Guía de estudio - Aprendizaje Supervisado: Clasificación

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

En el campo del aprendizaje automático, los algoritmos supervisados de clasificación son herramientas poderosas para resolver problemas en los que se requiere predecir la clase o categoría de una muestra en función de características o atributos dados. Estos algoritmos utilizan un conjunto de datos de entrenamiento con etiquetas conocidas para aprender patrones y realizar predicciones precisas sobre nuevos datos.

En esta guía, exploraremos algunos de los algoritmos supervisados más populares y discutiremos métricas clave para evaluar su rendimiento.

Comenzaremos con los Árboles de Decisión, que son estructuras jerárquicas que toman decisiones mediante la división del espacio de características en diferentes regiones. Aprenderemos cómo construir árboles de decisión, calcular la impureza de Gini y la ganancia de información, y cómo manejar la poda para evitar el sobreajuste.

Luego, nos adentraremos en el algoritmo K-Nearest Neighbors (KNN), un método intuitivo que clasifica las muestras en función de la cercanía a sus vecinos más cercanos. Veremos cómo funciona KNN, cómo seleccionar el valor de K y cómo medir la distancia entre las muestras.

Además de explorar los algoritmos de clasificación, también discutiremos métricas clave para evaluar su rendimiento. La precisión es una de ellas y mide la proporción de predicciones correctas realizadas por el modelo en relación con todas las predicciones realizadas. También exploramos el recall (sensibilidad), que mide la proporción de casos positivos que el modelo identifica correctamente. Analizaremos su importancia y cómo se relaciona con la precisión.

Otra métrica importante es el valor F1, que combina la precisión y el recall en una sola métrica. Discutiremos cómo se calcula y cuándo es útil utilizarlo en la evaluación de modelos de clasificación.

Para aplicar todos estos conceptos, utilizaremos un caso de estudio con un conjunto de datos de clasificación. Implementaremos los algoritmos mencionados y evaluaremos su

rendimiento utilizando las métricas de evaluación discutidas. También veremos cómo implementar los algoritmos en Python y cómo interpretar los resultados obtenidos.

En resumen, esta guía te proporcionará una introducción sólida a algunos de los algoritmos supervisados de clasificación más populares y a las métricas clave para evaluar su rendimiento. Aprenderás a comprender y aplicar con confianza estos algoritmos en tus propios proyectos de aprendizaje automático.

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Aprendizaje Supervisado

El aprendizaje supervisado es una rama del aprendizaje automático que se enfoca en entrenar modelos utilizando un conjunto de datos etiquetados, donde cada muestra tiene una etiqueta o clase asociada. El objetivo es que el modelo aprenda a mapear las características de las muestras a las etiquetas correspondientes, para poder hacer predicciones precisas sobre nuevas muestras no etiquetadas.

Una analogía sencilla para comprender el aprendizaje supervisado es pensar en un profesor y sus alumnos. Imagina que el profesor tiene un conjunto de ejercicios resueltos con sus respectivas respuestas (etiquetas). Durante la clase, el profesor muestra a los alumnos estos ejercicios y les explica cómo llegar a las respuestas correctas. A medida que los alumnos practican y resuelven ejercicios similares, el profesor evalúa sus respuestas y les brinda retroalimentación.

En este caso, el profesor representa al modelo de aprendizaje supervisado, los ejercicios resueltos son las muestras con sus etiquetas, y los alumnos son los datos de entrenamiento. El objetivo del profesor es enseñar a los alumnos a resolver los ejercicios correctamente, para que puedan aplicar ese conocimiento a ejercicios nuevos y obtener respuestas precisas.

De manera similar, en el aprendizaje supervisado, el modelo "aprende" a partir de ejemplos etiquetados y busca generalizar ese conocimiento para realizar predicciones correctas sobre nuevas muestras. El conjunto de datos etiquetados es como el conjunto de ejercicios resueltos y las características de las muestras corresponden a los pasos o reglas utilizados para resolver los ejercicios.

En resumen, el aprendizaje supervisado se basa en el principio de enseñanza y aprendizaje, donde un modelo aprende a partir de ejemplos etiquetados para realizar predicciones precisas sobre nuevas muestras.

Clasificación

La tarea de clasificación es un problema en el aprendizaje automático en el que se asigna una etiqueta o clase a cada muestra o instancia en función de sus características o atributos. El objetivo es entrenar un modelo que pueda aprender a clasificar nuevas muestras correctamente en las mismas categorías que las muestras de entrenamiento.

Para ilustrar esta tarea, consideremos un ejemplo simple en el que queremos decidir si debemos llevar un paraguas o no en función de la temperatura y el pronóstico del tiempo. Supongamos que tenemos un conjunto de datos con diferentes registros que incluyen la temperatura y el pronóstico del día, así como una etiqueta que indica si llevamos un paraguas o no.

Temperatura	Pronóstico	Llevar paraguas
26°C	Soleado	No
20°C	Nublado	Sí
18°C	Lluvioso	Sí
23°C	Nublado	No
15°C	Lluvioso	Sí

Tabla 1. Ejemplo de clasificación.

En este ejemplo, queremos entrenar un modelo de clasificación que pueda predecir si debemos llevar un paraguas o no en función de la temperatura y el pronóstico. Las características o atributos son la temperatura y el pronóstico, y la etiqueta es si llevamos un paraguas o no.

Aplicaciones de algoritmos de clasificación

Existen numerosos ejemplos de aplicaciones de clasificación en diferentes industrias. Aquí tienes algunos ejemplos de clasificación en industrias específicas:

1. **Industria de la salud:**
    a. Diagnóstico de enfermedades: Utilizando datos médicos, como imágenes médicas o resultados de pruebas, se pueden construir modelos de clasificación para diagnosticar enfermedades como cáncer, enfermedades cardíacas, enfermedades pulmonares, etc.

    b. Detección de fraudes en seguros de salud: Se pueden utilizar modelos de clasificación para identificar patrones sospechosos y detectar fraudes en reclamaciones de seguros de salud.

2. **Industria financiera:**
    a. Detección de fraudes en transacciones: Los algoritmos de clasificación pueden ayudar a identificar transacciones fraudulentas, como transacciones con tarjetas de crédito robadas o actividades sospechosas en cuentas bancarias.

    b. Evaluación crediticia: Los modelos de clasificación pueden utilizarse para evaluar el riesgo crediticio de los solicitantes y predecir su probabilidad de incumplimiento en el pago de préstamos.

3. **Industria minorista:**
    a. Segmentación de clientes: Los modelos de clasificación pueden agrupar a los clientes en diferentes segmentos con características similares, lo que ayuda a personalizar las estrategias de marketing y ofrecer productos y ofertas relevantes.

    b. Detección de fraudes en comercio electrónico: Se pueden construir modelos de clasificación para detectar actividades fraudulentas, como compras con tarjetas de crédito robadas o cuentas falsas.

4. **Industria de manufactura:**
    a. Control de calidad: Los modelos de clasificación pueden utilizarse para clasificar productos en diferentes categorías de calidad, lo que ayuda en la detección de defectos y la toma de decisiones sobre la aceptación o rechazo de productos.

    b. Mantenimiento predictivo: Los algoritmos de clasificación pueden predecir el estado de las máquinas y equipos para programar el mantenimiento de manera proactiva y evitar fallas costosas.

Estos son solo algunos ejemplos, y la clasificación se puede aplicar en una amplia gama de industrias, como telecomunicaciones, transporte, energía, agricultura, entre otras. Los modelos de clasificación ayudan a automatizar tareas, tomar decisiones basadas en datos y mejorar la eficiencia en diversas industrias.

Reflexiona:

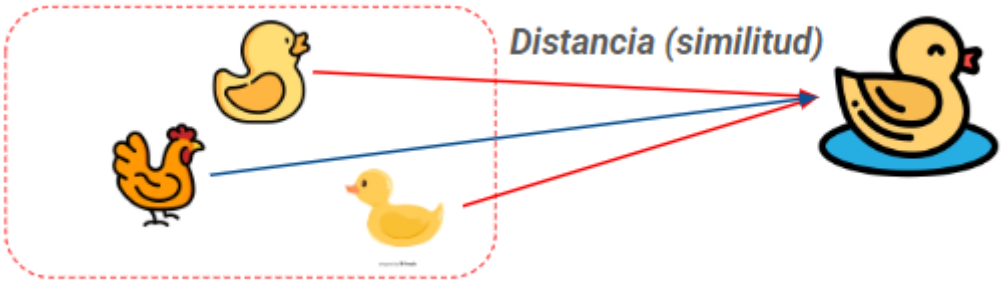
- En tu área de trabajo o conocimiento, ¿qué aplicaciones puede tener la clasificación?

K-vecinos más cercanos (KNN)

El algoritmo K-Nearest Neighbors (KNN) es un algoritmo de clasificación supervisado que se basa en el concepto de "vecinos más cercanos". En resumen, clasifica una muestra nueva en función de las clases de las muestras cercanas en el espacio de características.

La idea principal detrás de KNN es que las muestras que comparten características similares tienden a agruparse en el espacio. Por lo tanto, cuando se debe clasificar una nueva muestra, KNN busca los K vecinos más cercanos en función de una medida de distancia (como la distancia euclidiana) y asigna la clase más común entre esos vecinos a la nueva muestra. Una frase que resume la idea es “dime con quién andas y te diré quien eres”.

Una analogía útil para entender KNN es imaginar un vecindario donde cada casa tiene ciertas características, como el tamaño, el número de habitaciones y el precio de venta. Si deseas determinar el precio de venta de una casa nueva, puedes buscar en el vecindario las K casas más similares en términos de características y tomar el precio promedio de esas K casas como una estimación del precio de venta de la casa nueva.



Fuente: Desafío Latam

Algoritmo

El algoritmo K-Nearest Neighbors (KNN) consta de las siguientes etapas:

1. **Preparación de datos:** En esta etapa, se realiza el preprocesamiento de los datos, que puede incluir la limpieza de datos faltantes, la normalización de características y la codificación de variables categóricas. Es importante asegurarse de que los datos estén en un formato adecuado para el algoritmo KNN.

2. **Selección del valor de K:** El valor de K es un hiper parámetro que determina la cantidad de vecinos más cercanos que se considerarán para la clasificación. La elección del valor de K es crítica, ya que un valor incorrecto puede afectar el

rendimiento del modelo. Se pueden probar diferentes valores de K y seleccionar aquel que brinde el mejor rendimiento.

3. **Cálculo de distancias:** En esta etapa, se calcula la distancia entre la muestra a clasificar y todas las muestras del conjunto de entrenamiento. La distancia más comúnmente utilizada es la distancia euclidiana, aunque también se pueden utilizar otras métricas de distancia, como la distancia de Manhattan o la distancia de Minkowski.
   Ejemplo distancia euclidiana:

$$d = \sqrt{\sum (p_i - q_i)^2}$$

4. **Identificación de los K vecinos más cercanos:** Se seleccionan los K vecinos más cercanos a la muestra a clasificar en función de las distancias calculadas en la etapa anterior. Estos vecinos serán utilizados para determinar la clase de la muestra de prueba.

5. **Clasificación:** En esta etapa, se determina la clase de la muestra a clasificar en función de la clase más común entre los 'k' vecinos más cercanos. Por lo general, se utiliza la votación mayoritaria, donde la clase más frecuente entre los vecinos se asigna a la muestra.

   La siguiente fórmula corresponde a la forma en que el algoritmo realiza la clasificación en base a los 'k' vecinos más cercanos de la nueva muestra.

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

Es importante destacar que estas etapas son generales y pueden adaptarse según los requisitos y el contexto del problema específico. Además, el algoritmo KNN es bastante flexible y se puede modificar para adaptarse a diferentes necesidades, como utilizar pesos en función de la distancia o utilizar técnicas de reducción de dimensionalidad para mejorar el rendimiento.

Actividad guiada: Entrenando un algoritmo con KNN

1. Importamos las bibliotecas necesarias, y el dataset Iris

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
```

```
# Cargar el conjunto de datos Iris
data = load_iris()
X = data.data
y = data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Se importa "numpy" para el manejo de los datos y "matplotlib" para realizar los gráficos necesarios. Luego, se utilizan diferentes métodos de sklearn como dataset para importar el test de datos de iris, model_selection para utilizar el método train_test_split que es utilizado para dividir los datos en entrenamiento y validación, y finalmente KNeighborsClassifier que corresponde a la implementación del algoritmo KNN.

2. Creamos un clasificador KNN, considerando los 5 vecinos más cercanos. A partir de esto realizamos el entrenamiento y test

```
# Crear el clasificador KNN con K=5
knn = KNeighborsClassifier(n_neighbors=5)

# Entrenar el clasificador
knn.fit(X_train, y_train)

new_data = X_test[-3:]

# Realizar predicciones sobre los nuevos datos
predictions = knn.predict(new_data)
predictions

Output:
array([2, 0, 0])
```

Vemos así que al primer dato se le asignó la etiqueta "2", y a la segunda y la tercera se les asignó la etiqueta "0"

3. Visualizar las nuevas predicciones

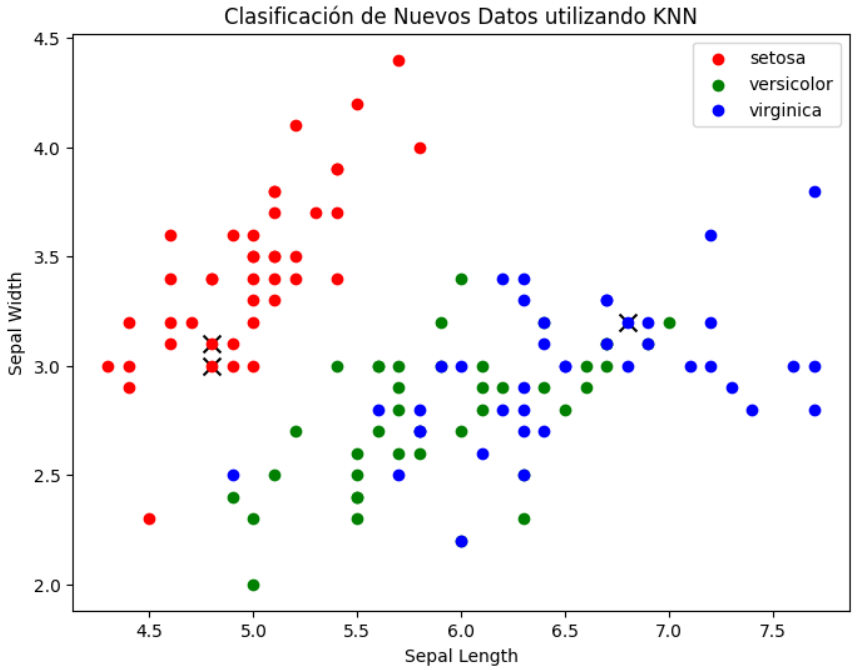
Se busca graficar el set de entrenamiento y en conjunto las nuevas muestras que se clasifican con una x, para mostrar que son nuevos datos.

```
# Graficar los nuevos datos junto con las clasificaciones predichas
plt.figure(figsize=(8, 6))
```

```
colors = ['red', 'green', 'blue']
for i in range(len(new_data)):
    plt.scatter(new_data[i, 0], new_data[i, 1], color='black', marker='x',
s=100)
    plt.scatter(new_data[i, 0], new_data[i, 1], color=colors[predictions[i]],
)

# Graficar los puntos del conjunto de datos original
for i in range(3):
    plt.scatter(X_train[y_train == i, 0], X_train[y_train == i, 1],
color=colors[i], label=data.target_names[i])

plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Clasificación de Nuevos Datos utilizando KNN')
plt.legend()
plt.show()
```



Como podemos observar, los datos “nuevos” son clasificados en relación con su cercanía con los demás “vecinos”, y tomando el color de la mayoría de ellos.

Ventajas y desventajas

El algoritmo KNN tiene sus ventajas y desventajas. Aquí hay una descripción de algunas de ellas, junto con posibles soluciones a las desventajas:

Ventajas

1. Fácil de entender e implementar.
2. No hace suposiciones sobre la distribución de los datos.
3. Es adecuado para problemas de clasificación multi clase.
4. Puede funcionar bien en conjuntos de datos pequeños o con pocos atributos.

Desventajas

1. Es computacionalmente costoso para conjuntos de datos grandes, ya que requiere calcular distancias entre todas las instancias.
2. Sensible a la escala de las características, por lo que es importante realizar una normalización adecuada.
3. Puede ser sensible a valores atípicos y ruido en los datos.
4. La elección del valor de k puede ser crucial y puede requerir ajuste manual o validación cruzada para encontrar el mejor valor.

Métricas de Desempeño

Las métricas de desempeño son medidas utilizadas para evaluar el rendimiento de un modelo de clasificación. A partir de la matriz de confusión, se derivan varias métricas que proporcionan información detallada sobre el desempeño del modelo.

Matriz de Confusión

La matriz de confusión es una tabla que se utiliza para evaluar el rendimiento de un modelo de clasificación. Es una representación visual de la cantidad de ejemplos que han sido clasificados correctamente o incorrectamente por el modelo en cada una de las clases.

La matriz de confusión generalmente tiene una estructura de dimensiones nxn, donde n es el número de clases en el problema de clasificación. Cada fila de la matriz representa la clase real de los ejemplos, mientras que cada columna representa la clase predicha por el modelo.

		Valor Predicho	
Valor Real		**Class = Yes**	**Class = No**
	Class = Yes	a (TP)	b (FN)
	Class = No	c (FP)	d (TN)

Matriz de Confusión

La matriz de confusión se divide en cuatro celdas principales:

1. **Verdaderos positivos (True Positives, TP):** Representa la cantidad de ejemplos que pertenecen a una clase específica y han sido clasificados correctamente en esa misma clase por el modelo.

2. **Verdaderos negativos (True Negatives, TN):** Representa la cantidad de ejemplos que no pertenecen a una clase específica y han sido clasificados correctamente como no pertenecientes a esa clase por el modelo.

3. **Falsos positivos (False Positives, FP):** Representa la cantidad de ejemplos que no pertenecen a una clase específica pero han sido clasificados incorrectamente como pertenecientes a esa clase por el modelo.

4. **Falsos negativos (False Negatives, FN):** Representa la cantidad de ejemplos que pertenecen a una clase específica pero han sido clasificados incorrectamente como no pertenecientes a esa clase por el modelo.

La matriz de confusión proporciona información valiosa sobre los errores cometidos por el modelo y permite calcular varias métricas de evaluación del rendimiento, como precisión, sensibilidad, valor F1 y exactitud.

Es importante tener en cuenta que la interpretación de la matriz de confusión depende del problema y del contexto específico. Cada celda de la matriz tiene su propia relevancia en función de los objetivos y las implicaciones de los resultados incorrectos del modelo.

En resumen, la matriz de confusión es una herramienta esencial para evaluar y comprender el rendimiento de un modelo de clasificación, brindando una visión detallada de los aciertos y errores de la clasificación en cada clase.

Accuracy, precision, recall

A partir de la matriz de confusión se pueden calcular múltiples métricas como las siguientes, para esta parte vamos a utilizar los cuadrantes de la matriz de confusión para definir las métricas:

1. **Exactitud (Accuracy):** Es la exactitud global del modelo corresponde a la proporción de datos que fueron correctamente clasificados, independiente de la categoría, por eso se considera la métrica de exactitud global.

$$accuracy = \frac{a+b}{a+b+c+d}$$

2. **Precisión (Precision):** La precisión es la proporción de ejemplos clasificados correctamente como positivos (verdaderos positivos) en relación con todos los ejemplos clasificados como positivos (verdaderos positivos y falsos positivos). Se

puede entender como la capacidad del modelo para identificar correctamente los positivos.

Analogía: Imagina que estás revisando un grupo de personas para identificar a los portadores de una enfermedad. La precisión sería la proporción de personas identificadas correctamente como portadoras de la enfermedad en relación con todas las personas identificadas como portadoras.

$$precision = \frac{a}{a+b}$$

3. **Sensibilidad o Recall (Recall):** La sensibilidad es la proporción de ejemplos clasificados correctamente como positivos (verdaderos positivos) en relación con todos los ejemplos reales positivos (verdaderos positivos y falsos negativos). Se puede entender como la capacidad del modelo para detectar correctamente los positivos.

   Analogía: Siguiendo el ejemplo anterior, la sensibilidad sería la proporción de personas identificadas correctamente como portadoras de la enfermedad en relación con todas las personas que realmente son portadoras.

$$recall = \frac{a}{a+c}$$

4. **Valor F1 (F1 Score):** El valor F1 es una métrica que combina la precisión y la sensibilidad en una sola medida. Es útil cuando se busca un equilibrio entre la precisión y la sensibilidad, ya que tiene en cuenta tanto los falsos positivos como los falsos negativos.

   Analogía: Imagina que estás calificando a los estudiantes en función de su participación en clase y su rendimiento en los exámenes. El valor F1 sería una medida que tiene en cuenta tanto la proporción de estudiantes calificados correctamente como los que asisten a clase (precisión) como la proporción de estudiantes calificados correctamente como los que realmente asisten a clase (sensibilidad).

   La métrica se define en base a recall (r) y precision (p) de la siguiente forma:

$$f1 = \frac{2rp}{r+p}$$

¡Manos a la obra! - Calcula las métricas

1. Utiliza algunos de los ejemplos que hayamos visto anteriormente para calcular las métricas.

Cross Validation

El método de validación cruzada (cross-validation) es una técnica utilizada en el aprendizaje automático para evaluar el rendimiento de un modelo de manera más robusta y evitar problemas de sobreajuste.

El método de validación cruzada se utiliza para estimar cómo se desempeñará un modelo en datos no vistos. En lugar de evaluar el modelo con un solo conjunto de datos de prueba, el método de validación cruzada divide el conjunto de datos disponible en múltiples subconjuntos o "folds". Luego, se realiza el proceso de entrenamiento y evaluación del modelo varias veces, utilizando diferentes combinaciones de subconjuntos de entrenamiento y prueba.

El enfoque más común es la validación cruzada k-fold, que implica dividir los datos en k subconjuntos (folds) del mismo tamaño. A continuación, se realiza el siguiente proceso k veces:

1. Se selecciona uno de los k subconjuntos como el conjunto de prueba y los restantes k-1 subconjuntos se utilizan como conjunto de entrenamiento.

2. El modelo se entrena utilizando los conjuntos de entrenamiento.

3. El modelo se evalúa utilizando el conjunto de prueba y se registra una métrica de rendimiento, como la precisión o el error.

4. Se repiten los pasos anteriores para cada uno de los k subconjuntos.

5. Al final del proceso de validación cruzada k-fold, se obtiene la media de las métricas de rendimiento registradas en cada iteración, lo que proporciona una estimación más confiable del rendimiento del modelo.

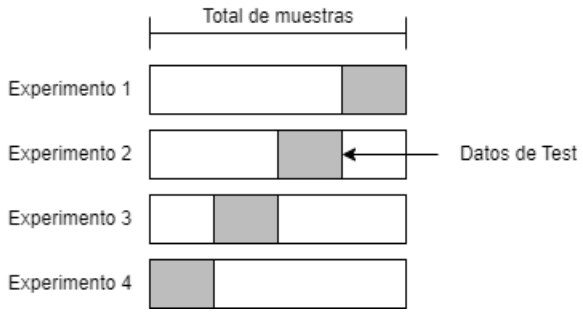


Imagen 2. Validación Cruzada
Fuente: Desafío Latam

El método de validación cruzada es beneficioso por varias razones:

1. Utiliza todo el conjunto de datos para entrenamiento y evaluación, lo que permite aprovechar al máximo los datos disponibles.

2. Proporciona una estimación más robusta del rendimiento del modelo al promediar los resultados de múltiples iteraciones de entrenamiento y evaluación.

3. Ayuda a evitar el sobreajuste, ya que el modelo se evalúa en diferentes conjuntos de prueba y no se ajusta demasiado a un conjunto específico.

El método de validación cruzada es ampliamente utilizado para seleccionar modelos y ajustar hiperparámetros. Al comparar diferentes modelos o configuraciones de hiperparámetros utilizando validación cruzada, es posible identificar aquellos que tienen un mejor rendimiento promedio en diferentes conjuntos de prueba y, por lo tanto, tienen una mayor probabilidad de generalizar bien en datos no vistos.

Es importante tener en cuenta que el método de validación cruzada no garantiza que el modelo se desempeñe igual en datos completamente nuevos, pero proporciona una estimación más sólida y confiable del rendimiento en comparación con una única división de entrenamiento/prueba.

En resumen, el método de validación cruzada es una técnica esencial en el aprendizaje automático para evaluar y comparar modelos de manera robusta. Divide los datos en subconjuntos, entrena y evalúa el modelo en múltiples iteraciones y proporciona una estimación más confiable del rendimiento del modelo en datos no vistos.

Actividad guiada: Eligiendo el hiperparámetro k

Vamos a determinar un posible valor para el parámetro k, utilizando el Método GridSearchCV

```
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# Cargar el conjunto de datos Iris
```

```
data = load_iris()
X = data.data
y = data.target

# Definir el clasificador KNN
knn = KNeighborsClassifier()

# Definir la cuadrícula de hiperparámetros a explorar
param_grid = {'n_neighbors': [3, 5, 7, 10]}

# Realizar la búsqueda de hiperparámetros utilizando validación cruzada
grid_search = GridSearchCV(knn, param_grid, cv=5)
grid_search.fit(X, y)

# Obtener los mejores hiperparámetros y su puntaje
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Mejores hiperparámetros:", best_params)
print("Puntaje del modelo con los mejores hiperparámetros:", best_score)

Output:
Mejores hiperparámetros: {'n_neighbors': 7}
Puntaje del modelo con los mejores hiperparámetros: 0.98
```

Árboles de Decisión

Los Árboles de Decisión corresponden a un algoritmo de aprendizaje supervisado utilizado tanto para problemas de clasificación como de regresión. El objetivo principal del Árbol de Decisión es crear un modelo que prediga el valor de una variable objetivo mediante la toma de decisiones basadas en reglas simples aprendidas a partir de las características del conjunto de datos.

Vamos a ver el siguiente ejemplo, donde tenemos un dataset que tiene pronóstico del tiempo y temperatura en categorías, con las que se quiere predecir si se debe llevar paraguas o no para salir.

A continuación se muestra el pequeño dataset y se muestra un árbol de decisión posible para el ejemplo:

	pronostico	temperatura	paraguas
0	sunny	hot	no
1	sunny	mild	no
2	overcast	hot	yes
3	rainy	mild	yes
4	rainy	cool	yes
5	rainy	cool	no
6	overcast	mild	yes
7	sunny	cool	no

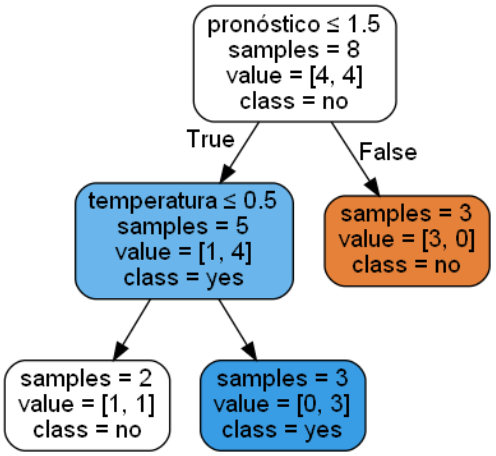


Imagen 2. Árbol de decisión
Fuente: Desafío Latam

Algoritmo

A continuación se muestra el paso a paso del algoritmo del árbol de decisión.

1. Recopilación y preparación de datos:
    a. Obtenemos un conjunto de datos etiquetado que contenga instancias de entrada y sus etiquetas de clase o valores objetivo correspondientes.

2. Construcción del árbol:
    a. Seleccionamos una característica del conjunto de datos que divida los datos de manera óptima según algún criterio, como la ganancia de información o la impureza de Gini. En este caso la fórmula es:

$$gini(p) = 1 - \sum_{i=1}^{c} p_i^2$$

- c es el número de clases en el conjunto de datos
- pi es la proporción de ejemplos en el conjunto de datos que pertenece a la categoría i.

La impureza de Gini toma valores entre 0 y 1, donde 0 indica una pureza perfecta (todos los ejemplos pertenecen a la misma clase) y 1 indica una impureza máxima (los ejemplos se distribuyen de manera uniforme entre todas las clases).

b. Divide el conjunto de datos en subconjuntos más pequeños en función del valor de la característica seleccionada.

c. Repite el proceso de selección de características y división en cada subconjunto de datos hasta que se cumpla un criterio de parada, como

alcanzar un número máximo de niveles en el árbol o cuando todos los subconjuntos de datos sean puros (contengan solo instancias de una clase).

3. Predicción:
    a. Una vez construido el árbol de decisión, se utiliza para realizar predicciones en nuevos datos. Esto se logra recorriendo el árbol de acuerdo con los valores de características de la instancia de prueba hasta llegar a una hoja, donde se toma la decisión final.

Ventajas y desventajas

Los árboles de decisión tienen sus ventajas y desventajas. Aquí hay una descripción de algunas de ellas, junto con posibles soluciones a las desventajas:

Ventajas

1. Fácil de entender e interpretar. Las decisiones tomadas por el árbol pueden ser visualizadas y explicadas fácilmente.
2. Capacidad para manejar tanto datos numéricos como categóricos.
3. No requiere supuestos sobre la distribución de los datos o la linealidad de las relaciones.
4. Puede capturar relaciones no lineales y características interactivas en los datos.

Desventajas

1. Puede ser propenso al sobreajuste si no se controla adecuadamente mediante la poda, la limitación de la profundidad o el ajuste de otros hiperparámetros.
2. Puede ser sensible a pequeñas variaciones en los datos de entrenamiento.
3. Los árboles de decisión simples pueden no capturar relaciones complejas en los datos.
4. Puede haber problemas de sesgo si ciertas clases o características dominan los datos de entrenamiento.

Espero que esta explicación, ejemplo de implementación, hiperparámetros y ventajas y desventajas del algoritmo de Árbol de Decisión te sean útiles para comprender su funcionamiento y su aplicación en problemas de aprendizaje supervisado.

Hiperparámetros

Los árboles de decisión tienen múltiples hiperparametros que pueden ajustarse, pero la mayoría apunta hacia la poda de las ramas, que tanto puedo dejar crecer el árbol, dentro de estos los más utilizados son los siguientes:

1. **Profundidad máxima (max_depth):** Este hiperparámetro controla la profundidad máxima del árbol. Un árbol más profundo puede capturar relaciones más complejas en los datos, pero también tiene más probabilidades de sobreajustarse. A medida

que aumenta la profundidad, el árbol se vuelve más complejo y puede aprender patrones más detallados.

2. **Número mínimo de muestras en una hoja (min_samples_leaf):** Este hiperparámetro establece el número mínimo de muestras requeridas en una hoja del árbol. Controla la cantidad mínima de muestras necesarias para que se forme una hoja. Un valor más alto evita particiones excesivas y puede ayudar a evitar el sobreajuste.

3. **Número mínimo de muestras requeridas para dividir un nodo interno (min_samples_split):** Este hiperparámetro especifica el número mínimo de muestras necesarias en un nodo para que se considere una división. Si la cantidad de muestras en un nodo es menor que el valor establecido, no se realizará una división adicional y se formará una hoja.

Naive Bayes

El algoritmo Naive Bayes se basa en el teorema de Bayes, el cual nos permite calcular la probabilidad de que un evento ocurra dado cierta evidencia. En el contexto de la clasificación, el algoritmo Naive Bayes nos ayuda a estimar la probabilidad de que un ejemplo pertenezca a una clase determinada dado un conjunto de características observadas.

La intuición clave del algoritmo Naive Bayes radica en la suposición "ingenua" de independencia condicional entre las características dadas las clases objetivo. Esto significa que asumimos que las características son independientes entre sí, es decir, que la presencia o ausencia de una característica no está relacionada con la presencia o ausencia de otras características. Aunque esta suposición es "ingenua" y puede no ser cierta en todos los casos, el algoritmo Naive Bayes ha demostrado ser efectivo en muchos escenarios.

La idea principal es que, dado un ejemplo con características observadas, el algoritmo Naive Bayes calcula la probabilidad de que ese ejemplo pertenezca a una clase específica utilizando la regla de Bayes. La regla de Bayes combina la probabilidad a priori de la clase con la probabilidad de observar las características dadas la clase, y luego normaliza el resultado dividiéndolo por la probabilidad marginal de las características.

El algoritmo Naive Bayes asigna el ejemplo a la clase con la probabilidad posterior más alta. En otras palabras, elige la clase que es más probable dada la evidencia observada.

Esta intuición se basa en la idea de que si podemos estimar las probabilidades de las clases y las probabilidades de las características dadas las clases, entonces podemos utilizar el teorema de Bayes para obtener la probabilidad de las clases dadas las características. Al hacer la suposición de independencia condicional entre las características, simplificamos el cálculo de estas probabilidades.

En resumen, el algoritmo Naive Bayes utiliza el teorema de Bayes y la suposición de independencia condicional entre características para clasificar ejemplos basado en la probabilidad de pertenecer a diferentes clases. A pesar de su simplicidad y su suposición "ingenua", el algoritmo Naive Bayes ha demostrado ser efectivo en una amplia gama de aplicaciones y es ampliamente utilizado en problemas de clasificación.

Teorema de Bayes

Recordemos que el teorema de Bayes es una fórmula matemática que describe cómo se actualiza la probabilidad de un evento dado un nuevo conocimiento o evidencia. Se basa en la idea de que la probabilidad condicional de un evento A dado un evento B se puede calcular utilizando la probabilidad del evento B dado el evento A, junto con la probabilidad a priori de A y B.

La fórmula del teorema de Bayes es la siguiente:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Donde:
P(A | B) es la probabilidad del evento A dado el evento B (probabilidad condicional).
P(B | A) es la probabilidad del evento B dado el evento A.
P(A) y P(B) son las probabilidades a priori de los eventos A y B, respectivamente.

Algoritmo

El algoritmo Naive Bayes es un algoritmo de clasificación supervisada que se basa en el teorema de Bayes. Utiliza la suposición "ingenua" (naive) de independencia condicional entre las características (variables predictoras) dadas las clases objetivo.

El algoritmo Naive Bayes calcula la probabilidad de que un ejemplo pertenezca a una clase determinada utilizando la fórmula del teorema de Bayes:

$$P(C|X) = \frac{P(X|C)P(C)}{P(X)}$$

Donde:

P(C | X) es la probabilidad de la clase C dada las características X (probabilidad posterior).
P(X | C) es la probabilidad de las características X dada la clase C (probabilidad de verosimilitud).
P(C) es la probabilidad a priori de la clase C.
P(X) es la probabilidad marginal de las características X.

La suposición "ingenua" del algoritmo Naive Bayes radica en considerar que las características son independientes entre sí, lo que simplifica el cálculo de las probabilidades de verosimilitud. Aunque esta suposición puede no ser cierta en la realidad, el algoritmo Naive Bayes ha demostrado ser efectivo en muchas aplicaciones prácticas.

Ventajas y desventajas

Naive Bayes tienen sus ventajas y desventajas. Aquí hay una descripción de algunas de ellas, junto con posibles soluciones a las desventajas:

Ventajas

1. Es fácil de entender e implementar.
2. Es eficiente en términos de tiempo de entrenamiento y clasificación.
3. Puede manejar eficazmente grandes conjuntos de datos.
4. Es resistente al ruido y puede manejar características irrelevantes.
5. Puede manejar datos categóricos y numéricos.

Desventajas

1. La suposición de independencia condicional entre las características puede no ser cierta en todos los casos, lo que puede afectar la precisión del modelo.
2. Puede tener dificultades para manejar clases poco representadas en los datos de entrenamiento.
3. No captura relaciones complejas entre las características.

A pesar de estas limitaciones, el algoritmo Naive Bayes es ampliamente utilizado en aplicaciones de clasificación, especialmente cuando los datos tienen muchas características y se requiere un rendimiento rápido y eficiente.

¡Manos a la obra! - Calidad de los vinos

Retoma la resolución del ejercicio visto en clases, para predecir la calidad de los vinos. Considera los archivos 02 - Calidad de los vinos.iypnb y winequality.csv

Preguntas de proceso

Reflexiona:

- ¿Cuáles son los algoritmos de clasificación más comunes y cómo funcionan?
- ¿Qué métricas se utilizan para evaluar el rendimiento de los modelos de clasificación y cómo se interpretan?
- ¿Cómo se selecciona el algoritmo de clasificación más adecuado para un problema específico?
- ¿Cómo se pueden ajustar los hiperparámetros de un algoritmo de clasificación y qué impacto tienen en el rendimiento del modelo?
- ¿Cuál es la diferencia entre validación cruzada, conjunto de entrenamiento y conjunto de prueba en el contexto de la evaluación del rendimiento de los modelos de clasificación?
- ¿Cuáles son las ventajas y desventajas de los diferentes algoritmos de clasificación?

Referencias bibliográficas

1. Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.
2. Python Data Science Handbook: https://jakevdp.github.io/PythonDataScienceHandbook
3. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
4. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

¡Continúa aprendiendo y practicando!