+FN+FP+TN)$

Precisión (Precision):

- La proporción de predicciones positivas correctas del total de casos predichos como positivos.
- Útil cuando el costo de los falsos positivos es alto.
- Fórmula: $TP / (TP + FP)$

Sensibilidad/Recall:

- La proporción de casos positivos reales que se predicen correctamente.
- Útil cuando el costo de los falsos negativos es alto.
- Fórmula: $TP / (TP+ FN)$

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	TRUE POSITIVE TP	FALSE NEGATIVE FN
	Negative	FALSE POSITIVE FP	TRUE NEGATIVE TN

dataaspirant.com

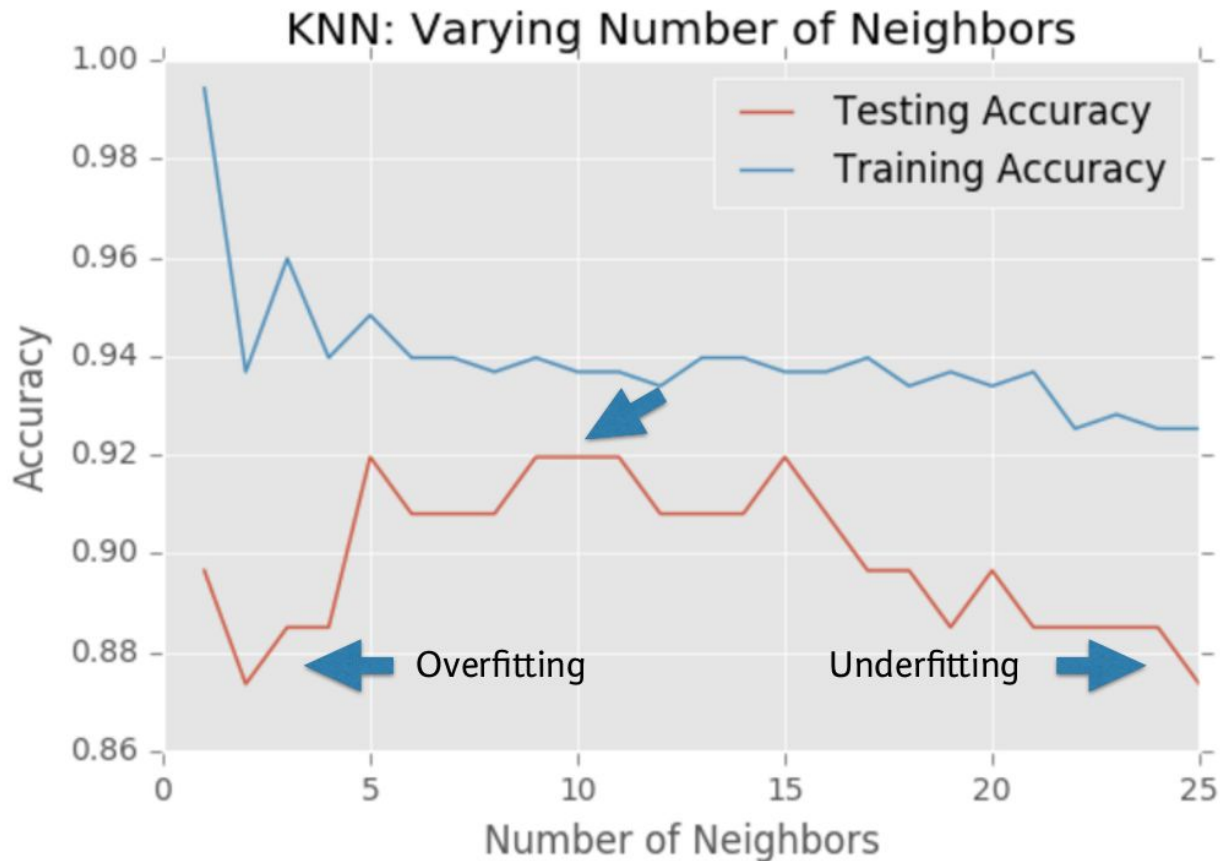
Score F1:

- La media armónica de precisión y recall.
- Útil cuando se busca un equilibrio entre precisión y exhaustividad.
- Fórmula: $2 * (Precisión * Recall) / (Precision + Recall)$

Área bajo la curva ROC (AUC-ROC):

- Mide la capacidad del modelo para distinguir entre clases.
- ROC es la Curva Característica Operativa del Receptor.
- AUC es el área bajo esta curva.
- Un valor de 0.5 indica un rendimiento aleatorio, y 1 indica un rendimiento perfecto.

Model complexity and over/underfitting



Exactitud o Accuracy

Ejemplo de desequilibrio de clases: Correos electrónicos

Clasificación de spam

No siempre es la métrica más útil

El 99 % de los correos electrónicos son reales; el 1 % son spam
Se podría crear un clasificador que prediga que TODOS los correos electrónicos son reales

- ¡99 % de precisión!
- Pero es pésimo para clasificar el spam
- Falla en su propósito original
- Se necesitan métricas más adecuadas

- **Alta precisión:** No se predijeron muchos correos electrónicos reales como spam.
- **Alta Sensibilidad:** Se predijo correctamente la mayoría de los correos electrónicos spam.

Ajustando la Performance

Si pudo calcular la precisión de los datos utilizados para ajustar el clasificador.

NO indica la capacidad de generalizar.
Se debe dividir los datos en conjuntos de entrenamiento y de prueba.

- Ajustar/entrenar el clasificador en el conjunto de entrenamiento.
- Hacer predicciones en el conjunto de prueba.
- Comparar las predicciones con las etiquetas conocidas.

```
In [1]: from sklearn.model_selection import train_test_split

In [2]: X_train, X_test, y_train, y_test =
...: train_test_split(X, y, test_size=0.3,
...:                  random_state=21, stratify=y)

In [3]: knn = KNeighborsClassifier(n_neighbors=8)

In [4]: knn.fit(X_train, y_train)

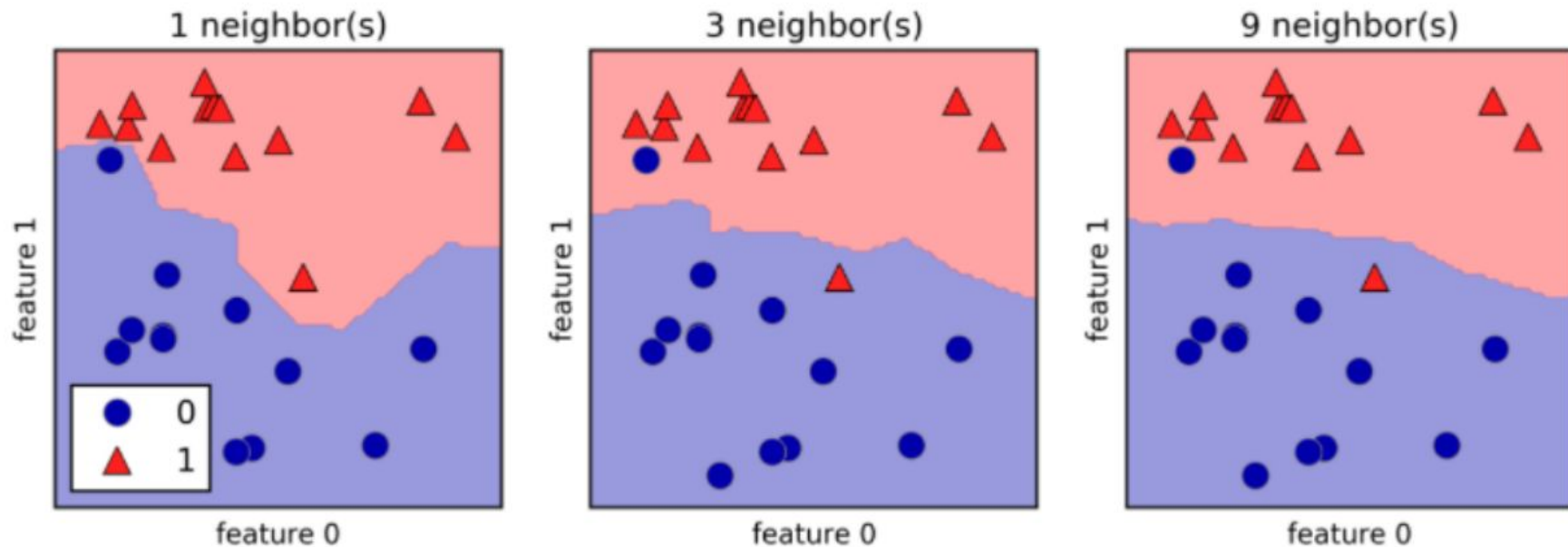
In [5]: y_pred = knn.predict(X_test)

In [6]: print("Test set predictions:\n {}".format(y_pred))
Test set predictions:
[2 1 2 2 1 0 1 0 0 1 0 2 0 2 2 0 0 0 1 0 2 2 2 0 1 1 1 0 0
 1 2 2 0 0 2 2 1 1 2 1 1 0 2 1]

In [7]: knn.score(X_test, y_test)
Out[7]: 0.9555555555555556
```

Complejidad del Modelo

En función de K



Reporte de Clasificación

```
In [1]: from sklearn.metrics import classification_report
```

```
In [2]: from sklearn.metrics import confusion_matrix
```

```
In [3]: knn = KNeighborsClassifier(n_neighbors=8)
```

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y,  
...: test_size=0.4, random_state=42)
```

```
In [5]: knn.fit(X_train, y_train)
```

```
In [6]: y_pred = knn.predict(X_test)
```

```
In [7]: print(confusion_matrix(y_test, y_pred))
```

```
[[52  7]  
 [ 3 112]]
```

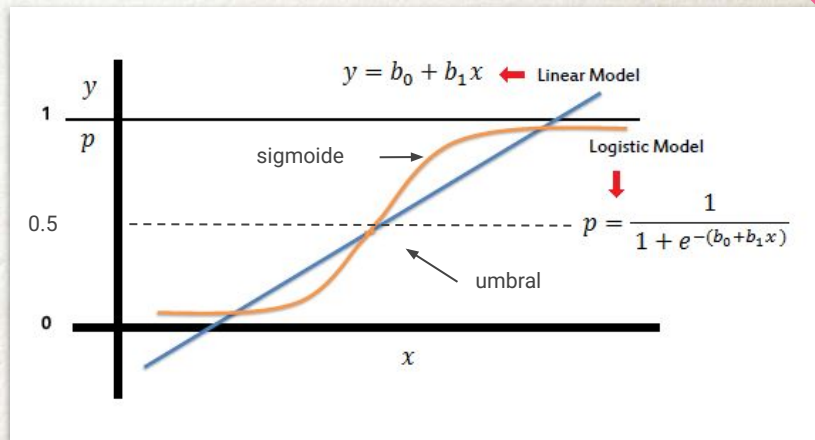
```
In [8]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.95	0.88	0.91	59
1	0.94	0.97	0.96	115
avg / total	0.94	0.94	0.94	174

Regresión Logística

Clasificación Binaria

- La regresión logística genera probabilidades p
- Se utiliza para predecir la probabilidad de que una instancia pertenezca a una determinada clase.
 - Si $p > 0.5$: Los datos se etiquetan como «1».
 - Si $p < 0.5$: Los datos se etiquetan como «0».
- Utiliza la función sigmoide para mapear cualquier valor de entrada a un valor entre 0 y 1, que representa una probabilidad.
- Aunque es un algoritmo de clasificación, la regresión logística es un modelo lineal. Encuentra la mejor línea que separa las dos clases.



Características

Cómo funciona?

1. **Hipótesis:** La regresión logística modela la probabilidad de que una instancia (**x**) pertenezca a una clase particular utilizando la función sigmoide:

$$p(x) = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$$

2. **Función de costo:** La función de costo utilizada en la regresión logística es la función de costo de entropía cruzada, que mide el rendimiento del modelo.
3. **Optimización:** El objetivo es minimizar la función de costo para encontrar los valores óptimos de los parámetros (**b₀**, **b₁**). Esto se hace típicamente utilizando algoritmos de optimización como el descenso de gradiente.
4. **Predicción:** Una vez que se han aprendido los parámetros óptimos, se puede utilizar el modelo para predecir la probabilidad de que nuevas instancias pertenezcan a una clase particular.

Ventajas:

- Fácil de implementar e interpretar.
- Eficiente para el entrenamiento.
- Funciona bien para problemas de clasificación linealmente separables.
- Proporciona probabilidades de clase.

Desventajas:

- Puede sufrir de overfitting si el número de características es demasiado grande en relación con el número de instancias.
- Solo puede modelar relaciones lineales entre las características y los log-odds de las clases.
- Requiere poco o ningún ruido entre las variables independientes y la variable dependiente.
- El rendimiento se degrada si las características son multicolineales.

Medidas de Performance

Loss Function (Función de Pérdida)

$$L(\hat{y}, y) = - (y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

- \hat{y} Valor predicho
- y Valor verdadero

La función de pérdida (L) cuantifica qué tan bien el modelo estima la relación entre las características de entrada (X) y la salida (Y) para una sola muestra de entrenamiento,

Para las m muestras del conjunto de training definimos la **Función de Costo (Cost Function): J**

J es la media de la entropía cruzada de pérdida

$$J = \frac{1}{m} \sum_{i=1}^m L(\hat{y}_i, y_i)$$

El objetivo de la regresión logística es minimizar la función de costo (J).

Implementación en scikit-learn

```
In [1]: from sklearn.linear_model import LogisticRegression
```

```
In [2]: from sklearn.model_selection import train_test_split
```

```
In [3]: logreg = LogisticRegression()
```

```
In [4]: X_train, X_test, y_train, y_test = train_test_split(X, y,  
    ...: test_size=0.4, random_state=42)
```

```
In [5]: logreg.fit(X_train, y_train)
```

```
In [6]: y_pred = logreg.predict(X_test)
```

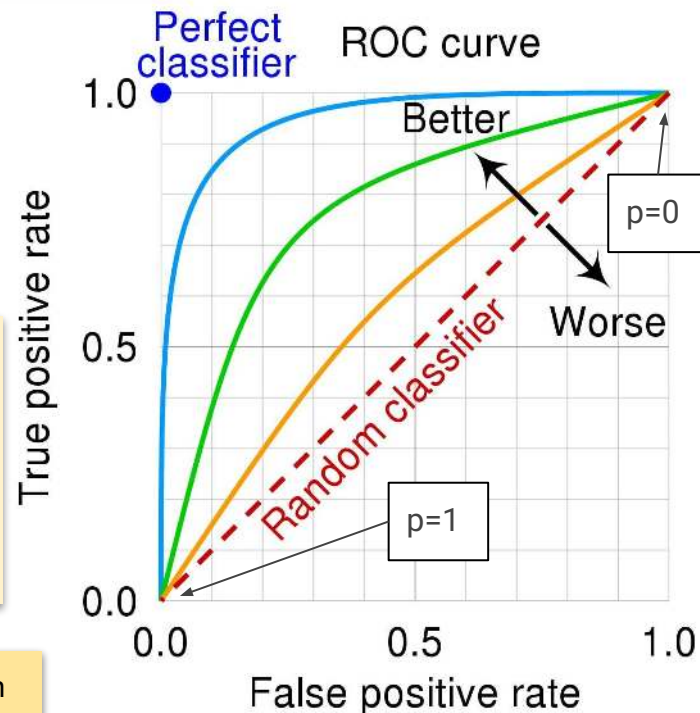

Umbral de Probabilidad

- Por defecto, el umbral de regresión logística es 0.5
- No es específico de la regresión logística
- Los clasificadores k-NN también tienen umbrales
- ¿Qué ocurre si variamos el umbral?

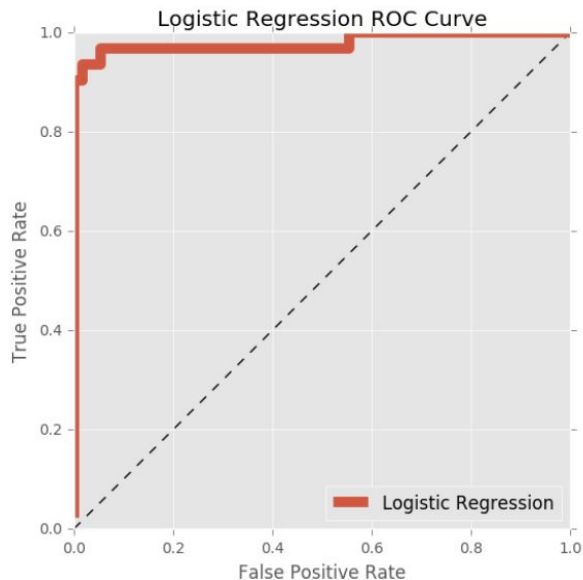
Curva ROC y AUC: La **Curva Operativa del Receptor (ROC)** traza la tasa de verdaderos positivos contra la tasa de falsos positivos a varios umbrales de clasificación. El **Área Bajo la Curva (AUC)** representa la capacidad del modelo para distinguir entre clases.

Pérdida logarítmica (Log Loss): Mide el rendimiento de un modelo de clasificación donde la salida es un valor de probabilidad entre 0 y 1.

Criterio de información de Akaike (AIC): Una medida de la bondad de ajuste de un modelo que también penaliza la complejidad del modelo.



Receiver Operator Curve (ROC)



```
logreg.predict_proba(X_test)[: ,1]
```

```
In [1]: from sklearn.metrics import roc_curve
```

```
In [2]: y_pred_prob = logreg.predict_proba(X_test)[: ,1]
```

```
In [3]: fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
```

```
In [4]: plt.plot([0, 1], [0, 1], 'k--')
```

```
In [5]: plt.plot(fpr, tpr, label='Logistic Regression')
```

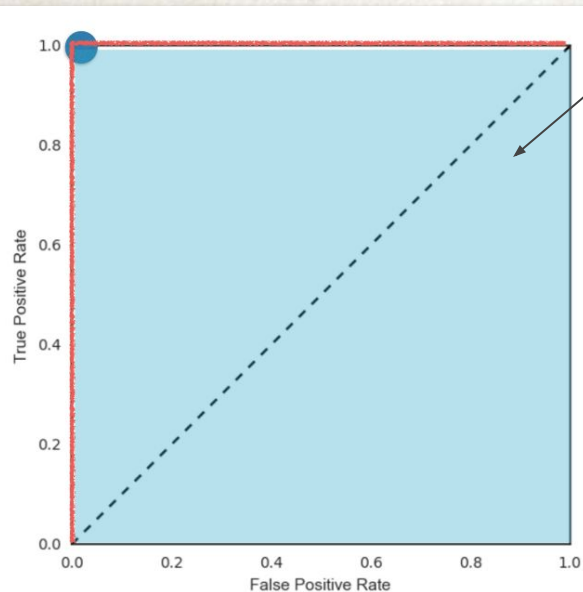
```
In [6]: plt.xlabel('False Positive Rate')
```

```
In [7]: plt.ylabel('True Positive Rate')
```

```
In [8]: plt.title('Logistic Regression ROC Curve')
```

```
In [9]: plt.show();
```

ROC: Area Bajo la Curva (AUC)



Área más grande bajo la curva ROC = mejor modelo

```
In [1]: from sklearn.metrics import roc_auc_score
```

```
In [2]: logreg = LogisticRegression()
```

```
In [3]: X_train, X_test, y_train, y_test = train_test_split(X, y,  
...: test_size=0.4, random_state=42)
```

```
In [4]: logreg.fit(X_train, y_train)
```

```
In [5]: y_pred_prob = logreg.predict_proba(X_test)[:,-1]
```

```
In [6]: roc_auc_score(y_test, y_pred_prob)
```

```
Out[6]: 0.997466216216
```

Cross-Validation

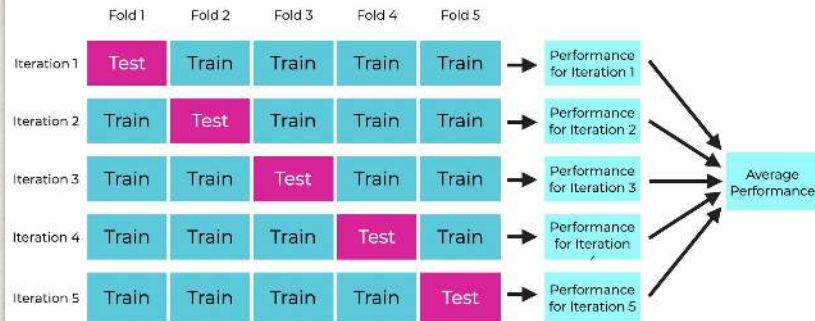
La validación cruzada es una técnica esencial en el aprendizaje automático para evaluar el rendimiento de un modelo y su capacidad para generalizar a datos no vistos. Ayuda a mitigar problemas como el sobreajuste y proporciona una estimación más confiable del rendimiento del modelo.

¿Qué es la validación cruzada?

La validación cruzada es un método de remuestreo que evalúa un modelo de aprendizaje automático dividiendo el conjunto de datos en varias particiones (o "pliegues"), entrenando el modelo en algunas de estas particiones y evaluándolo en las restantes. Este proceso se repite varias veces, utilizando diferentes particiones como conjuntos de evaluación en cada iteración, y luego se combinan los resultados para obtener una estimación del rendimiento del modelo.

Validación cruzada k-fold: El conjunto de datos se divide en k pliegues. El modelo se entrena en k-1 pliegues y se evalúa en el pliegue restante. Este proceso se repite k veces, cada vez dejando fuera un pliegue diferente para la evaluación. El rendimiento del modelo se promedia en todos los k pliegues.

CROSS VALIDATION, EXPLAINED



AUC using cross-validation

```
In [7]: from sklearn.model_selection import cross_val_score
```

```
In [8]: cv_scores = cross_val_score(logreg, X, y, cv=5,  
    ...:                             scoring='roc_auc')
```

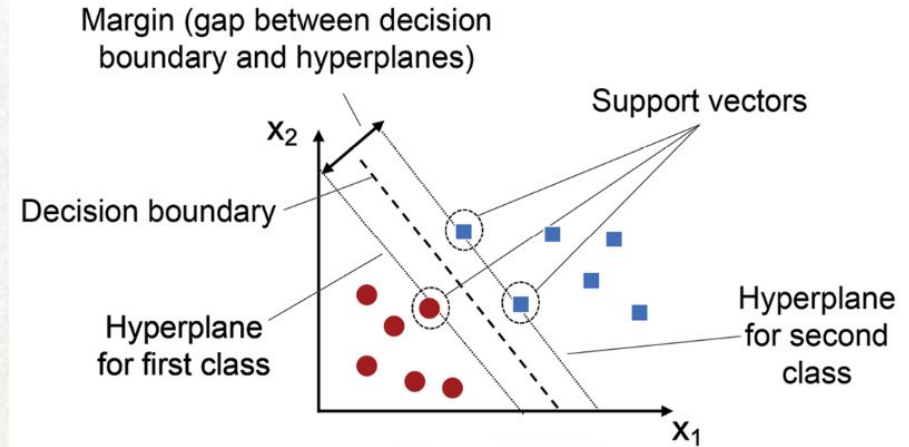
```
In [9]: print(cv_scores)
```

```
[ 0.99673203  0.99183007  0.99583796  1.          0.96140652]
```

Support Vector Machine (SVM)

La Máquina de Vectores de Soporte (SVM) es un algoritmo de aprendizaje automático supervisado que se utiliza para tareas de clasificación y regresión. Si bien puede gestionar problemas de regresión, la SVM es especialmente adecuada para tareas de clasificación.

La SVM busca encontrar el hiperplano óptimo en un espacio N-dimensional para separar los puntos de datos en diferentes clases. El algoritmo maximiza el margen entre los puntos más cercanos de diferentes clases.



Terminología usada en SVM

Hiperplano: Un límite de decisión que separa diferentes clases en el espacio de características, representado por la ecuación $wx + b = 0$ en la clasificación lineal.

Vectores de Soporte: Los puntos de datos más cercanos al hiperplano, cruciales para determinar el hiperplano y el margen en la MTS.

Margen: La distancia entre el hiperplano y los vectores de soporte. La SVM busca maximizar este margen para un mejor rendimiento de la clasificación.

Núcleo (Kernel): Una función que asigna los datos a un espacio de mayor dimensión, lo que permite a la MTS gestionar datos separables de forma no lineal.

Margen Duro: Un hiperplano de margen máximo que separa perfectamente los datos sin errores de clasificación.

Margen Suave: Permite algunos errores de clasificación mediante la introducción de variables de holgura, equilibrando la maximización del margen y las penalizaciones por errores de clasificación cuando los datos no son perfectamente separables.

C: Un término de regularización que equilibra la maximización del margen y las penalizaciones por errores de clasificación. Un valor C más alto aplica una penalización más estricta por errores de clasificación.

Pérdida de Bisagra (Hinge Loss): Función de pérdida que penaliza puntos mal clasificados o violaciones de margen, combinada con regularización en SVM.

Problema Dual: Implica la resolución de multiplicadores de Lagrange asociados con vectores de soporte, lo que facilita el truco del kernel y un cálculo eficiente.

Prediciendo Cáncer de Mama

Objetivo: Predecir si el cáncer es benigno o maligno. El uso de datos históricos de pacientes diagnosticados con cáncer permite a los médicos diferenciar los casos malignos de los benignos, a los que se les asignan atributos independientes.

- Cargar el conjunto de datos de cáncer de mama desde sklearn.datasets.
- Separar las características de entrada y las variables objetivo.
- Construir y entrenar los clasificadores SVM utilizando el kernel RBF.
- Dibujar el diagrama de dispersión de las características de entrada.

```
# Load the important packages
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

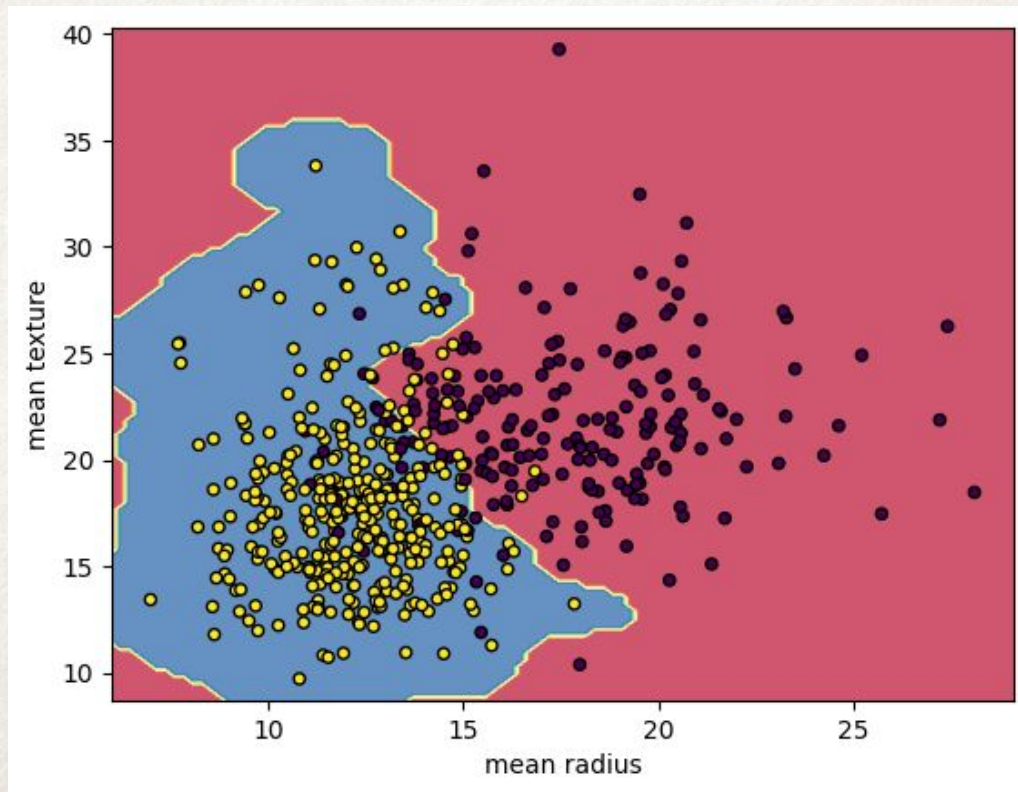
# Load the datasets
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

# Build the model
svm = SVC(kernel="rbf", gamma=0.5, C=1.0)
# Trained the model
svm.fit(X, y)

# Plot Decision Boundary
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    cmap=plt.cm.Spectral,
    alpha=0.8,
    xlabel=cancer.feature_names[0],
    ylabel=cancer.feature_names[1],
)

# Scatter plot
plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```


Resultados



Ventajas y Desventajas

Ventajas

- **Rendimiento en altas dimensiones:** La SVM destaca en espacios de alta dimensión, lo que la hace ideal para la clasificación de imágenes y el análisis de expresión génica.
- **Capacidad no lineal:** Mediante el uso de funciones kernel como RBF y polinomiales, la SVM gestiona eficazmente las relaciones no lineales.
- **Resiliencia de valores atípicos:** La función de margen suave permite a la SVM ignorar los valores atípicos, lo que mejora la robustez en la detección de spam y anomalías.
- **Compatibilidad binaria y multiclase:** La SVM es eficaz tanto para la clasificación binaria como para la multiclase, ideal para aplicaciones de clasificación de texto.
- **Eficiencia de memoria:** La SVM se centra en los vectores de soporte, lo que la hace eficiente en el uso de memoria en comparación con otros algoritmos.

Desventajas

- **Entrenamiento lento:** Las SVM pueden ser lentas con grandes conjuntos de datos, lo que afecta su rendimiento en tareas de minería de datos.
- **Dificultad de ajuste de parámetros:** Seleccionar el kernel correcto y ajustar parámetros como C requiere un ajuste minucioso, lo que afecta a los algoritmos de las SVM.
- **Sensibilidad al ruido:** Las SVM presentan dificultades con conjuntos de datos ruidosos y clases superpuestas, lo que limita su eficacia en situaciones reales.
- **Interpretabilidad limitada:** La complejidad del hiperplano en dimensiones superiores hace que las SVM sean menos interpretables que otros modelos.
- **Sensibilidad al escalado de características:** Un escalado de características adecuado es esencial; de lo contrario, los modelos de SVM podrían tener un rendimiento deficiente.

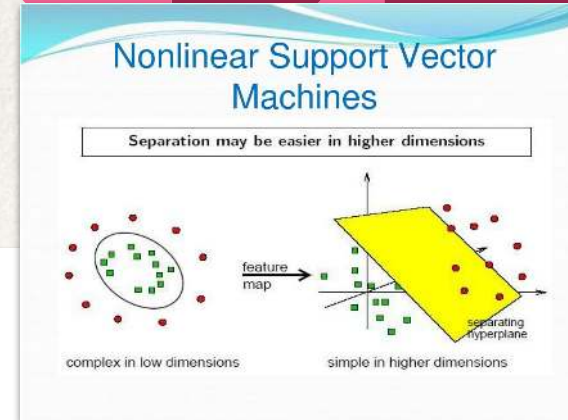
Tipos de SVM

Tipos de Máquinas de Vectores de Soporte

Según la naturaleza del límite de decisión:

SVM lineal: Utilizan un límite de decisión lineal para separar los puntos de datos de diferentes clases. Cuando los datos se pueden separar linealmente con precisión, las SVM lineales son muy adecuadas. Esto significa que una sola línea recta (en 2D) o un hiperplano (en dimensiones superiores) puede dividir completamente los puntos de datos en sus respectivas clases. Un hiperplano que maximiza el margen entre las clases es el límite de decisión.

SVM no lineal: Se pueden utilizar para clasificar datos cuando no se pueden separar en dos clases mediante una línea recta (en el caso de 2D). Mediante el uso de funciones kernel, las SVM no lineales pueden manejar datos separables de forma no lineal. Los datos de entrada originales se transforman mediante estas funciones kernel en un espacio de características de mayor dimensión, donde los puntos de datos se pueden separar linealmente. Una SVM lineal se utiliza para localizar un límite de decisión no lineal en este espacio modificado.



Sumario

- El aprendizaje supervisado trabaja con datos etiquetados
- Las dos metodologías usadas son Regresión y Clasificación
- Para poder implementar un Algoritmo de Aprendizaje Automático debemos Realizar un Análisis Exploratorio de Datos en nuestros Datasets.
- Las variables categóricas deben ser codificadas numéricamente
- El concepto de Bias y Varianza está íntimamente ligado con la bondad de ajuste del modelo (underfitting & overfitting)
- Es importante Normalizar/Estandarizar los datos de entrada de forma que no prevalezcan unas variables sobre las otras.
- KNN es uno de los algoritmos más usados en clasificación y regresión
- La regresión logística se usa en clasificación de variables binarias
- Support vector Machines (SVM) es muy eficiente en clasificación multiclase.
- Todos estos algoritmos también se pueden usar para hacer Regresión
- Todas las técnicas tienen sus ventajas y desventajas, su aplicación depende del problema a resolver.