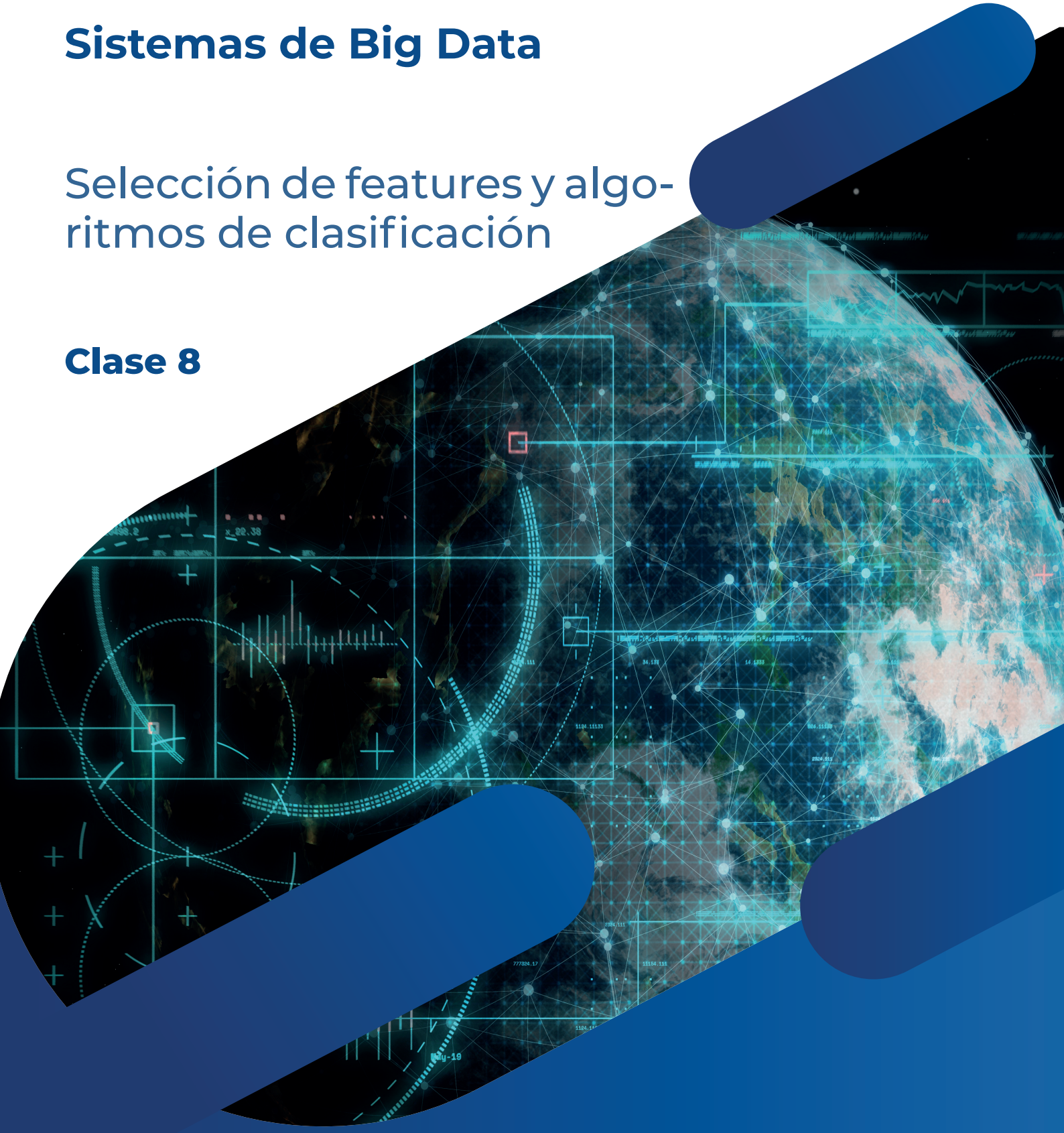
Sistemas de Big Data

Selección de features y algoritmos de clasificación

Clase 8



MAESTRÍA EN
SISTEMAS DE INFORMACIÓN
Mención Data Science

La excelencia no se improvisa

1. INTRODUCCIÓN DE LA CLASE

La selección de *features* constituye un pilar fundamental en el desarrollo de modelos de aprendizaje automático, ya que permite identificar y transformar las variables más relevantes para mejorar la precisión y reducir la complejidad del modelo. En este contexto, la **Feature Transformation** se encarga de modificar y combinar características originales para revelar patrones subyacentes, mientras que la **Reducción de Dimensionalidad** se centra en eliminar redundancias y minimizar el ruido presente en los datos, facilitando la interpretación y optimización del proceso de entrenamiento (Guyon & Elisseeff, 2003).

Por otro lado, los algoritmos de clasificación son herramientas esenciales para asignar categorías a nuevas observaciones, basándose en el conocimiento adquirido a partir de datos históricos. Entre ellos, el método de **k–vecinos más cercanos** se destaca por su simplicidad y efectividad en problemas de baja dimensionalidad, mientras que los á**rboles de decisión** ofrecen una representación gráfica y fácil de interpretar de las reglas de decisión. Además, el algoritmo **Naive Bayes** utiliza principios probabilísticos para realizar predicciones de manera eficiente, incluso con conjuntos de datos de alta dimensionalidad (Hastie, Tibshirani, & Friedman, 2009).

RDA 3: Evaluar los fundamentos del Big Data para el análisis de grandes volúmenes de datos

Clase 8. Selección de *features* y algoritmos de clasificación

8.1. Feature Selection

Los objetivos de esta sección son:

- Explicar lo que implica la selección de *features*
- Discutir el objetivo de la selección de *features*
- Listar tres enfoques para la selección de *features*

La selección de *features* se refiere a escoger el conjunto de *features* apropiados para el **análisis subsecuente**. El objetivo de la selección de *features* es tener la menor cantidad de *features* que mejor captura las características del problema que se está encarando. Mientras más pequeño sea el número de *features* a usar, más simple será el análisis. Por supuesto que el conjunto de *features* que se va a usar debe incluir todos los *features* relevantes para el problema. Así que tiene que haber un balance entre la **expresividad** y la **cantidad de *features* por usar**. Existen varios métodos para la selección de *features*:

- Añadir features
- Eliminación de features
- Recodificación de features
- Combinación de features

Nuevos *features* pueden derivarse de algunos existentes (Figura 1). Por ejemplo, un nuevo *feature* para especificar si el estudiante vive o no en el campus esto puede aportar mucha información para aplicaciones a la universidad.

Los *features* también pueden ser **eliminados.** Cuando existe una alta correlación entre un par de *features* se puede eliminar uno de ellos. Por ejemplo, el precio de venta y el monto de impuestos de un producto están altamente correlacionados; en este caso, conviene eliminar uno de ellas, ya que el otro brindaría información redundante y haría más complejo el análisis.

Name	State
Angela	AK
Sidney	CA
Ratan	WA
Kiril	OR
Zhou	CA

Name	State	*In-State*
Angela	AK	***F***
Sidney	CA	***T***
Ratan	WA	***F***
Kiril	OR	***F***
Zhou	CA	***T***

Figura 1. *Añadir features.*

También se pueden **eliminar *features*** con alta cantidad de valores duplicados, dado que al tener pocos datos no implicaría una pérdida de información significativa. *Features* irrelevantes tales como ID, row number, etc.

Combinar *features.* Se pueden crear variables a partir de otras y así brindar información que por sí solas no brindan. Por ejemplo, en la Figura 2, el índice de masa corporal (IMC / BMI, por sus siglas en inglés) se puede calcular a partir del peso y la altura de una persona. Ese índice puede indicar si la persona tiene sobrepeso, peso regular, etc.

Name	Height	Weight
Angela	1.8	68
Sidney	1.5	70
Ratan	2.0	84

Name	Height	Weight	*BMI*
Angela	180	68	***21***
Sidney	153	70	***30***
Ratan	204	84	***20***

Figura 2. *Combinar features.*

Recodificación de *features.* Esto implica cambiar el formato de la variable en función del aporte que brinde a la aplicación. Por ejemplo, en el caso de *marketing* es posible que se necesite la edad del cliente en forma de categoría; por ejemplo: adolescente, joven adulto, adulto y sénior. Otro ejemplo puede ser cuando se necesita identificar si el precio de venta de un producto es superior o no a la media; en ese caso, estamos hablando de una categoría binaria (1 si es mayor y 0 si no)

Otro caso en la recodificación de *features* es dividir el *features* en varias en función de sus valores por ejemplo el *feature* dirección se puede dividir en dirección, estado y código postal, tal cómo se muestra la Figura 3.

Address
430 Park Drive, CA, 97283
7800 W. View Street, FL, 34642
1243 Mountain Ave., CO, 80334

Address	State	Zip
430 Park Drive	CA	97283
7800 W. View Street	FL	34642
1243 Mountain Ave.	CO	80334

Figura 3. Dividir f*eatures*

8.1.1. Feature Transformation

Los objetivos de esta sección son:

- Listar tres operaciones para ***feature transformation***
- Discutir por qué es importante el **escalamiento (*scaling*)**

La **transformación** implica crear *features* en función de *features* iniciales, con el objetivo de hacer el proceso más fácil y adecuado para el análisis subsecuente. Una **transformación** muy común es ***scaling,*** que implica cambiar los rangos de valores para que tenga otras unidades de medida.

Por **ejemplo**, si se trabaja con el peso en kilos y la estatura en metros, la magnitud de la primera variable será mucho mayor que la de la segunda, para que ambas estén en los mismos rangos de valores posibles, se las puede escalar en valores entre 0 y 1 (Figura 4). Otro ejemplo puede ser cuando tienes la cantidad de años de trabajo y los ingresos medidos en dólares, en este caso también podemos **escalar** ambas **variables** en valores entre **0 y 1. Esta transformación es conocida también como *min max value*.**

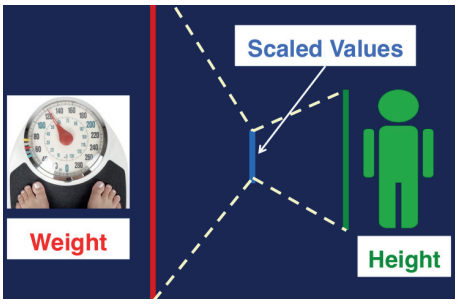


Figura 4. *Scaling.*

También se pueden transformar los *features* usando la **normalización zero** o estandarización. El proceso es calcular el promedio y el desvío o desviación estándar y por cada elemento de la variable restar el promedio y dividirlo por el desvío estándar. Esto generará una nueva variable con promedio 0 y desvío estándar 1. Este método es cuando se tienen ***ouliers*** que pueden afectar la transformación **0 y 1.**

Otra **transformación** de datos es ***filtering*** (Figura 5), que se encarga de eliminar ruido, esto se puede aplicar en señales de audio y en imágenes. **Agregación** es otra transformación que permite o **sumarizar** los datos en función de cierta magnitud; por ejemplo, si tenemos la información sobre el promedio de la velocidad del viento o podemos generar en intervalos de 10 o de 60 segundos.

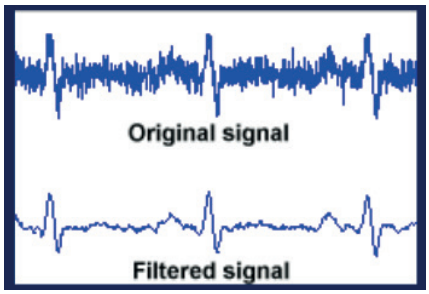


Figura 5. *Filtering.*

8.1.2. Reducción de dimensionalidad

Los objetivos de esta sección son los siguientes:

- Explicar en qué consiste la reducción de dimensionalidad
- Discutir los beneficios de la reducción de dimensiones
- Describir cómo ***Principal Component Analysis*** **(PCA)** transforma los datos

La cantidad de ***features*** de tu *dataset* define la dimensionalidad de tus datos. Si tienes dos variables, tu *dataset* es de dos dimensiones; si tiene tres, es de tres dimensiones y así, sucesivamente. Al aumentar la dimensionalidad, el espacio en que tu problema se desenvuelve también se incrementa, requiriendo significativamente más instancias. Si el espacio crece, crece también la cantidad de datos 'esparsos'.

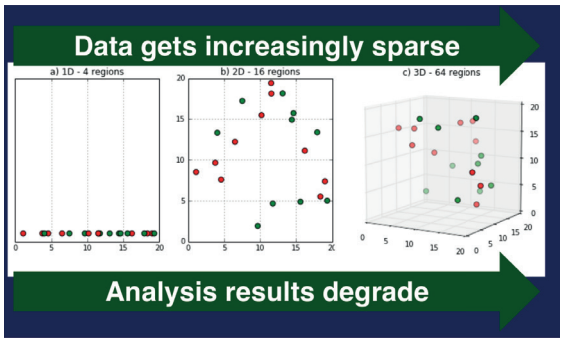


Figura 6. *Maldición de la multidimensionalidad.* Obtenido de: Ng, A., Shyu, E., Bagul, A., & Ladwig, G. (s.f.). Machine Learning Specialization. DeepLearning.AI y Stanford Online

Eso se puede ver claramente en la Figura 6. Cuando las dimensiones aumentan también aumenta la cantidad de regiones de manera exponencial y los datos se hacen 'esparsos'. Adicionalmente, cuando hay muchas variables, ciertos cálculos son más **difíciles** de realizar dado que la distancia entre los puntos es mucho mayor y la comparación entre elementos se vuelva más difícil. Estos problemas se los conoce como la ***maldición de la multidimensionalidad.***

Maldición de la multidimensionalidad [http://medium.com/vlgiitr/the-curse-of-dimensionality-15f950e519d2]

Para evitar la maldición de la multidimensionalidad, lo que se necesita es reducir la cantidad de dimensiones. Para ello se aplican algoritmos que permitan reducir la cantidad de dimensiones, garantizando la representatividad de estas dimensiones respecto del *dataset* original. Un primer acercamiento a la **reducción de dimensiones** (Figura 7) es eliminar un *feature* cuando existe un par que se encuentra altamente correlacionado.

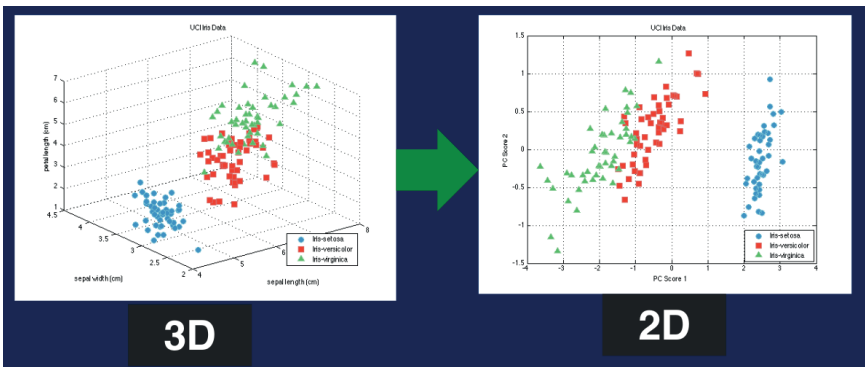


Figura 7. *Reducción de dimensionalidad*. Obtenido de: Ng, A., Shyu, E., Bagul, A., & Ladwig, G. (s.f.). Machine Learning Specialization. DeepLearning.AI y Stanford Online

Otro acercamiento a la reducción de dimensiones es determinar matemáticamente cuáles son las variables más importantes, conservar esas y eliminar el resto. Esto implica eliminar las variables irrelevantes, haciendo que los **análisis subsiguientes** sean más simples.

Una de las técnicas más usadas para la **reducción** de **dimensiones** es *Principal Component Analysis* (PCA). El objetivo de PCA es reducir un espacio dimensional alto a uno más bajo conservando la mayor variabilidad posible. En las siguientes líneas se muestra la idea principal de *PCA. PCA convierte un conjunto de* ***variables correlacionadas*** *en otro conjunto de* ***variables no correlacionas,*** *conservando la mayor variabilidad posible. Estas nuevas variables se las conoce como componentes principales.*

Algunos de los puntos claves a considerar en el PCA son:

- PCA busca un nuevo **sistema** de **coordenadas** de tal forma que:
  - **PC1** captura la mayor cantidad de varianza

    - **PC2** captura la segunda proporción de varianza, etc.
- Las primeras componentes principales capturan la mayor cantidad de varianza. Definiendo un espacio dimensional mucho menor para tus datos.

Algo importante que se debe tener en cuenta es la **interpretabilidad.** Es más complejo interpretar las componentes principales que usar las variables originales, dado que las componentes principales no tienen una **definición natural**.

8.2. Algoritmos de clasificación

Los objetivos de esta sección son:

- **Describir** la finalidad de los algoritmos de clasificación.
- **Nombrar** algunos algoritmos comunes de clasificación.

La tarea fundamental de un algoritmo de clasificación es predecir una **categoría** con base a *input variables.* Y el **objetivo** es que las predicciones sean lo más parecidas posible a los valores reales. Eso se consigue en la fase de entrenamiento, que no es más que ajustar los parámetros del modelo en función de los *inputs* de entrada. Existen múltiples algoritmos de clasificación, entre ellos podemos citar:

- *k- vecinos más cercanos*
- *árboles de decisión*
- *Naive Bayes*

8.2.1. k – vecinos más cercanos

Es un algoritmo de *machine learning* que puede ser usado tanto para clasificación como para regresión. Para modelos de clasificación, el algoritmo **k-vecinos más cercanos** funciona de la siguiente manera:

Dado un conjunto de entrenamiento X_train con *labels* y_train y dada una instancia de datos a ser clasificada x_test, los pasos son los siguientes:

1. Buscar las instancias más similares (a las cuales las llamaremos X_NN) a x_test que se encuentran en X_train
2. Obtener los *labels* y_NN para cada una de las instancias en X_NN
3. Predecir el *label* para x_test combinando los *labels* y_NN/e.g. por mayoría de votos

La Figura 8 muestra un espacio de dos dimensiones para un problema de clasificación. Si tenemos un registro nuevo, por ejemplo las 'x', en el gráfico se le asignará la clasificación (0 o 1) de la instancia que se encuentre más próxima a ellos. En este caso, ambos puntos serán clasificados como clase 1. Este concepto aplica tanto para modelos de clasificación como para modelos de regresión.

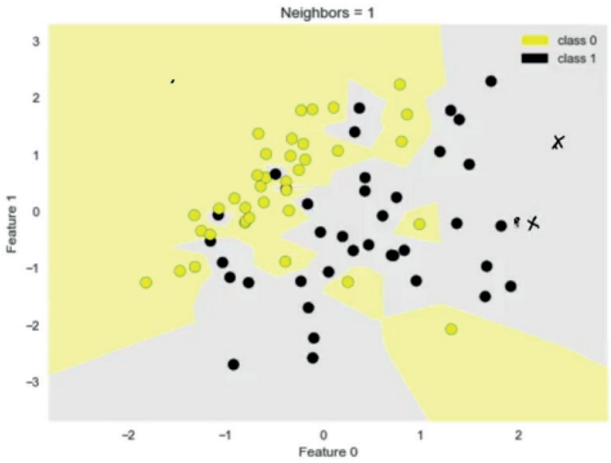


Figura 8. *K – vecinos más cercanos ejemplo básico.*

La Figura 9 muestra la estrategia para evitar la creación de modelos muy complejos:

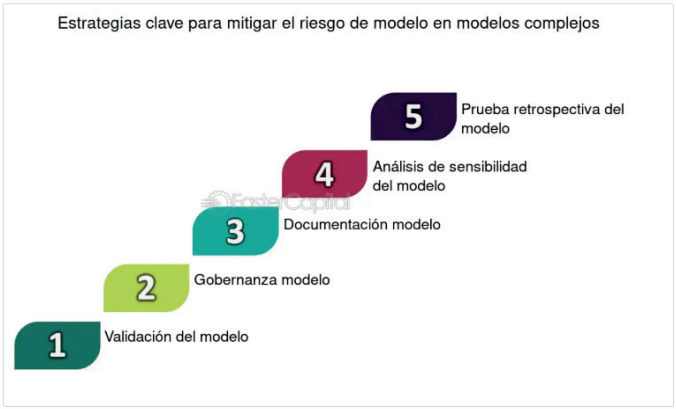


Figura 9. *Estrategias para evitar modelos complejos.* https://fastercapital.com/es/contenido/Complejidad-del-modelo--gestion-del-riesgo-del-modelo-en-entornos-de-modelado-complejos.html

Algunos parámetros importantes para considerar tanto para modelos de clasificación como para regresión, usando el algoritmo k – vecinos más cercanos:

Complejidad de los modelos

n_neighbors: número de k_vecinos más cercanos a considerar por defecto en la implementación de Python se usa el valor de 5.

Model Fitting

metric: función de distancia entre los puntos de datos, por defecto en la implementación de Python se usa la distancia de Minkowski.

K_vecinos más cercanos, aplicación[https://repositorio.utdt.edu/handle/20.500.13098/11781]

8.2.2. Arboles de decisión

Los **árboles de decisión** son uno de los algoritmos más **utilizados** en *machine learning* debido a su **simplicidad**, **interpretabilidad** y **capacidad** para manejar tanto **problemas** de **clasificación** como de **regresión**. Un árbol de decisión es un modelo predictivo, que toma decisiones **basadas en características de los datos**, representadas en **nodos de un árbol jerárquico**. Cada nodo interno realiza una prueba sobre una característica, y cada hoja del árbol representa una etiqueta de clase (para clasificación) o un valor (para regresión). Esta estructura permite representar relaciones complejas de manera comprensible (Breiman et al., 1986).

La Figura 10 brinda información sobre las ramas del árbol de decisión. La lista *value* indica el número de ejemplos de cada clase que quedaron en la hoja, durante el proceso de entrenamiento. Dado que el *dataset* iris (un tipo de flor) tiene 3 clases hay 3 elementos en la lista. Esta rama tiene 37 ejemplos de setosa, 0 de versicolor y 0 de virginica.

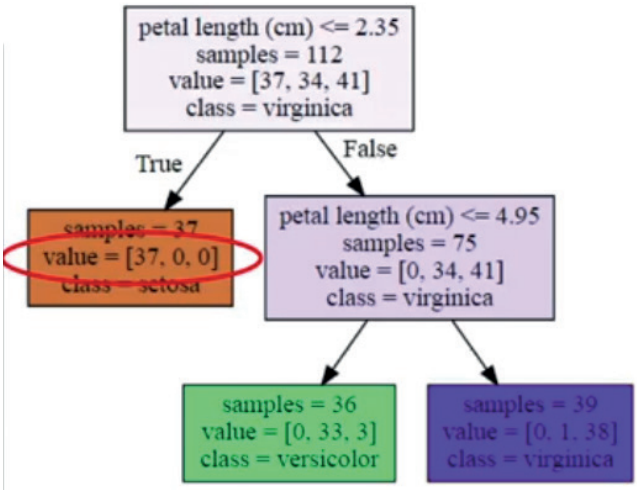


Figura 10. *Informatividad de las separaciones del árbol de decisión.*

Funcionamiento de los árboles de decisión

El funcionamiento de los árboles de decisión se basa en un proceso de **particionamiento recursivo del conjunto de datos**. Este proceso busca **dividir** el conjunto de datos de manera que las **particiones** resultantes sean lo más **homogéneas** posible, en términos de la variable objetivo. Para lograrlo, se emplean medidas como la ***entropía*** y el ***índice de Gini*** para elegir el mejor atributo en cada nodo (Quinlan, 1986). La Figura 11 presenta un ejemplo básico de implementación de un árbol de decisión utilizando la biblioteca *scikit-learn* de Python, para un problema de clasificación con el conjunto de datos *Iris*:

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Cargar el conjunto de datos Iris
iris = load_iris()
X = iris.data
y = iris.target

# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Crear un modelo de árbol de decisión
clf = DecisionTreeClassifier(random_state=42)

# Entrenar el modelo
clf.fit(X_train, y_train)

# Hacer predicciones
y_pred = clf.predict(X_test)

# Evaluar el modelo
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

Figura 11. *Implementación de árbol de decisión con s-klearn.*

En este ejemplo, un árbol de decisión se entrena con el conjunto de datos *Iris*, y se **evalúa** el **rendimiento del modelo** mediante la **métrica de precisión**. El modelo **clasifica** las **flores** del conjunto de datos en **tres categorías, basadas en sus características morfológicas.**

Ventajes y desventajas de los árboles de decisión

Los **árboles de decisión** tienen varias ventajas, entre las cuales se destaca su **capacidad** para ser **interpretados fácilmente por los humanos**. Esto es crucial en aplicaciones donde la **explicabilidad** es importante, como en el sector financiero o en la atención médica (Breiman et al., 1986). Sin embargo, una de sus principales **desventajas** es que pueden **sobreajustarse** si no se controlan adecuadamente. El sobreajuste ocurre cuando el árbol se adapta demasiado a los datos de entrenamiento, perdiendo capacidad de generalización para datos nuevos (Breiman, 2001). Para mitigar este problema, es común emplear **técnicas de poda**, que eliminan ramas del árbol que no aportan valor significativo.

Técnicas de poda y ensamble de modelos

Una técnica importante para mejorar los árboles de decisión es la **poda**, que consiste en **eliminar nodos innecesarios o ramas que no contribuyen al rendimiento del modelo**. La **poda** ayuda a **reducir el sobreajuste** y **mejora la capacidad de generalización** del modelo. A continuación, la Figura 12 presenta un ejemplo de implementación de poda utilizando el parámetro **max_depth** de *scikit-learn,* para limitar la profundidad del árbol y **evitar** el **sobreajuste.**

```python
# Crear un modelo de árbol de decisión con poda (limitando la profundidad)
clf_poda = DecisionTreeClassifier(max_depth=3, random_state=42)

# Entrenar el modelo
clf_poda.fit(X_train, y_train)

# Hacer predicciones
y_pred_poda = clf_poda.predict(X_test)

# Evaluar el modelo
print(f"Accuracy con poda: {accuracy_score(y_test, y_pred_poda)}")
```

Figura 12. *Poda de árbol de decisión con s-klearn.*

En este ejemplo, se limita la **profundidad máxima del árbol a 3**, lo que evita que el modelo crezca excesivamente y se sobreajuste a los datos de entrenamiento.

Otras técnicas avanzadas que pueden mejorar el rendimiento de los árboles de decisión incluyen ***Random Forests*** **y** ***Gradient Boosting***. Estos enfoques construyen **múltiples árboles de decisión** y combinan sus resultados para mejorar la precisión general del modelo. Los ***Random Forests*** funcionan entrenando **muchos árboles sobre subconjuntos aleatorios de los datos**, lo que **reduce** la **varianza** y **mejora** la **robustez** del modelo. Por otro lado, ***Gradient Boosting*** **construye árboles secuenciales**, donde **cada árbol corrige los errores del anterior**, lo que puede llevar a un rendimiento significativamente superior en ciertos contextos (Breiman, 2001).

Ventajas	**Desventajas**
Fácil de visualizar e implementar	Puede presentar problemas de *overfiting*
No se necesita normalización de datos	Usualmente se necesita ensamblar árboles para obtener un mejor *performance*

8.2.3. Naïve Bayes

Navïe Bayes Classifier es un algoritmo simple de la familia de los clasificadores basados en probabilidad. Se llama '*Navie' (ingenuo)* porque asume que, dado una clase, los *features* son condicionalmente independientes. En otras palabras, *Naïve Bayes Classifier* asume que para todas las instancias de una determinada clase, los *features* no tienen correlación entre ellos. Entre algunas de sus características principales podemos citar:

- Son altamente eficientes en tanto en capacidad de aprendizaje como en predicción.
- Su capacidad de generalización tiende a ser baja en comparación con algoritmos de aprendizaje más sofisticados.
- Pueden ser muy buenos para determinadas tareas.

Tipos de *Naïve Bayes*

El **Naïve Bayes Classifier** es un algoritmo de clasificación basado en el teorema de Bayes, que asume independencia condicional entre las características del conjunto de datos. Existen tres variantes principales de este clasificador: **Bernoulli Naïve Bayes**, **Multinomial Naïve Bayes** y **Gaussian Naïve Bayes**, cada una diseñada para distintos tipos de datos y aplicaciones (Manning, Raghavan & Schütze, 2008).

El **Bernoulli Naïve Bayes** se emplea en datos binarios, donde cada característica puede tomar valores de 0 o 1. Es comúnmente utilizado en clasificación de texto con modelos de presencia/ausencia de palabras, como en el filtrado de spam (McCallum & Nigam, 1998). En contraste, el **Multinomial Naïve Bayes** maneja datos discretos y cuenta la frecuencia de ocurrencia de palabras en documentos, siendo ampliamente aplicado en problemas de minería de texto y categorización de documentos (Rennie, Shih, Teevan & Karger, 2003). Finalmente, el **Gaussian Naïve Bayes** es adecuado para datos continuos, asumiendo que las características siguen una distribución normal. Es usado en reconocimiento de patrones y detección de anomalías (Murphy, 2012).

El **decision boundary** (frontera de decisión) en el **clasificador Gaussian Naïve Bayes** es la curva o superficie que separa las regiones del espacio de características en función de la probabilidad de pertenencia a cada clase, asumiendo que las características siguen una distribución normal (Gaussian) dentro de cada clase (Murphy, 2012). A continuación, se muestra una ilustración de cómo sería el *decision boundary* para un problema tipo de clasificación.

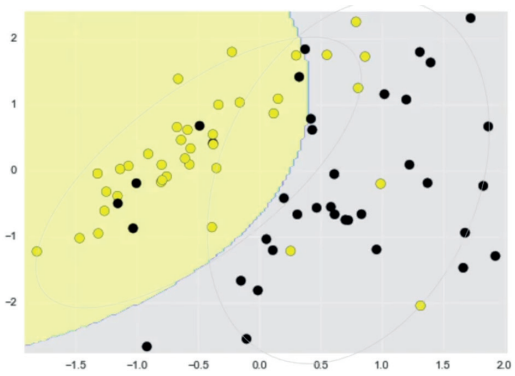


Figura 14. *Decision Boundary – Gaussian Naïve Bayes Classifier.*

Ventajas y desventajas de usar *Naïve Bayes Classifier*

Ventajas	Desventajas
Fácil de entender	Asumir que todos los *features* son condicionalmente independientes; dada una clase determinada, no es algo realista
Estimación de parámetros fácil y eficiente	Otros clasificadores usualmente tienen mejor poder de generalización
Trabaja bien con datos que tienen alta dimensionalidad	Su confianza estimada para las predicciones no es muy alta
Usualmente se usa como *baseline* para compararlo con modelos más sofisticados	

Referencias citadas en la Clase 8

- Guyon, I., & Elisseeff, A. (2003). *An introduction to variable and feature selection*. Journal of Machine Learning Research, 3, 1157-1182.
- Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.
- Ribeiro, M.T., Singh, S., & Guestrin, C. (2016). Why should I trust you? Explaining the predictions of any classifier. En *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp.1135-1144).
- Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1986). *Classification and regression trees*. Wadsworth & Brooks/Cole Advanced Books & Software.
- Quinlan, J.R. (1986). *Induction of decision trees*. Machine Learning, 1(1), 81-106.

Definición de los términos citados en la Clase 8

Datos 'esparsos'.

Los **datos 'esparsos' o dispersos** son aquellos en los que la mayoría de los valores en un conjunto de datos son ceros o valores nulos; es decir, la gran mayoría de las celdas contienen información vacía o irrelevante. Este tipo de datos es común en áreas como el análisis de texto, donde las matrices de términos frecuentemente tienen muchas entradas vacías, debido a que muchas palabras no aparecen en todos los documentos, o en sistemas de recomendación; donde los usuarios no han calificado todos los productos.

La característica clave de los datos dispersos es que contienen una baja proporción de valores no nulos, lo que permite aplicar técnicas de almacenamiento y procesamiento especializadas para manejar eficientemente este tipo de información. Al hacerlo, se optimiza el uso de memoria y se mejora el rendimiento computacional, ya que solo se almacenan los valores relevantes, reduciendo el espacio nece-

sario para su almacenamiento y procesamiento.

Maldición de la multidimensionalidad.
La **maldición de la multidimensionalidad** se refiere a los desafíos y problemas que surgen cuando se trabaja con datos en espacios de alta dimensión. A medida que aumenta el número de características (dimensiones) de un conjunto de datos, el volumen del espacio de búsqueda crece exponencialmente, lo que hace que los datos se distribuyan de manera muy dispersa. Esto provoca que las distancias entre puntos de datos se vuelvan más similares, dificultando tareas como la clasificación, el agrupamiento o la predicción. Además, la mayor cantidad de dimensiones requiere más datos para entrenar modelos precisos, lo que incrementa la necesidad de almacenamiento y computación. En resumen, la maldición de la multidimensionalidad se refiere a la complejidad y la disminución de la efectividad de los algoritmos a medida que se añaden más dimensiones a los datos. Para mitigar este problema, se suelen utilizar técnicas como la reducción de dimensionalidad, que buscan reducir el número de características manteniendo la mayor cantidad de información relevante posible.

Profundización Clase 8

Infografía sobre k – vecinos más cercanos:

Descripción:
El algoritmo KNN clasifica un punto de datos según la mayoría de los 'k_vecinos más cercanos'. Utiliza una medida de distancia (como la distancia euclidiana) para encontrar los puntos más cercanos y asignar la clase mayoritaria de esos puntos.

Características:

- **No paramétrico.** No hace suposiciones sobre la distribución de los datos.
- **Basado en instancias.** No construye un modelo explícito, simplemente guarda los datos.
- **Requiere parámetro 'k'.** 'k' representa el número de vecinos cercanos por considerar.

Ventajas:

- Fácil de entender e implementar
- Funciona bien con datos no lineales
- No necesita entrenamiento, solo almacenamiento de datos

Desventajas:

- Computacionalmente costoso en grandes conjuntos de datos.
- Sensible a la elección de 'k' y a la escala de las características

Infografía sobre árboles de decisión:

Descripción:
Un árbol de decisión es un modelo de clasificación que divide los datos en ramas a través de preguntas binarias (sí/no); sobre las características de los datos. Cada nodo del árbol representa una característica, y las hojas representan la clase final.

Características:

- **División jerárquica.** Cada nodo divide el espacio de datos en función de una característica específica.
- **Cualitativo y cuantitativo.** Puede manejar tanto características numéricas como categóricas.
- **Fácil de interpretar.** Su estructura en forma de árbol facilita la visualización y comprensión.

Ventajas:

- Interpretabilidad clara (fácil de visualizar)
- No necesita normalización de datos
- Puede manejar tanto variables continuas como discretas

Desventajas:

- Propenso al sobreajuste si no se poda adecuadamente
- Puede ser inestable (pequeñas variaciones en los datos pueden cambiar el modelo)

PUCE Virtual

La excelencia no se improvisa

síguenos