PUCE Virtual
EDUCACIÓN FLEXIBLE

Aprendizaje automático (Machine Learning)

Aprendizaje supervisado segunda parte

Clase 6

MAESTRÍA EN
SISTEMAS DE INFORMACIÓN
Mención Data Science

La excelencia no se improvisa

INTRODUCCIÓN

El **aprendizaje supervisado** es una técnica dentro del *machine learning* en la que un modelo aprende a asociar entradas con salidas conocidas, a partir de un conjunto etiquetado de datos. Entre los algoritmos más representativos está **Naïve Bayes**, un clasificador probabilístico basado en el teorema de Bayes, que asume independencia entre las características; es útil en problemas como clasificación de texto y detección de spam (Manning, Raghavan & Schütze, 2008). Por otro lado, **Random Forest** es un modelo basado en ensamblado de múltiples árboles de decisión, reduciendo la varianza y mejorando la capacidad de generalización en comparación con un solo árbol (Breiman, 2001). Su versatilidad lo hace eficaz tanto en clasificación como en regresión, siendo utilizado en aplicaciones como detección de fraudes y predicción de enfermedades.

El avance del aprendizaje supervisado ha dado lugar a modelos más complejos, como las **redes neuronales artificiales**, que se inspiran en la estructura del cerebro para aprender patrones en los datos (Goodfellow, Bengio & Courville, 2016). Con el desarrollo del **Deep Learning**, se han diseñado redes neuronales profundas, capaces de procesar grandes volúmenes de información en tareas como reconocimiento de imágenes y procesamiento del lenguaje natural (LeCun, Bengio & Hinton, 2015). Sin embargo, uno de los desafíos en el entrenamiento de estos modelos es el **data leakage, que ocurre cuando información del conjunto de prueba se filtra inadvertidamente en el entrenamiento, generando evaluaciones optimistas, que no reflejan el desempeño real del modelo en datos nuevos** (Kaufman, Rosset, Perlich & Stitelman, 2012). Identificar y mitigar este problema es crucial para asegurar la validez y reproducibilidad de los modelos en entornos reales.

RDA 2: Evalúa los modelos de aprendizaje automático para identificar y cuantificar potenciales sesgos y limitaciones, empleando métricas adecuadas.

Clase 6. Aprendizaje supervisado segunda parte

Naïve Bayes

Naïve Bayes Classifier es un algoritmo simple, de la familia de los clasificadores basados en probabilidad. Se llama '*Naïve'* (ingenuo) porque asume que, dada 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:

- Altamente eficientes en cuento a capacidad de aprendizaje y predicción se refiere.
- 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).

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, la Figura 1 muestra cómo sería la *decision boundary* para un problema tipo de clasificación.

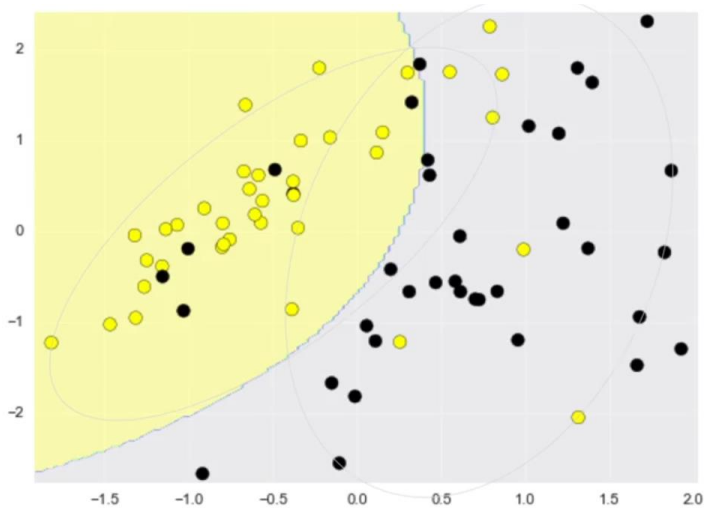


Figura 1. Decision Boundary – Gaussian Naïve Bayes Classifier.

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	

Random Forest

Random Forest no es un solo árbol de decisión, es un ensamble de árboles decisión. Adicionalmente, es ampliamente usado y tiende a tener excelentes resultados en la resolución de múltiples problemas.

Scikit–learn tiene una implementación específica para este tipo de algoritmos y se encuentra en el módulo *sklearn.ensemble:*

- *Clasificación: RandomForestClassifier*
- *Regresión: RandomForestRegressor*

Un solo árbol de decisión tiende a caer en *overfitting*, mientras que el uso combinado de varios árboles de decisión tiende a generar modelos más estables, con una mejor generalización. La clave en este algoritmo es la diversidad de árboles de decisión individuales que lo componen. Los ensambles de árboles deberían ser diversos e introducir una variación aleatoria en la construcción de cada uno. El proceso de generación del *Random Forest* se explica de la siguiente manera:

El proceso de selección de los **datos** de entrenamiento para cada uno de los árboles se hace de forma aleatoria, este proceso se conoce como ***bootstrap sampling***. Del mismo modo, los ***features*** que se usarán en cada uno de los árboles se seleccionan de forma aleatoria también. Lo primero que se hace para construir un **Random Forest** es escoger la cantidad de árboles de decisión que tendrá, eso se define con el parámetro **n_estimator**, que se aplica tanto para problemas de regresión como para los de clasificación. Luego, el parámetro **max_features** define la cantidad de ***features*** que se consideran en cada uno de los árboles de decisión.

La Figura 2 ilustra el proceso que sigue el Random Forest.

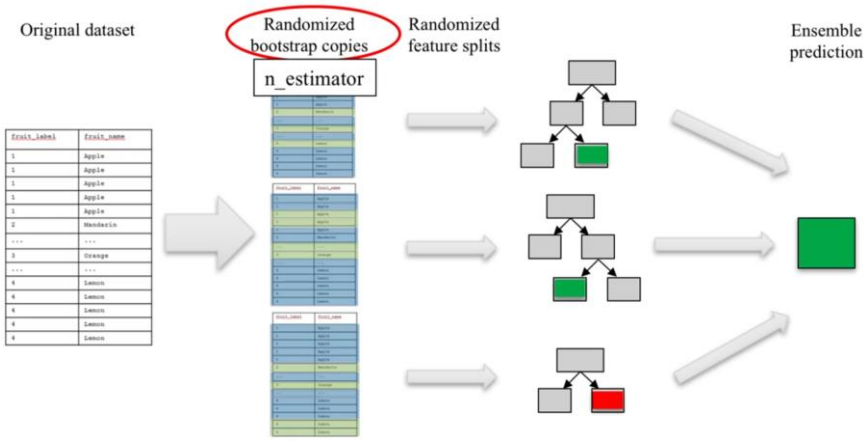


Figura 2. Random Forest Process.

El proceso de aprendizaje del Random Forest es sensible al valor que tenga el parámetro **max_features**. Un valor de **max_features** de 1 tendería a crear árboles diversos y, en consecuencia, más complejos. Por otro lado, con un valor de **max_features** cercano a la cantidad total de *features* se obtendrán árboles más simples.

Predicciones usando Random Forest

1. Se hace una predicción para cada uno de los árboles del Random Forest.
2. Las predicciones de los árboles se combinan de la siguiente manera:
    a. Para un modelo de regresión: se toma el promedio de las predicciones individuales
    b. Para un modelo de clasificación:
        i. Cada árbol genera una probabilidad para cada clase.
        ii. Se saca el promedio de las probabilidades de todos los árboles.
        iii. La predicción será la clase con la probabilidad más alta.

La Figura 3 ilustra el proceso descrito en el párrafo anterior.

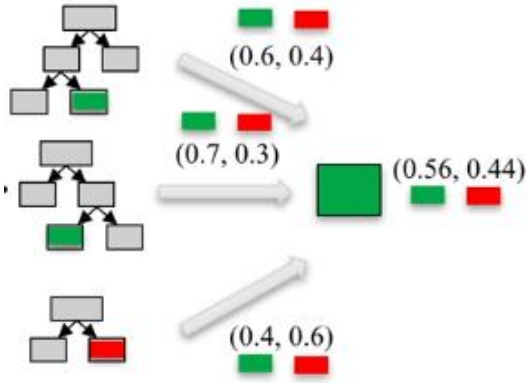


Figura 3. Predicciones usando Random Forest.

Ventajas y Desventajas de usar Random Forest

Ventajas	**Desventajas**
Altamente usado, muy buen *performance* en muchos problemas	Los resultados suelen ser difíciles de interpretar para los humanos.
No necesita normalización de datos o un extensivo tuneo de parámetros.	Al igual que los árboles de decisión, el Random Forest podría no ser una buena alternativa para problemas con alta dimensionalidad.
Al igual que en los árboles de decisión, maneja una gran cantidad de tipos de datos.	
Fácilmente paralelizable usando múltiples *CPU*.	

Parámetros esenciales para el *RandomForestClassifier*

- **n_estimator**: número de árboles a usar en el *ensamble.*
- **max_features**: tiene un gran efecto en el *performance.*
- **max_depth**: controla la profundidad de cada árbol.
- **n_jobs**: cantidad de *cores* en paralelo durante el proceso de entrenamiento.

Redes neuronales

Las **redes neuronales artificiales (ANN, por sus siglas en inglés)** son modelos computacionales inspirados en el funcionamiento del cerebro humano y están diseñadas para reconocer patrones complejos en los datos (Goodfellow, Bengio, & Courville, 2016). Se utilizan ampliamente en problemas de clasificación, regresión y reconocimiento de imágenes, entre otros (LeCun, Bengio, & Hinton, 2015).

Arquitectura de las redes neuronales

Una red neuronal está compuesta por **neuronas artificiales**, organizadas en tres tipos de capas:

1. **Capa de entrada**: recibe los datos de entrada.
2. **Capas ocultas**: procesan la información aplicando funciones de activación como **ReLU, sigmoide o tanh** (Nielsen, 2015).
3. **Capa de salida**: genera la predicción final del modelo.

Cada neurona está conectada a otras, a través de **pesos sinápticos;** estos se ajustan mediante algoritmos de entrenamiento, como **descenso de gradiente estocástico (SGD) o Adam** (Kingma & Ba, 2015).

Tipos de redes neuronales:

1. **Perceptrón multicapa (MLP)**: se compone de múltiples capas y es entrenado con el algoritmo de **retropropagación** (Rumelhart, Hinton, & Williams, 1986).
2. **Redes convolucionales (CNNs)**: especializadas en el procesamiento de imágenes, utilizan capas convolucionales para detectar patrones en diferentes escalas (LeCun et al., 1998).
3. **Redes recurrentes (RNNs)**: diseñadas para procesar secuencias de datos, como textos o series temporales (Hochreiter & Schmidhuber, 1997).
4. **Redes profundas (Deep Learning)**: son redes con muchas capas ocultas, lo que permite aprender representaciones más complejas de los datos (Goodfellow et al., 2016).

Aplicaciones de las redes neuronales:

Las redes neuronales han revolucionado diversas áreas, entre ellas:

- **Visión por computadora**: en tareas como reconocimiento facial y conducción autónoma (He, Zhang, Ren, & Sun, 2016).
- **Procesamiento de lenguaje natural (NLP)**: se usan en traductores automáticos y asistentes virtuales (Vaswani et al., 2017).
- **Medicina**: diagnóstico de enfermedades mediante el análisis de imágenes médicas (Esteva et al., 2017).

Revisión de regresión lineal y logística

A continuación, repasamos dos algoritmos básicos de *machine learning* para regresión y clasificación respectivamente. En esencia, para ambos casos se tienen *features* de entrada, a los cuales se les asigna un peso y se aplica una función para obtener una salida.

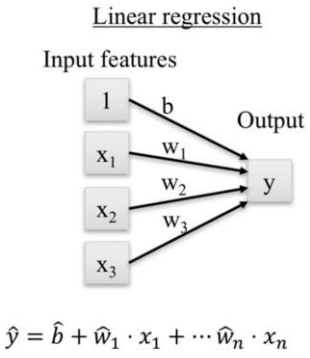


Figura 4. Regresión lineal.

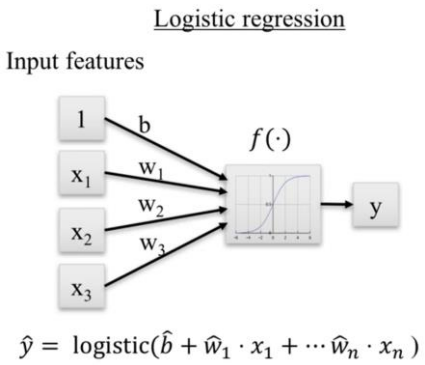


Figura 5. Regresión logística**.**

Perceptrón multicapa con una capa oculta

El **perceptrón multicapa (MLP, Multi-Layer Perceptron)** es un tipo de **red neuronal artificial** que consiste en múltiples capas de neuronas organizadas en una estructura jerárquica, lo que le permite modelar relaciones no lineales en los datos (Goodfellow, Bengio, & Courville, 2016). Se considera una extensión del **perceptrón** simple, introducido por Rosenblatt en 1958, pero con la inclusión de una o más **capas ocultas**, que mejoran su capacidad de aprendizaje (Rosenblatt, 1958).

Arquitectura del MLP

El MLP está compuesto por tres tipos de capas principales:

1. **Capa de entrada**: recibe los datos y los pasa a la siguiente capa sin modificaciones (Nielsen, 2015).
2. **Capas ocultas**: responsables de procesar la información aplicando **funciones de activación** como sigmoide, tangente hiperbólica (**tanh**) o **ReLU** (Nielsen, 2015).
3. **Capa de salida**: genera la predicción final, dependiendo del problema, aplicando funciones como **softmax** en clasificación o activación lineal en regresión (LeCun, Bengio, & Hinton, 2015).

La Figura 6 ilustra el proceso que sigue el perceptrón multicapa:

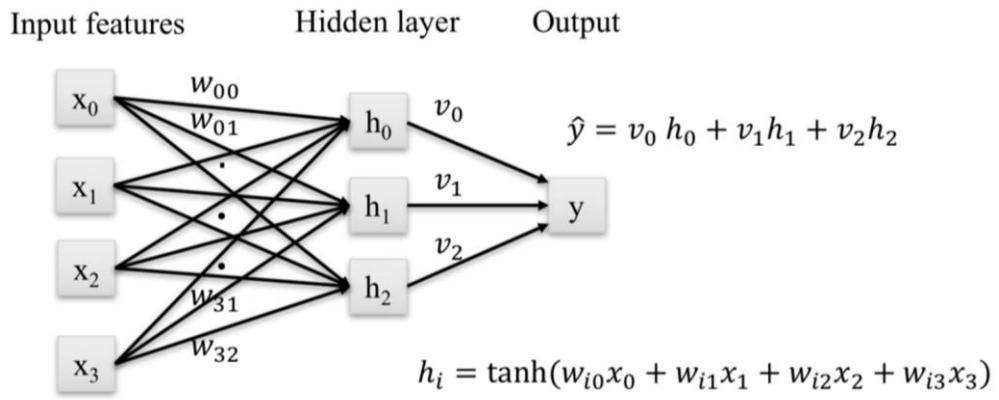


Figura 6. Perceptrón multicapa.

Funciones de activación

Las **funciones de activación** son fundamentales en las redes neuronales, ya que permiten la introducción de no linealidad en el modelo y facilitan el aprendizaje de patrones complejos en los datos (Goodfellow, Bengio, & Courville, 2016). Dos de las funciones más utilizadas en redes neuronales profundas son **ReLU (Rectified Linear Unit)** y **tanh (tangente hiperbólica)**.

Función de activación ReLU (Rectified Linear Unit)

La función **ReLU** se define matemáticamente como:

$$f(x) = \max(0, x)$$

Figura 7. Función de activación ReLU.

Esto significa que **cualquier valor negativo se convierte en 0**, mientras que los valores positivos permanecen sin cambios (Nair & Hinton, 2010). Esta propiedad hace que ReLU sea computacionalmente eficiente y ayude a mitigar el problema del **desvanecimiento del gradiente**, un inconveniente común en redes profundas (Glorot, Bordes, & Bengio, 2011).

Ventajas de ReLU:

- **Evita el desvanecimiento del gradiente** para valores positivos, permitiendo un entrenamiento más eficiente (Goodfellow et al., 2016).
- **Computacionalmente eficiente**, ya que solo requiere una comparación simple para su cálculo (Nair & Hinton, 2010).
- **Introduce esparcidad**, ya que algunos valores de activación se convierten en cero, reduciendo el sobreajuste (Glorot et al., 2011).

Desventajas de ReLU:

- **El problema de las neuronas muertas**: si demasiadas neuronas tienen valores negativos, estas dejan de aprender porque su gradiente se vuelve cero (Maas, Hannun, & Ng, 2013).
- **No está centrada en cero**, lo que puede generar inestabilidad en la optimización (Glorot & Bengio, 2010).

Función de activación tanh (tangente hiperbólica)

La función **tanh** se define matemáticamente como:

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Figura 8. Función de activación tanh.

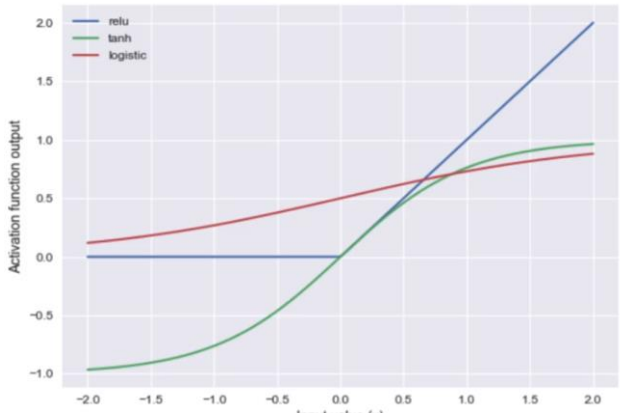
Ventajas de tanh:

- **Centrada en cero**, lo que mejora la estabilidad del entrenamiento en comparación con la sigmoide (LeCun et al., 1998).
- **Útil para datos con valores negativos y positivos**, ya que permite una mejor distribución de las activaciones (Goodfellow et al., 2016).

Desventajas de tanh:

- **Sufre del problema del desvanecimiento del gradiente**, lo que puede dificultar el entrenamiento de redes profundas (Bengio, Simard, & Frasconi, 1994).
- **Computacionalmente más costosa** que ReLU, ya que requiere operaciones exponenciales (LeCun et al., 1998).

La Figura 9 ilustra la distribución de varias funciones de activación:



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Figura 9. Funciones de activación.

Ventajas y desventajas de las redes neuronales

Ventajas	**Desventajas**
Se consideran las bases del estado del arte de modelos que incluyen arquitecturas avanzadas que capturan, de forma efectiva, la complejidad de los *features* dada la suficiente data y por de cómputo.	Modelos más complejos requieren más tiempo de entrenamiento, así como mayor customización.
	Se necesita un preprocesamiento de datos cuidadoso.
	Es buena opción cuando los *features* son de tipos similares; pero mala cuando los *features* son de diferente tipo.

Parámetros importantes para MLPClassifiers y MLPRegressors

- **hidden_layer_sizes:** configura el número de capas ocultas y de neuronas por capa.
- **alpha:** controla el peso en la penalización por regularización, que reduce los pesos a 0.
- **activation:** controla la función no lineal usada para definir la función de activación (ReLU, logistic, tanh)

Data leakage

Data leakage ocurre cuando los datos que se están usando para entrenar contienen información sobre lo que se está tratando predecir. Introducir información sobre el *target* durante el proceso de entrenamiento no es legítimamente adecuado. A continuación, cito algunos ejemplos obvios.

- Usar el *target* como un *feature* más.
- Incluir datos de testeo en el conjunto de datos de entrenamiento.

Si el performance de tu modelo es 'demasiado bueno', muy probablemente se deba a un tema de *data leakage.* A continuación, se muestra una ilustración a modo de ejemplo de usar todo el *dataset* para calcular el promedio:

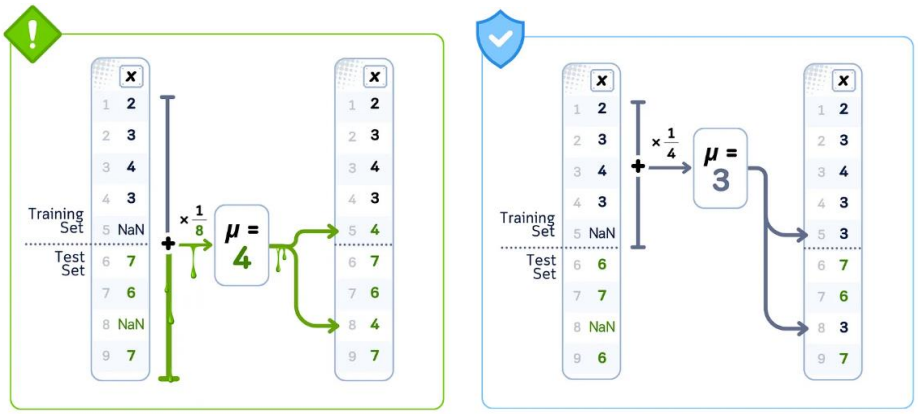


Figura 10. *Data leakage* https://medium.com/towards-data-science/data-leakage-in-preprocessing-explained-a-visual-guide-with-code-examples-33cbf07507b7

Ejemplos sutiles de *data leakage*

- ***Target**:* usuario se queda o no en la página web. *Giveaway feature:* duración de la sesión con base en información acerca de futuras visitas.
- ***Target**:* predecir si un usuario en un banco es propenso a abrir una cuenta. *Giveaway feature:* el número de cuenta solo se llena cuando el usuario abre una.
- Cualquiera de estas ***leaked features*** es altamente predictiva del *target,* pero no estará legítimamente disponible cuando la nueva predicción se tenga que realizar.

Otros ejemplos de *data leakage*

Data Leakage [https://www.bluevoyant.com/knowledge-center/data-leakage-common-causes-examples-tips-for-prevention]

Leakage en los datos de entrenamiento:

- Realizar tareas de preprocesamiento usando parámetros calculados a partir de usar el *dataset* completo.
- En series de tiempo: cuando se usan registros de datos futuros al momento realizar una nueva predicción.
- Errores en la adquisición de datos o cuando se usan valores que indican ausencia de datos (e.g. el valor de 999) puede encriptar información sobre valores faltantes que revelan información acerca del futuro.

Leakage en *features*:

- Eliminar variables que no son legítimas, sin eliminar variables que encriptan la misma información o información relacionada.
- Deshacer randomización intencional, que releva información específica acerca del *target*.

Detectar *data leakage:*

- Antes de construir el modelo:
    - Realizar un análisis exploratorio de datos para encontrar posibles sorpresas en los datos.
    - ¿Los *features* están altamente correlacionados con el target?
- Después de construir el modelo:
    - Buscar *features* que tengan algún comportamiento sorpresivo en el modelo entrenado.
    - ¿Hay *features* que tienen un peso muy alto o alto, *information gain*?
    - ¿El performance final del modelo es sorpresivamente muy alto comparado con el performance de modelos similares?
- Limitaciones en el *deployment* de los modelos:
    - Potencialmente costoso en términos de tiempo de desarrollo, pero más realista.
    - ¿El modelo entrenado generaliza bien con nuevos datos?

The Theachery of Leakage [https://medium.com/@colin.fraser/the-treachery-of-leakage-56a2d7c4e931]

Minimizar el *data leakage*

- Realizar *data preparation* con cada *fold* de *cross validation* por separado.
- Cuando se trabaje con series de tiempo, usar *timestamp.*
- Antes de trabajar con un nuevo *dataset*, se divide un *dataset* de validación final.

Referencias citadas en la Clase 6

- Breiman, L. (2001). Random forests. *Machine Learning, 45*(1), 5-32.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- Kaufman, S., Rosset, S., Perlich, C., & Stitelman, O. (2012). Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD), 6*(4), 1-21.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436-444.
- Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*(7639), 115-118.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)* (pp. 770-778).
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735-1780.
- Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533-536.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems (NeurIPS)*

Definición de los términos citados en la Clase 6

Campana de Gauss.

La campana de Gauss, también conocida como **distribución normal**, es una función matemática que describe cómo se distribuyen los datos en muchas situaciones del mundo real. Su forma es simétrica y tiene un pico central, donde se agrupa la mayoría de los valores, mientras que los extremos representan valores menos frecuentes. Se define por dos parámetros: la media, que indica el centro de la distribución; y la desviación estándar, que mide la dispersión de los datos. Es fundamental en estadística, ya que muchos fenómenos naturales y sociales, como la estatura de las personas o los errores de medición, siguen este patrón.

Perceptrón multicapa

El **perceptrón multicapa** (MLP, por sus siglas en inglés) es un tipo de red neuronal artificial, compuesta por múltiples capas de neuronas: una capa de entrada, una o más capas ocultas y una capa de salida. Cada neurona en una capa está conectada con las neuronas de la siguiente, a través de pesos ajustables; y las señales se propagan hacia adelante, mediante funciones de **activación no lineales**, como la **sigmoide** o **ReLU**. Durante el entrenamiento, se usa el algoritmo de retropropagación para ajustar los pesos y minimizar el error. El **MLP** es ampliamente utilizado en problemas de clasificación y regresión, debido a su capacidad para modelar relaciones complejas en los datos.

PUCE Virtual
EDUCACIÓN FLEXIBLE

La excelencia no se improvisa

síguenos

MDASC - 12 - 241