
AI ALGORITHMS

Study Book

Author

Nelson Bruno Andrés Moreno Cabañas

Santiago, Chile

2024

Contents

1	Introduction	3
2	Machine Learning	3
2.1	Linear Regression	3
2.1.1	Regularization	4
2.2	Decision Trees	5
2.3	Random Forest	6
2.3.1	Ensemble Methods	6
2.4	Gradient Boosting Algorithm	8
2.5	Naive Bayes	9
2.5.1	Gaussian Naive Bayes	9
2.5.2	Multinomial Naive Bayes	10
2.6	Support Vector Machines	10
2.6.1	Soft Margin	12
2.6.2	Kernel Trick	12
2.6.3	Kernels	13
2.7	ARIMA	13
2.7.1	P Value	14
2.7.2	D Value	14
2.7.3	Q Value	15
2.8	K-Means	16
2.9	DBSCAN	17
2.10	Principal Component Analysis (PCA)	18
3	Others	19
3.1	Performance Metrics	19
3.2	Bias vs Variance	19
3.3	Oversampling and Undersampling	20
3.4	Random Noise Feature Importance	20
3.5	SHAP Values	20
3.6	Outlier Detection	20
3.6.1	Interquantile Range	21
3.6.2	Z - Score	21
3.6.3	DBSCAN	22
3.6.4	Isolation Forest	22
3.7	Cross-Validation Techniques	22
3.7.1	Time Series Cross Validation	23
4	Statistics	23
4.1	Causal Inference	23
4.1.1	Matching Imputation	24
4.1.2	Meta Learners	25

1 Introduction

Esta es la introducción.

2 Machine Learning

Consideremos un conjunto de datos $X = (x_1, \dots, x_N)$ donde para cada $i \in 1, \dots, N$, $x_i = [x_i^1, \dots, x_i^M]$ es decir, un conjunto de N datos con M features cada uno. Consideremos además $Y = (y_1, \dots, y_N)$ las etiquetas o labels cada cada dato. En problemas de clasificación binario, $y_i \in \{0, 1\}$ y en problemas de clasificación multiclase, $y_i \in \{1, \dots, L\}$. Para problemas de regresión, $Y = (y_1, \dots, y_N)$ con cada $y_i \in \mathbb{R}$.

2.1 Linear Regression

Un modelo de regresión lineal es un modelo de aprendizaje **supervisado** utilizado para problemas de **regresión** y **clasificación**. Se construye a través de la búsqueda de parámetros β_0, \dots, β_M que multiplican el *input* para obtener la predicción. En problemas de regresión,

$$\hat{y} = f(\beta) = \beta_0 + \sum_{j=1}^M \beta_j x^j$$

y en problemas de clasificación a través de una función sigmoide

$$\hat{y} = f(\beta) = \frac{1}{1 + e^{-(\beta_0 + \sum_{j=0}^M \beta_j x^j)}}$$

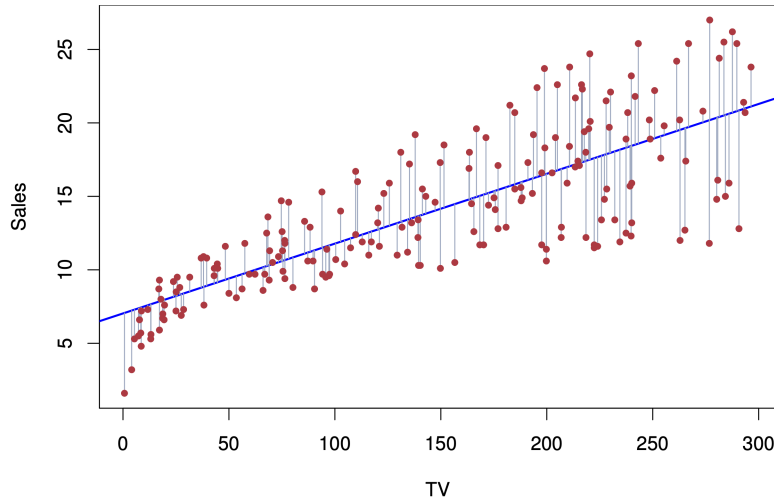


Figure 1: Linear Regression Diagram

El problema a optimizar (mínimos errores cuadrados) queda entonces definido por

$$\min_{\beta} \quad \frac{1}{N} \sum_{i=1}^N (y_i - f(\beta))^2$$

Cuya solución es cerrada en el caso de estimar una regresión y estimada a través del descenso de gradiente en el caso de una clasificación.

2.1.1 Regularization

Para prevenir el *overfitting* y que la importancia de los parámetros quede mejor distribuida, es posible agregar a la función un término regularizador de la siguiente forma

$$\begin{aligned} \min_{\beta} \quad & \frac{1}{N} \sum_{i=1}^N (y_i - f(\beta))^2 \\ \text{s.t.} \quad & \|\beta\|_p^p \leq t \end{aligned}$$

De manera equivalente, por el método de *Lagrange*, sin optimizar el valor de λ y eliminando constantes que no dependen de β

$$\min_{\beta} \quad \frac{1}{N} \sum_{i=1}^N (y_i - f(\beta))^2 + \lambda \|\beta\|_p^p$$

Cuando $p = 1$ se le conoce como regresión **Lasso** y cuando $p = 2$ una regresión **Ridge**. La combinación de ambas restricciones es conocida como **Elastic Net**.

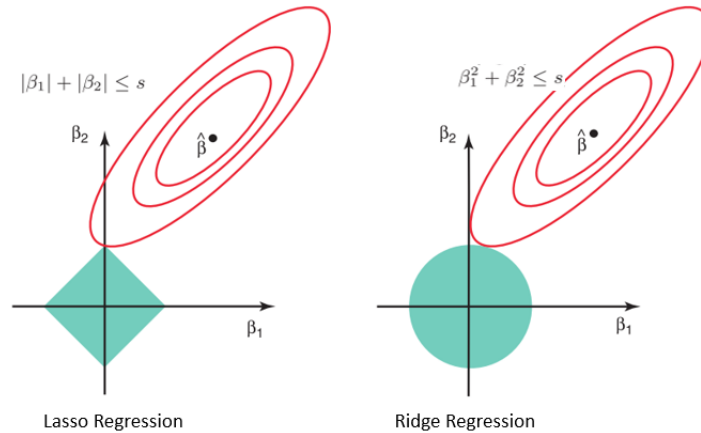


Figure 2: Lasso and Ridge Diagram

Notar que la restricción para el caso *Lasso* hace más probable que las curvas de nivel intersecten la restricción en una esquina (en mayor dimensión es incluso más probable) por lo que los parámetros que no son importantes para el modelo, serán llevados a 0. En el caso de la restricción *Ridge*, la forma permite que los valores queden acotados pero ninguno será llevado a 0.



Figure 3: Regularization Feature Importance

2.2 Decision Trees

Un árbol de decisión es un modelo de aprendizaje **supervisado** utilizado para problemas de **regresión** y **clasificación**. El objetivo es aprender simples reglas de decisión a partir de las features.



Figure 4: Decision Tree Diagram

En el caso de un problema de clasificación, la variable a escoger y el corte correspondiente se puede elegir como aquel que minimice el desorden de los elementos. Definimos primero la **entropía** según

$$H(p) = - \sum_{j=1}^L p_j \log_2 p_j$$

donde p_j es la frecuencia relativa del label j en un grupo. Notar que la entropía es mínima cuando el grupo solo tiene elementos de la clase 0 o de la clase 1 ($p_j = 1$) y máxima cuando hay la misma cantidad de elementos de cada clase ($p_j = \frac{1}{2}$).

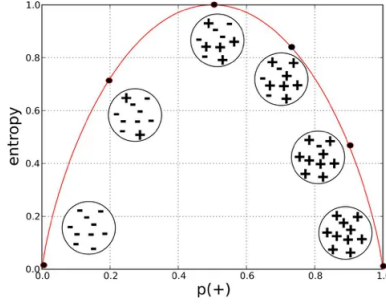


Figure 5: Entropy Diagram

Con la entropía ya definida, definamos la **Information Gain** como

$$IG(S, D) = H(S) - \sum_{V \in D} \frac{|V|}{|D|} H(V)$$

Donde el primer término es la entropía antes del split y el segundo término es la suma de las entropías después del split. Podemos iterar hasta que la *Information Gain* no tenga modificaciones (es decir, llegar a los nodos puros) pero esto podría traer problemas de *overfitting*, en general esto se regula con la profundidad del árbol y escogiendo en cada iteración, la división que maximiza el IG.

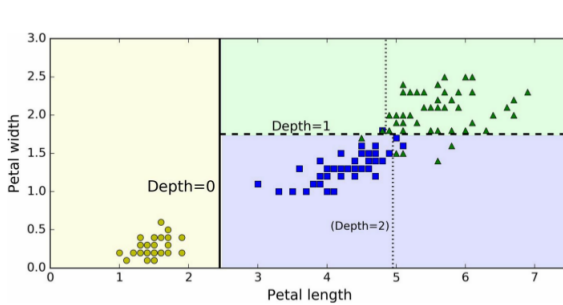
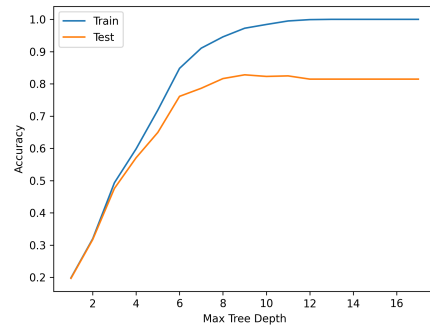


Figure 6-2. Decision Tree decision boundaries

(a) Decision Tree Boundaries



(b) Max Depth and Accuracy

Figure 6: Decision Tree Implementation

En vez de la entropía, es posible usar otro indicador como el **Gini Index** definido como:

$$G(p) = 1 - \sum_{j=1}^L p_j^2$$

2.3 Random Forest

2.3.1 Ensemble Methods

Los **métodos de ensamblaje** son aquellos en los que se combinan múltiples estimadores entrenados sobre los datos para generar una predicción más robusta (menor varianza) y generalizada. La predicción

final se puede realizar por *Majority Voting*, *Simple Average* o *Weighted Average*.

Existen 3 estrategias principales en los métodos de ensamblaje:

1. **Bagging**: Corresponde a una abreviación de *Bootstrap Aggregating*, esta estrategia entrena cada estimador base con una **muestra con reemplazo** de ejemplos del conjunto de entrenamiento. (**Reduce la varianza**)
2. **Boosting**: Esta estrategia se basa en entrenar secuencialmente estimadores base débiles que **aprenden de los errores del anterior** para crear un estimador robusto. (**Reduce el bias**)
3. **Stacking**: Este método combina las predicciones de múltiples estimadores fuertes en una sola.

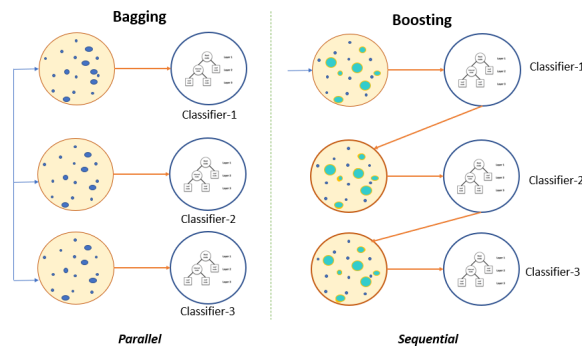


Figure 7: Bagging and Boosting Diagram

El algoritmo de *Random Forest* es un método **supervisado de ensamblaje** basado en *Decision Trees*. Este, utiliza la estrategia de **bagging** para entrenar cada árbol de decisión sobre muestras con reemplazo del conjunto de entrenamiento y además, cada árbol es entrenado sobre un **subconjunto aleatorio de features** para asegurar que no haya similitud entre ellos. Ambas estrategias permiten mejorar la precisión del modelo y controlar el *overfitting*.

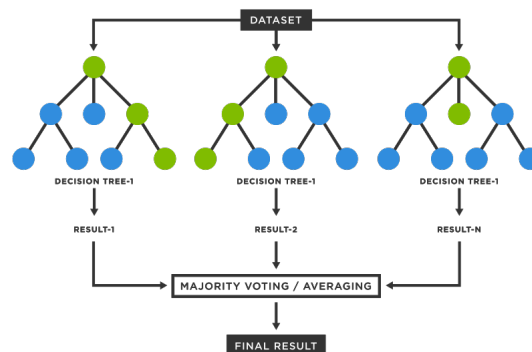


Figure 8: Random Forest Diagram

Para estimar la **feature importance** en modelos basados en árboles, se pueden mirar 3 posibles aspectos:

1. Suma de las *information gain* con una variable determinada (o cuánto reduce la entropía, es equivalente).
2. Cantidad de nodos resultantes de la división con esa variable.
3. Profundidad promedio de un árbol en que su primera división es con esa variable.

Para extender esto a *Random Forest*, se toma un promedio entre los distintos árboles de decisión que lo conforman.

2.4 Gradient Boosting Algorithm

Los modelos de *Gradient Boosting* son algoritmos basados en árboles que pueden ser utilizados para problemas de **clasificación y regresión**.

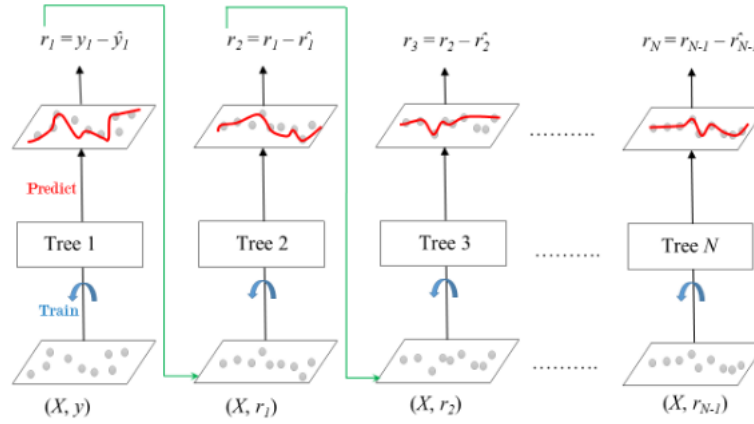


Figure 9: Gradient Boosting Diagram

El entrenamiento consta en el aprendizaje secuencial de árboles de baja complejidad (*weak learners*) que aprenden de los errores del anterior, vale decir, que aprenden en el conjunto de los **residuos** del modelo anterior. Sea $L(y, F(x))$ el error de la predicción del modelo $F(x)$ y el valor real y .

El modelo $F(x)$ es el resultado de una combinación lineal de *weak learners* tal que

$$F(x) = \sum_{k=1}^T \gamma_k h_k(x)$$

Donde T es la cantidad de árboles (n estimators), h_k los *weak learners* y γ_k su peso. Iniciando en un modelo inicial $F_0(x)$ entrenado sobre los datos, cada iteración se encarga de construir $F_k(x)$ según

$$F_k = F_{k-1} + \gamma_k h_k$$

Para encontrar los valores γ_k, h_k se siguen los siguientes pasos:

1. Calcular los residuos i en la iteración k como $r_{i,k} = - \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \right]_{F(x)=F_{k-1}(x)}$. En el caso de L una función de error MSE, $r_{i,k} = 2(y_i - F_{k-1}(x_i)) \propto y_i - F_{k-1}(x_i)$

2. Entrenar un *weak learner* sobre (X, r) .
3. Encontrar el valor de γ_k como aquel que resuelve

$$\gamma_k = \arg \min_{\gamma} \sum_{i=1}^N L(y_i, F_{k-1} + \gamma h_k(x_i))$$

4. Hacer update al modelo según $F_k = F_{k-1} + \gamma_k h_k$

La importancia de las variables es medida de manera similar a un *Random Forest*, a través de la suma ponderada de las *Information Gain* (o bien de la disminución de la entropía) que cada una de las features provoca en los distintos árboles.

Existen otras formas de calcular la importancia de las variables, lo cual se discute en la sección 3.5.

2.5 Naive Bayes

Este modelo de aprendizaje **supervisado** puede ser utilizado para problemas de clasificación. Este algoritmo es una aplicación del teorema de *Bayes* en el que se asume (*Naive*) la independencia condicional entre los pares de features x^j dado el valor del label y

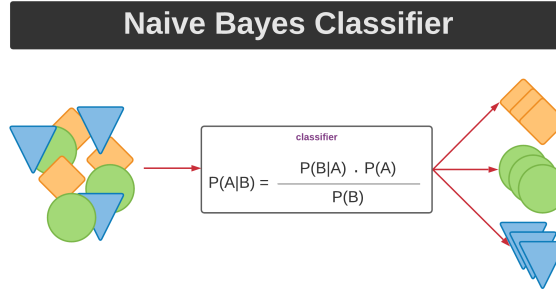


Figure 10: Naive Bayes Diagram

La formulación es la siguiente:

$$P(y|x^1, \dots, x^M) = \frac{P(y)P(x^1, \dots, x^M|y)}{P(x^1, \dots, x^M)} = \frac{P(y) \prod_{j=1}^M P(x^j|y)}{P(x^1, \dots, x^M)} \propto P(y) \prod_{j=1}^M P(x^j|y)$$

y la predicción se realiza según

$$\hat{y} = \operatorname{argmax}_y P(y) \prod_{j=1}^M P(x^j|y)$$

2.5.1 Gaussian Naive Bayes

Aquí consideramos que $x^j|y \sim \mathcal{N}(\mu_j, \sigma_j^2)$, es decir, cada feature sigue una distribución normal según:

$$P(x^j|y) = \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\frac{(x^j - \mu_j)^2}{2\sigma_j^2}\right)$$

donde la media y varianza μ_j y σ_j respectivamente, son calculados a través de la máxima verosimilitud, es decir, si $x_i = (x_i^1, x_i^2, \dots, x_i^M)$ con $i \in \{1, \dots, N\}$ los datos, entonces $\mu_j = \frac{1}{N} \sum_{i=1}^N x_i^j$ y $\sigma_j = \frac{1}{N-1} \sum_{i=1}^N (x_i^j - \mu_j)^2$. Notar que esto se debe calcular utilizando los datos de la clase respectiva y .

La definición anterior funciona bien con tipos de data numéricos.

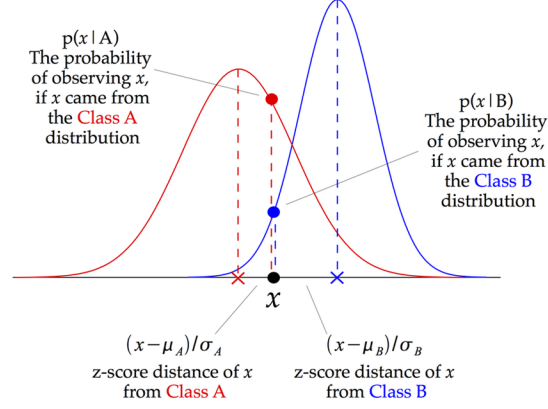


Figure 11: Gaussian Naive Bayes Diagram

2.5.2 Multinomial Naive Bayes

Aquí consideramos que $x^j|y$ sigue una distribución multinomial donde los parámetros $(p_{j_1}, \dots, p_{j_k})$ de esta distribución son calculados según

$$p_{j_k} = \frac{N_{j_k} + \alpha}{N_j + \alpha}$$

Donde N_{j_k} es la cantidad de veces que la categoría k de la feature j aparece en los datos con clase y del conjunto de entrenamiento y $N_j = \sum_k N_{j_k}$. El parámetro α es un *Smoothing Prior* para estabilidad numérica.

La definición anterior funciona bien con tipos de datos categóricos.

2.6 Support Vector Machines

Las *Support Vector Machines* o SVM, son modelos de aprendizaje **supervisados** que pueden ser usados para problemas de clasificación y regresión. Se construyen a partir de la búsqueda de un hiper-plano separador y vectores de soporte que maximizan la distancia entre las clases.

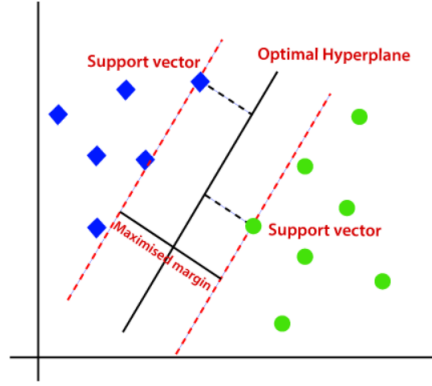


Figure 12: SVM Diagram

Consideremos el caso binario donde $Y \in \{0, 1\}^n$. El hiper-plano separador está definido por

$$H = \{x \in \mathbb{R}^n | w^\top x + b = 0\}$$

Donde w es el vector perpendicular al hiper-plano y b un *offset*. De esta forma si $w^\top x + b > 0$ quiere decir que x pertenece a la clase 1 y $w^\top x + b < 0$ que x pertenece a la clase 0, escrito de otra forma

$$y_i(w^\top x + b) \geq 1 \quad \forall i \in \{1, \dots, N\}$$

En el caso de un problema de clasificación linealmente separable, existen infinitos hiper-planos que satisfacen las condiciones anteriores (basta con rotar ligeramente el hiper-plano) por lo que vamos a exigir además las siguientes condiciones sobre vectores de soporte x_- y x_+ .

$$\begin{aligned} w^\top x_+ + b &= 1 \\ w^\top x_- + b &= -1 \end{aligned}$$

Notar entonces que con esta condición, es posible calcular el ancho m de la separación entre el hiper-plano y el vector de soporte. Recordemos que la distancia m de un vector x a un hiperplano con vector normal w viene dada por

$$m = \frac{|\langle w, x \rangle|}{\|w\|}$$

Considerando la definición de los vectores de soporte, se tiene que

$$2m = \frac{|\langle w, x_+ \rangle|}{\|w\|} + \frac{|\langle w, x_- \rangle|}{\|w\|} = \frac{|1 - b|}{\|w\|} + \frac{|-1 - b|}{\|w\|}$$

Además el *offset* $b \in [0, 1]$ por la definición anterior, así

$$m = \frac{1}{\|w\|}$$

Finalmente, el problema de optimización quedaría de la siguiente forma

$$\begin{aligned} & \max_{\omega, b} \quad \frac{1}{\|w\|} \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \forall i \in \{1, \dots, N\} \end{aligned}$$

Equivalente a

$$\begin{aligned} & \min_{\omega, b} \quad \frac{1}{2} \|w\|^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \forall i \in \{1, \dots, N\} \end{aligned}$$

Este problema se resuelve utilizando el **dual** (*quadratic programming*) y es fundamental para la extensión no-lineal de la SVM (*Kernel Trick*).

2.6.1 Soft Margin

Los datos son usualmente no linealmente separables, por lo que hay que permitir un error en la clasificación de ciertos puntos (*soft margin*). Este cambio en el problema de optimización se reduce a la siguiente regularización:

$$\begin{aligned} & \min_{\omega, b} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \forall i \in \{1, \dots, N\}, \xi_i \geq 0 \end{aligned}$$

2.6.2 Kernel Trick

En la formulación del problema dual, la función objetivo requiere el cómputo del producto interno entre todos los puntos del conjunto de entrenamiento. Si buscamos proyectar nuestra data a una dimensión mayor (aplicar algún mapeo ϕ para hacer la separación posible), esto se puede realizar calculando los productos $\langle \phi(x_i), \phi(x_j) \rangle$ para todo i y j .

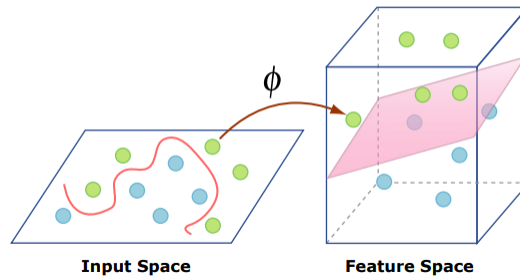


Figure 13: Kernel Trick

El paso fundamental del truco del kernel es que no es necesario conocer el mapeo ϕ explícitamente pues por el teorema de *Mercer*

$$K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$$

donde $K : X \times X \rightarrow \mathbb{R}$ es un kernel de *Mercer* en un espacio de *Hilbert* (posiblemente de dimensión infinita), por lo que solo basta que definamos K para tener un posible mapeo de las características.

2.6.3 Kernels

Para que un *kernel* pueda utilizarse en el contexto de las SVM, es importante que cumpla con las condiciones de *Mercer*:

- Symmetry: $K(x, y) = K(y, x) \quad \forall x, y$
- Positive Semi-Definiteness: Para cualquier vector $c \in \mathbb{R}^n$, y x_1, \dots, x_n una cantidad finita de puntos, se debe satisfacer que

$$\sum_{i,j} c_i c_j K(x_i, x_j) \geq 0$$

Algunos ejemplos de *kernels* que se pueden utilizar son:

1. **Linear Kernel:** $K(x, y) = \langle x, y \rangle$.

El más útil cuando la data es linealmente separable.

2. **Polynomial Kernel:** $K(x, y) = (x^\top y + c)^d$.

El parámetro d controla el nivel de complejidad del kernel pero valores muy altos podrían llevar a *overfitting*.

3. **Gaussian Radial Basis Function (RBF) Kernel:** $K(x, y) = e^{-\gamma \|x-y\|^2}$.

Este kernel es el más popular pues **mapea los datos a un espacio de dimensión infinita**. El parámetro γ controla la complejidad de los *decision boundaries* al agregar mayor o menor *spread* al kernel.

4. **Sigmoid Kernel:** $K(x, y) = \tanh(ax^\top y + b)$

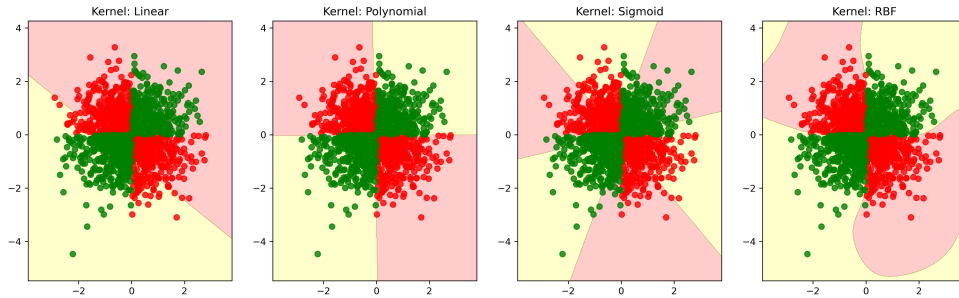


Figure 14: Kernel Decision Boundaries

2.7 ARIMA

ARIMA es la abreviación de *Auto-Regressive Integrated Moving Average* y es un método estadístico para realizar *forecasting* sobre series de tiempo que integra los siguientes conceptos:

1. Toma en consideración patrones de crecimiento/decrecimiento en la serie de tiempo (*Auto-Regressive*).

2. Estima tasa de crecimiento/decrecimiento (*Integrated*).
3. Controla el ruido entre datos consecutivos en el tiempo (*Moving Average*).

La fórmula general para este tipo de modelos viene dada por

$$Y_t = c + \phi_1 y_{t-1}^d + \cdots + \phi_p y_{t-p}^d + \theta_1 e_{t-1} + \cdots + \theta_q e_{t-q} + e_t$$

Aquí c es una constante y e es un término de error. Los modelos de este tipo son escritos como $ARIMA(p, d, q)$ donde:

- p es la cantidad de tiempos en que la variable es mirada al pasado (*Lag*).
- d es la cantidad de veces que la variable es diferenciada para producir una señal estacionaria. $d = 0$ refiere a que la señal ya es estacionaria, $d = 1$ es que la señal crece/decrece linealmente y $d = 2$ es que la señal crece exponencialmente.
- q representa la cantidad de *lag* para el término de error e , esto captura el *Moving Average*.

2.7.1 P Value

En la práctica, es posible determinar el valor de p a través del *Partial Autocorrelation Plot*. Este gráfico muestra la relación de un valor en la serie de tiempo con **un solo lag** (eliminando relaciones de tiempos intermedios ajustando una regresión lineal y quedándose sólo con el parámetro correspondiente).

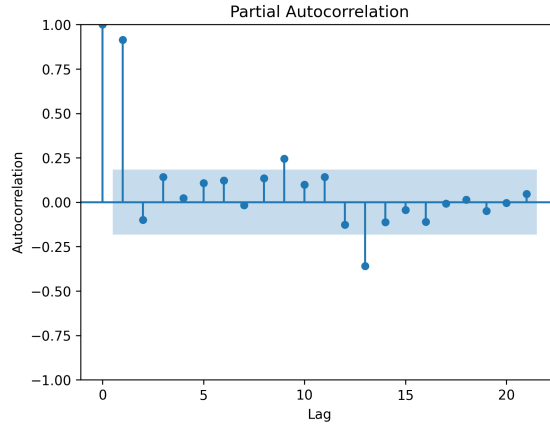


Figure 15: Partial Autocorrelation Plot

El valor óptimo es el último punto después del cual todos los lags están dentro de las bandas azules (intervalos de confianza), en este caso $p = 13$.

2.7.2 D Value

El valor de d se puede calcular diferenciando la serie de tiempo hasta encontrar una serie estacionaria. Esto lo podemos medir con un test estadístico de estacionalidad (*Augmented Dickey Fuller Test*).

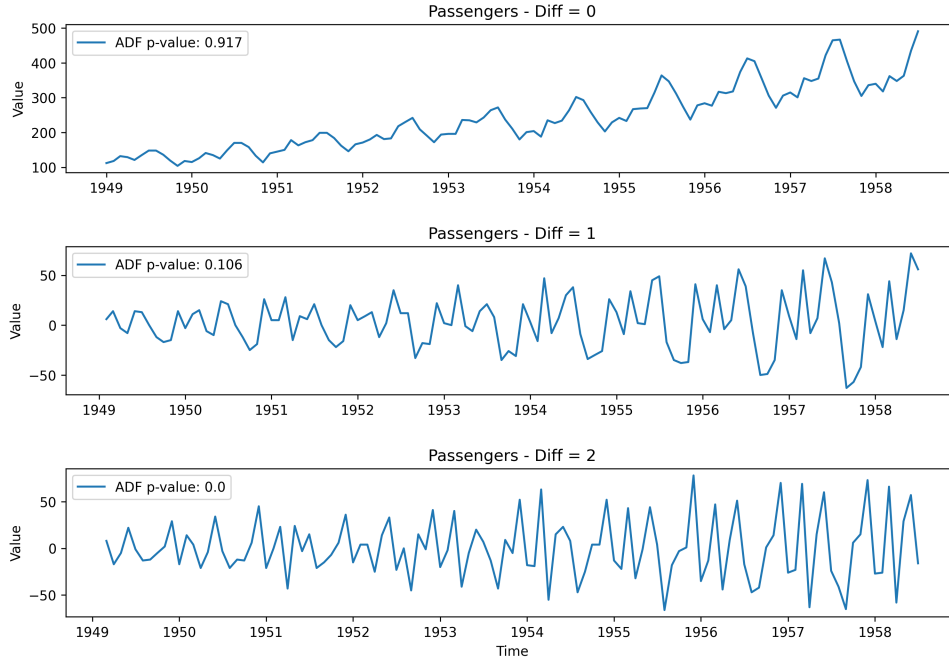


Figure 16: Time Series Differentiation Plot

Vemos que el p -valor en la segunda diferenciación ya es lo suficientemente pequeño para asumir estacionalidad en la serie de tiempo, así $d = 2$.

2.7.3 Q Value

Finalmente, para determinar el valor q es posible ver el *Autocorrelation Plot* que muestra la relación de un valor en la serie de tiempo con **todos los p lags** anteriores.

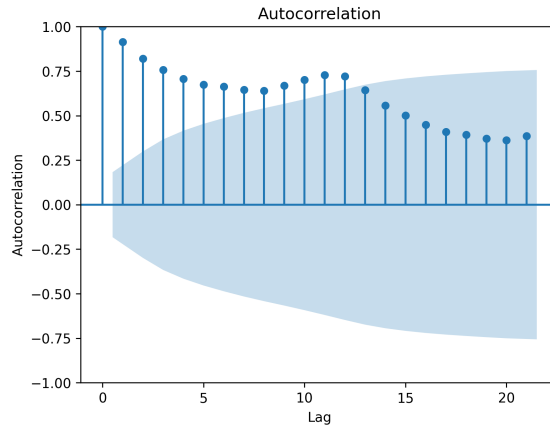


Figure 17: Autocorrelation Plot

En este caso, el último valor anterior a que todos los lag caigan en la zona azul, es $q = 12$.

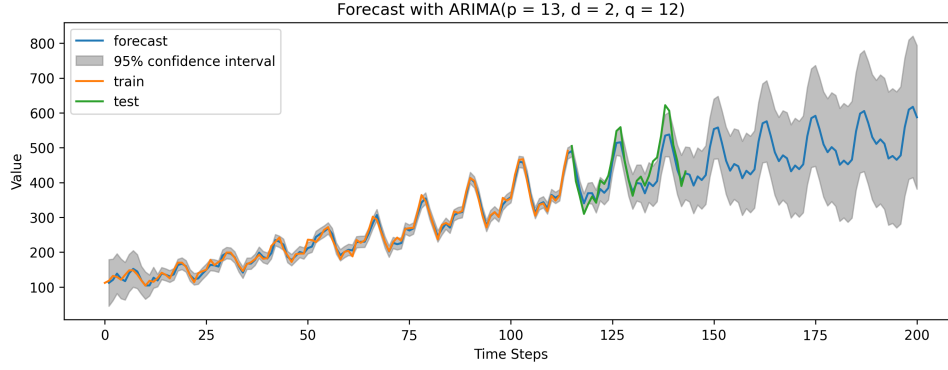


Figure 18: ARIMA over Flight Passengers Forecasting

2.8 K-Means

El algoritmo de *K-Means* es un algoritmo **no supervisado de clustering**. El objetivo es agrupar los datos alrededor de **centroides** de tal forma en que se minimice la suma de alguna medida de distancia. Vale decir, encontrar el conjunto $S = \{S_1, \dots, S_k\}$ de clusters (k hiperparámetro) tal que

$$S = \arg \min_S \sum_{i=1}^k \sum_{x \in S_i} \|x - \mu_i\|_2^2$$

Notar que $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x_i$ es el centroide del cluster S_i para todo i . El algoritmo que realiza esta búsqueda es el siguiente:

1. Seleccionar k centroides aleatorios del conjunto de datos.
2. Asignar cada punto del conjunto de datos al centroide más cercano.
3. Actualizar el nuevo centroide del cluster S_i según $\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x_i$. Iterar los pasos 2 y 3 hasta que la asignación no cambie.

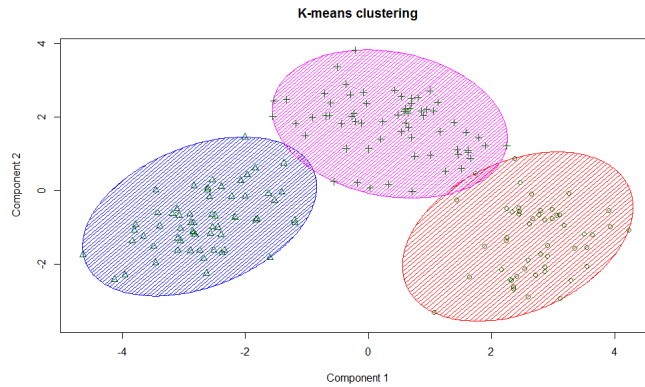


Figure 19: K-Means Diagram

Este algoritmo siempre converge pero no asegura el **óptimo global**. Además, la distancia euclidiana asume que los clusters tienen **forma esférica**. Por último, es muy **sensible a outliers**

La selección del hiper-parámetro k se puede hacer utilizando el método del codo (*Elbow Method*). Es decir, calcular el error de la función objetivo para distintos valores de k y seleccionar el menor valor de k tal que la función objetivo no cambie en gran magnitud con los valores siguientes.

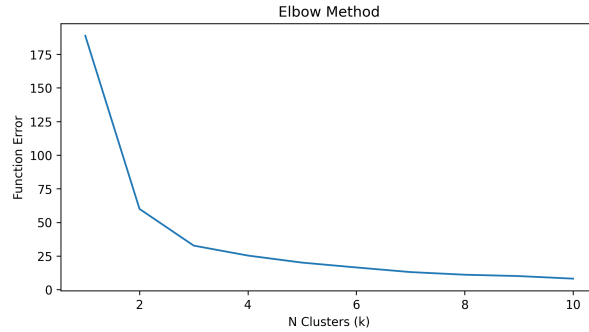


Figure 20: Elbow Method

En este ejemplo, el valor óptimo de clusters con el *Elbow Method* es $k = 3$.

2.9 DBSCAN

El algoritmo DBSCAN (*Density-based spatial clustering of applications with noise*) es un algoritmo basado en densidad que permite resolver problemas de **clustering** y realizar **outlier detection**.

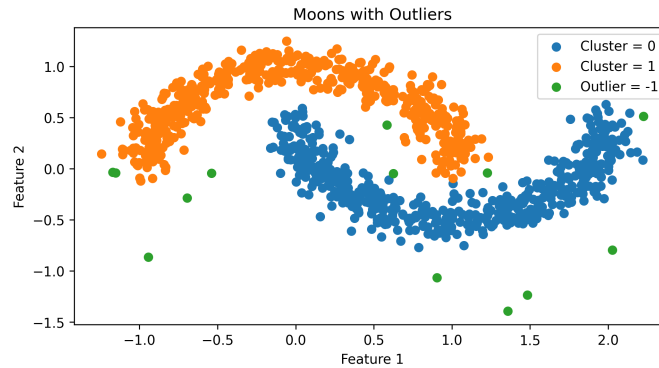


Figure 21: DBSCAN Example

Para realizar esto, se siguen los siguientes pasos:

1. Encontrar todos aquellos puntos (*core points*) que tengan más de $MinNeighbor$ vecinos a distancia menor de ϵ .
2. Todos los *core points* conexos serán asignados a un cluster.
3. Todos los *non core points* serán asignados al cluster del *core point* más cercano. El resto será considerado como *outliers*.

Sea $C = \{C_1, \dots, C_k\}$ el número de clusters generados, el algoritmo anterior asegura la optimalidad del siguiente problema de optimización:

$$\begin{aligned} \min_C \quad & |C| \\ \text{s.t.} \quad & d(p, q) \leq \epsilon \quad \forall p, q \in C_i, \quad \forall C_i \in C \end{aligned}$$

2.10 Principal Component Analysis (PCA)

El análisis de componentes principales (PCA) es un método no supervisado de **reducción de dimensionalidad**. Consideremos la matriz con los datos $X \in \mathcal{M}_{N \times M}$, es decir, el dataset con N filas y M columnas. Se define la matriz de covarianza $K = (K_{ij})_{ij} \in \mathcal{M}_{M \times M}$ donde

$$K_{ij} = \text{Cov}(X_i, X_j) = \mathbb{E}((X_i - \mathbb{E}(X_i))(X_j - \mathbb{E}(X_j))) = \mathbb{E}(X_i X_j) - \mathbb{E}(X_i)\mathbb{E}(X_j)$$

Cuando las features i, j son linealmente independientes, entonces $K_{ij} = 0$. Si ambas crecen conjuntamente, entonces su covarianza será positiva y de caso contrario, será negativa. A continuación, se calculan los **valores y vectores propios**, es decir, los valores $(\lambda_i)_i$ y $(v_i)_i$ que cumplen para todo i la relación

$$Kv_i = \lambda v_i$$

Los componentes principales serán aquellos vectores propios $(v_i)_i$ cuyo valor propio asociado $(\lambda_i)_i$ es mayor, es decir, la **dirección de máxima varianza**. Para obtener la transformación de los datos a k dimensiones, basta juntar los k vectores propios con mayor valor propio en una matriz $V \in \mathcal{M}_{M \times k}$ y calcular

$$\hat{X} = XV \in \mathcal{M}_{N \times k}$$

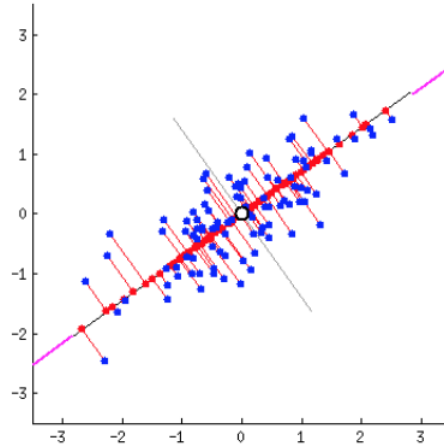


Figure 22: PCA Diagram

En problemas de clasificación o *clustering*, este enfoque puede ser útil para **visualizar etiquetas/clusters** en un gráfico de 2 o 3 dimensiones, o bien, puede ser utilizado para disminuir y condensar la información de un dataframe.

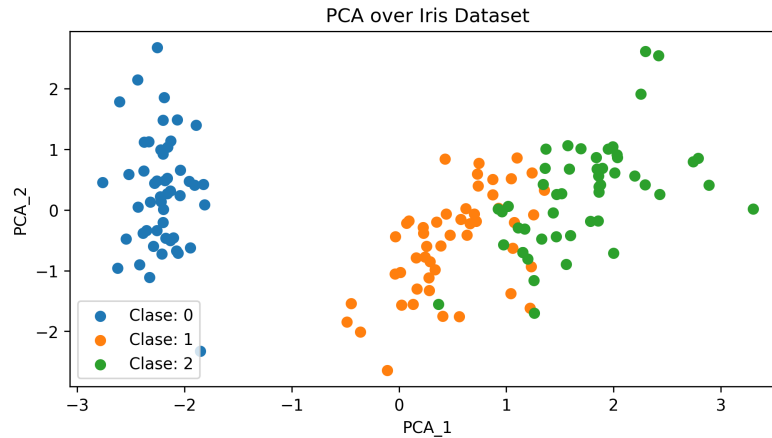


Figure 23: PCA over Iris Dataset

3 Others

3.1 Performance Metrics

3.2 Bias vs Variance

El dilema de *Bias vs Variance* describe la relación entre la complejidad del modelo, la precisión de las predicciones y cómo éste se comporta al predecir datos nunca antes vistos. El error estimado de una predicción viene dado en términos generales por

$$\text{Expected Error} = (\text{Bias})^2 + \text{Variance} + \text{Irreducible Error}$$

Así, un modelo que crece en complejidad reducirá su bias pero aumentará su varianza (extremo: *overfitting*) y a la vez, reducir la complejidad permitirá generalizar mejor reduciendo la varianza pero aumentando el bias (extremo: *underfitting*).

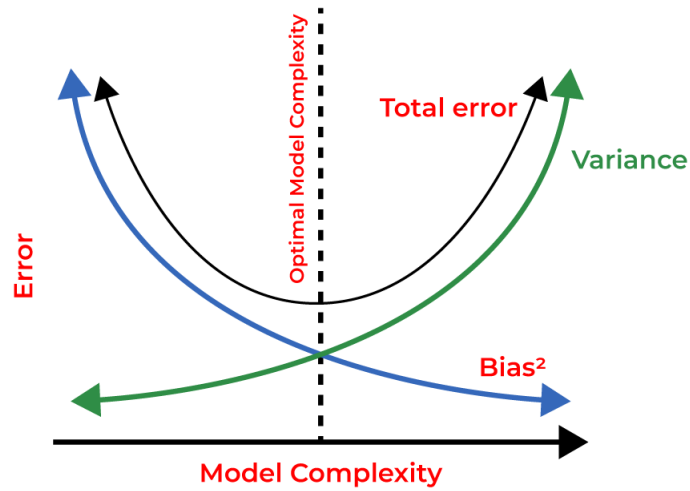


Figure 24: Bias vs Variance Diagram

3.3 Oversampling and Undersampling

3.4 Random Noise Feature Importance

3.5 SHAP Values

El SHAP values (*SHapley Additive exPlanations*) es un algoritmo modelo-agnóstico basado en teoría de juegos que permite **interpretar las predicciones**, la importancia de las variables y su impacto.

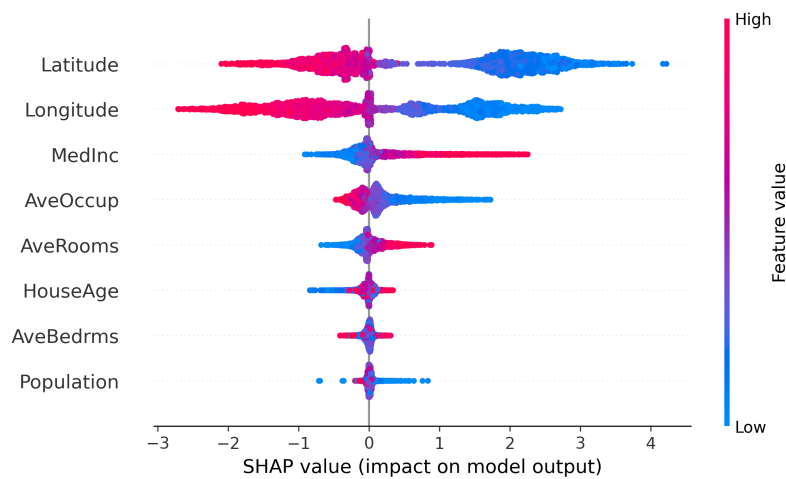


Figure 25: SHAP Values Example

3.6 Outlier Detection

La detección de *outliers* es la práctica de encontrar **datos anómalos** o fuera de distribución en un *dataset*. Es de suma importancia para la construcción de modelos de fraude o para mejorar el

performance de modelos sensibles a *outliers*.

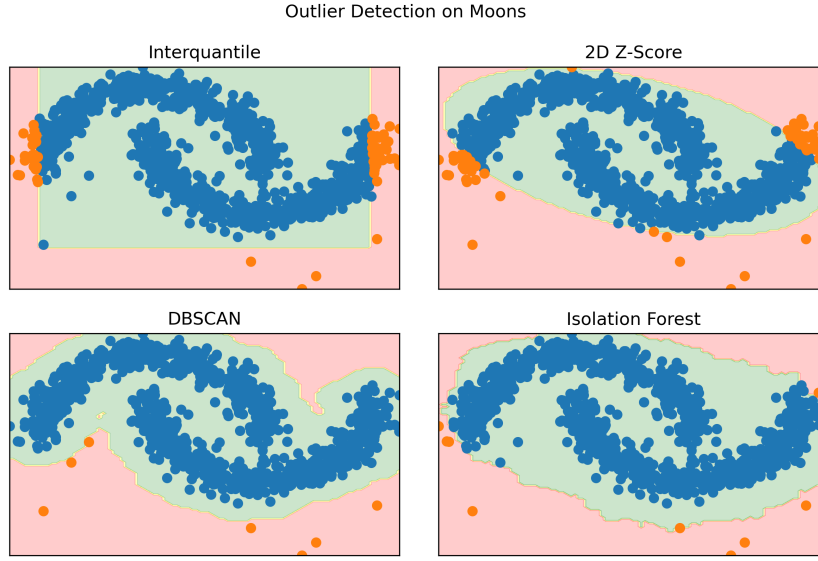


Figure 26: Outlier Detection Contour Plot

3.6.1 Interquantile Range

Definimos el *Interquantile Range* IC como la diferencia entre el percentil 75 y el percentil 25. Es decir,

$$IC = Q_3 - Q_1$$

En este caso, los *outliers* serán aquellos datos x_i tal que $x_i \geq Q_3 + I_{\text{range}}IC$ o bien $x_i \leq Q_1 - I_{\text{range}}IC$ donde I_{range} permite controlar el porcentaje de outliers que se espera encontrar. (Equivalencia en distribución normal $I_{\text{range}} = 1.5 \approx Z_{\text{value}} = 2.69$).

Notar que en esta metodología no se asume normalidad en la distribución de los datos.

3.6.2 Z - Score

Cuando los datos siguen una distribución normal, se pueden considerar como outliers aquellos datos x_i que cumplen

$$\left| \frac{x_i - \mu}{\sigma} \right| \geq Z_{\text{value}}$$

Para los casos con más de una feature (multidimensional), se puede extender esta definición utilizando la **distancia de Mahalanobis** definida como

$$D = \sqrt{(x_i - \mu_i)^\top \Sigma^{-1} (x_i - \mu_i)}$$

Donde Σ es la matriz de covarianza y μ_i el vector de media.

3.6.3 DBSCAN

Los datos que no cumplan el mínimo de vecinos a distancia menor de ϵ son considerados por DBSCAN como *outliers*.

3.6.4 Isolation Forest

El algoritmo de *Isolation Forest* selecciona de manera aleatoria una feature y un corte aleatorio entre el mínimo y el máximo valor presente en esa feature. Esto es repetido múltiples veces hasta aislar cada uno de los puntos.

Este algoritmo iterativo, puede ser representado en el siguiente diagrama:

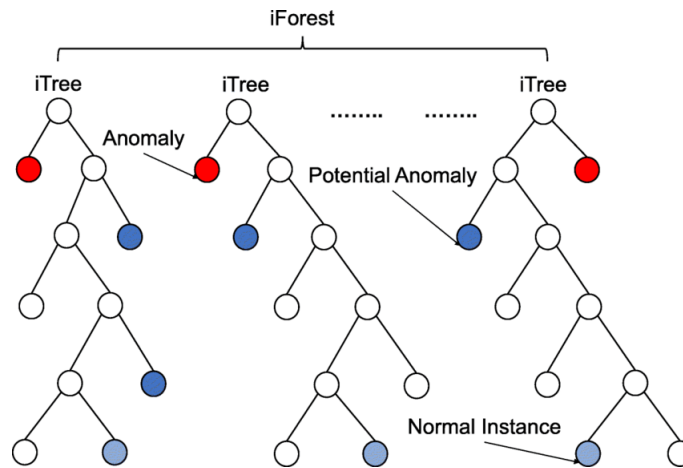


Figure 27: Isolation Forest Diagram

Los puntos que en promedio, requieren **menos divisiones para ser aislados**, son los que el algoritmo considera como potenciales datos anómalos (*outliers*).

3.7 Cross-Validation Techniques

El *Cross-Validation* es un método que permite evaluar el rendimiento de un modelo de *Machine Learning* y que busca eliminar el sesgo de la elección del conjunto de entrenamiento y testeo. Es ampliamente utilizado para la búsqueda de los mejores hiper-parámetros de un modelo.

1. **KFold**: Esta técnica consiste en dividir el conjunto de entrenamiento en k partes. En cada iteración, se selecciona una de las particiones para testing y el resto para training. El score será el promedio de los scores obtenidos. Existe una variación llamada **StratifiedKFold** en la cual se asegura además que el *target* tenga la misma distribución en cada partición.
2. **ShuffleSplit**: Esta técnica selecciona la partición de entrenamiento y testing de manera aleatoria. No asegura que las particiones sean las mismas en cada iteración.
3. **LeavePOut**: Esta técnica deja p datos para el conjunto de testeo y entrena con todo el resto. Este proceso se realiza con todas las combinaciones posibles lo que asegura una estimación con menor *bias* aunque es extremadamente ineficiente.

3.7.1 Time Series Cross Validation

En el caso de series de tiempo, es importante notar que al momento de realizar *Cross-Validation* para problemas de *Forecast*, es importante **no entrenar con datos posteriores al testing** pues en la práctica, no se tendría acceso a estos.

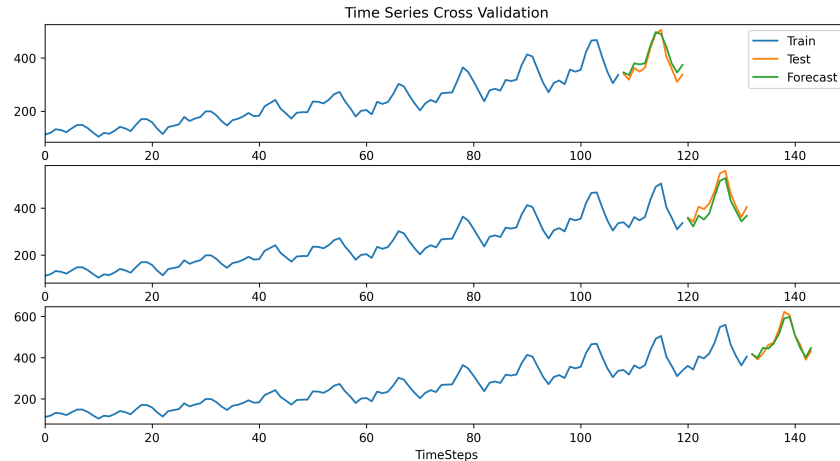


Figure 28: Time Series Cross-Validation Example

4 Statistics

4.1 Causal Inference

La inferencia causal es el proceso de determinar el **efecto independiente de una variable** en un sistema más complejo. En general, este proceso es necesario cuando buscamos obtener conclusiones de datos pasados y en los que no es posible realizar un *A/B testing*.

Existen 3 desafíos al realizar este proceso:

- **Cofounders:** Son aquellas variables que tienen un impacto en el *outcome* y que incluso podrían tener un impacto en otras variables.
- **Selection Bias:** Selección no representativa del grupo de control y tratamiento.
- **Counterfactuals:** Imputar valores en el grupo de control y tratamiento en base a *Machine Learning* o algoritmos de *Matching*.

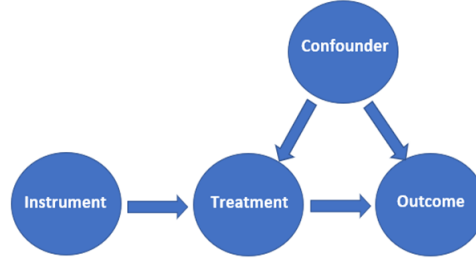


Figure 29: Casual Inference - DAG Diagram

Para resolver este problema, es necesario tomar algunos supuestos:

1. **Causal Markov Condition:** La influencia de las variables y el *outcome* puede ser representado a través de un **Grafo Acíclico Dirigido (DAG)** en el que se asume la **condición de Markov**, es decir si $Y \rightarrow S \rightarrow C$, podemos asumir que $C \perp\!\!\!\perp Y|S$
2. **SUTVA:** (Stable Unit Treatment Value Assumption) El grupo de control y tratamiento **no tiene influencia el uno con el otro**.
3. **Ignorability:** Se excluye el ruido proveniente de cualquier otra fuente.

Consideremos el siguiente ejemplo en el que buscamos determinar si el efecto de un tratamiento tiene un impacto en una variable *target*.

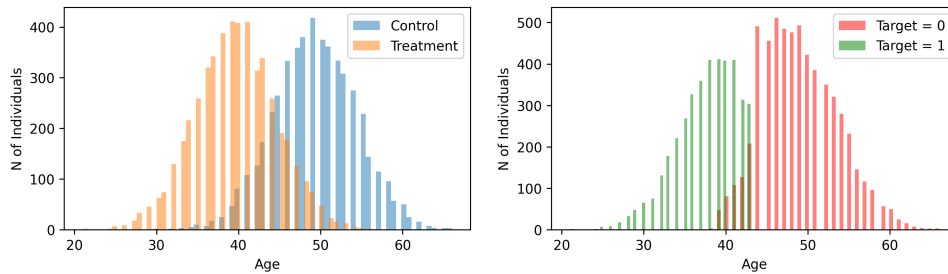


Figure 30: Casual Inference - Distribution Example

Vemos que existe un *selection bias* pues el grupo de tratamiento y control tienen distribuciones de edad distintas. Existen múltiples enfoques para resolver este problema.

4.1.1 Matching Imputation

Es posible utilizar un algoritmo de *Matching* como *NearestNeighbors* para imputar el posible *outcome* que hubiese tenido un individuo al recibir o no el tratamiento. En nuestro ejemplo, para cada edad de los individuos del grupo de control, buscaríamos al sujeto con la edad más cercana en el grupo de tratamiento e imputaríamos su respuesta al tratamiento y viceversa.

La efectividad del tratamiento se puede medir a través del promedio de los *outcome* cuando reciben y cuando no reciben el tratamiento.

4.1.2 Meta Learners

Este enfoque plantea utilizar algoritmos de *Machine Learning* para determinar el efecto del tratamiento en el *outcome* del experimento. Sea y_i el *target*, w si recibió el tratamiento y X_i el conjunto de variables del modelo, definimos ITE (*Individual Treatment Effect*) según

$$\text{ITE} = [p(y_i = 1|w_i = 1, X_i) - p(y_i = 1|w_i = 0, X_i)]$$

- **S - Model** Este algoritmo es el más simple de todos pues agrega la variable *treatment* como input de un único modelo. El valor de ITE es calculado para cada individuo variando el valor del tratamiento.
- **T - Model** Este algoritmo entrena 2 clasificadores, uno encargado del grupo de control y otro para el grupo de tratamiento. El valor de ITE es calculado como la resta del *output* de ambos modelos.

Hay que tener en consideración que este modelo requiere una **calibración** para asegurar que el *output* de los modelos sean probabilidades.

- **X - Model**: SOON

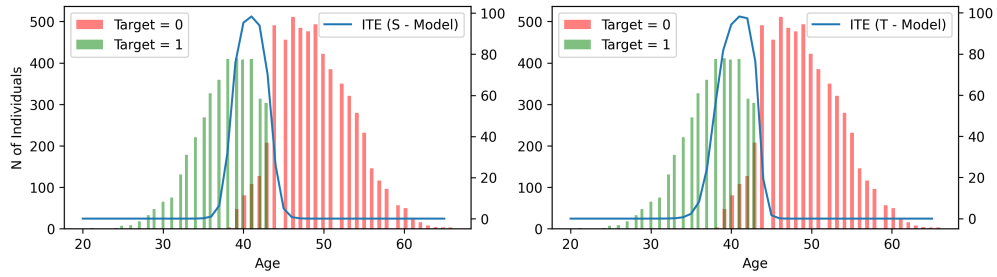


Figure 31: Meta Learners

Vemos así que el impacto del tratamiento aislando el efecto de la edad, ocurre según ITE entre los 35 y 45 años.

References