

---

# AI ALGORITHMS

---

## Study Book

### **Author**

Nelson Bruno Andrés Moreno Cabañas

Santiago, Chile

2024

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Machine Learning</b>	<b>3</b>
2.1	Decision Trees . . . . .	3
2.2	Random Forest . . . . .	4
2.2.1	Ensemble Methods . . . . .	4
2.3	Naive Bayes . . . . .	5
2.3.1	Gaussian Naive Bayes . . . . .	6
2.3.2	Multinomial Naive Bayes . . . . .	7
2.4	Support Vector Machines . . . . .	7
2.4.1	Soft Margin . . . . .	9
2.4.2	Kernel Trick . . . . .	9
2.4.3	Kernels . . . . .	9
2.5	ARIMA . . . . .	10
<b>3</b>	<b>Others</b>	<b>13</b>
3.1	Performance Metrics . . . . .	13
<b>4</b>	<b>REMOVE</b>	<b>14</b>
4.1	Pictures . . . . .	14
4.2	Citation . . . . .	14

# 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,  $y_i \in \{1, \dots, L\}$ .

### 2.1 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.

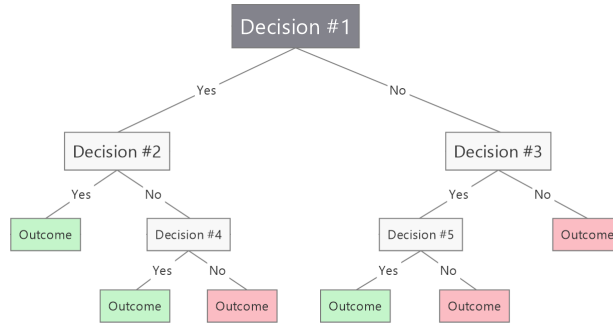


Figure 1: Decision Tree Diagram

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 ( $p_j = \frac{1}{2}$ ).

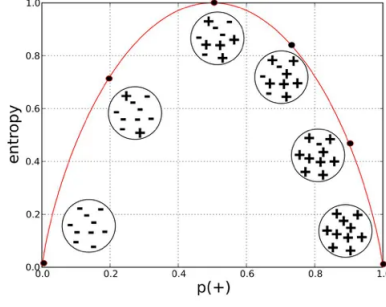


Figure 2: Entropy Diagram

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.

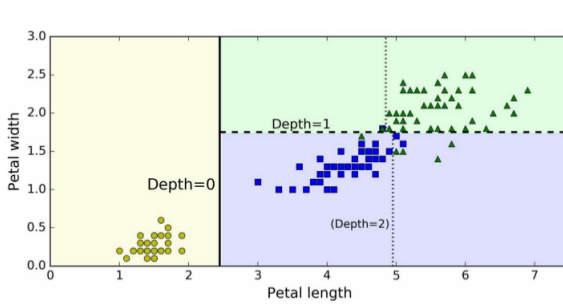
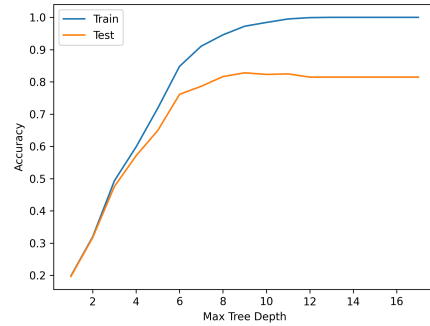


Figure 6-2. Decision Tree decision boundaries

(a) Decision Tree Boundaries



(b) Max Depth and Accuracy

Figure 3: Decision Tree Implementation

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

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

## 2.2 Random Forest

### 2.2.1 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.
2. **Boosting**: Esta estrategia se basa en entrenar secuencialmente estimadores base débiles que **aprenden de los errores del anterior** para crear un estimador robusto.
3. **Stacking**: Este método combina las predicciones de múltiples estimadores fuertes en una sola.

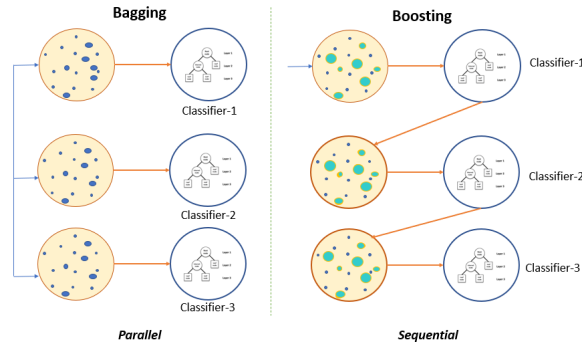


Figure 4: Bagging and Boosting Diagram

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*.

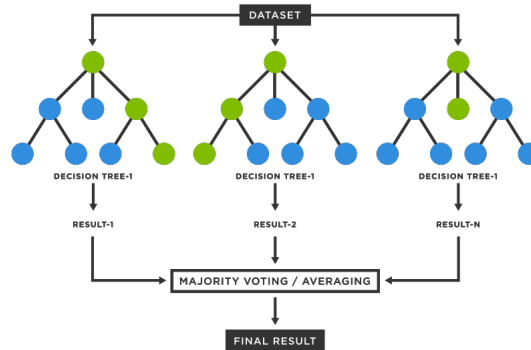


Figure 5: Random Forest Diagram

## 2.3 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 condi-

cional entre los pares de features  $x^j$  dado el valor del label  $y$

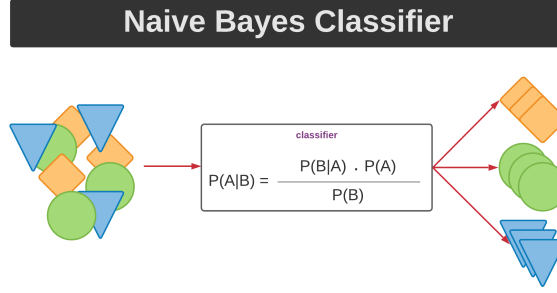


Figure 6: Naive Bayes Diagram

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

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

### 2.3.1 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  $\mu_j$  y  $\sigma_j$  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.

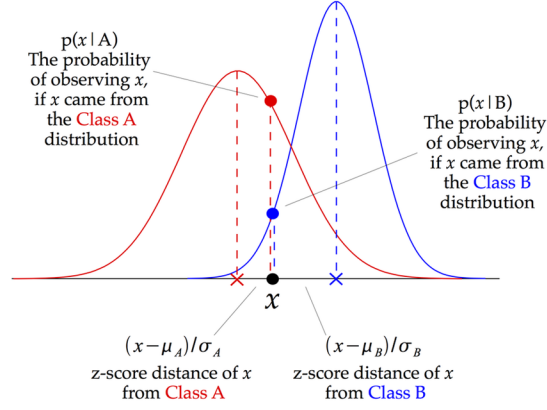


Figure 7: Gaussian Naive Bayes Diagram

### 2.3.2 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  $\alpha$  es un *Smoothing Prior* para estabilidad numérica.

La definición anterior funciona bien con tipos de datos categóricos.

## 2.4 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.

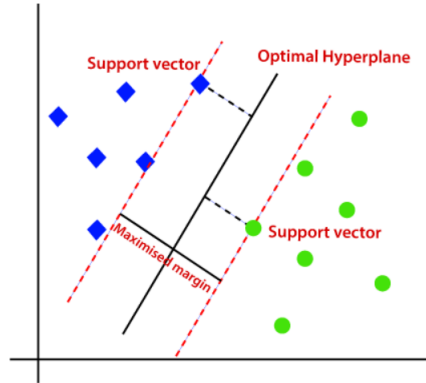


Figure 8: SVM Diagram

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*).



### 2.4.1 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:

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

### 2.4.2 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  $\phi$  para hacer la separación posible), esto se puede realizar calculando los productos  $\langle \phi(x_i), \phi(x_j) \rangle$  para todo  $i$  y  $j$ .

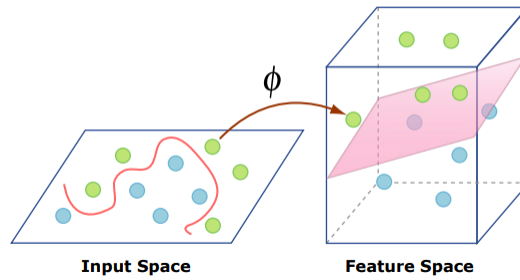


Figure 9: Kernel Trick

El paso fundamental del truco del kernel es que no es necesario conocer el mapeo  $\phi$  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.

### 2.4.3 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  $\gamma$  controla la complejidad de los *decision boundaries* al agregar mayor o menor *spread* al kernel.

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

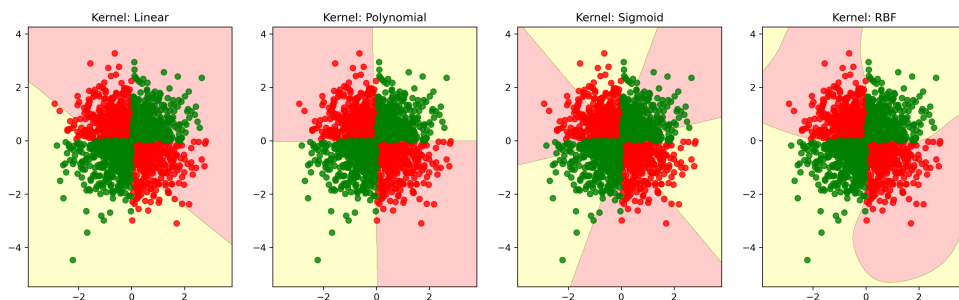


Figure 10: Kernel Decision Boundaries

## 2.5 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

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

Aquí  $c$  es una constante y  $e$  es un término de error. Los modelos de este tipo son escritos como  $\text{ARIMA}(p, d, q)$  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*.

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**

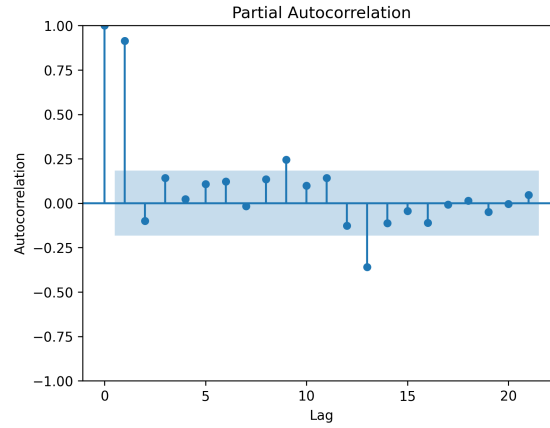


Figure 11: Partial Autocorrelation Plot

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  $p = 13$ .

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*).

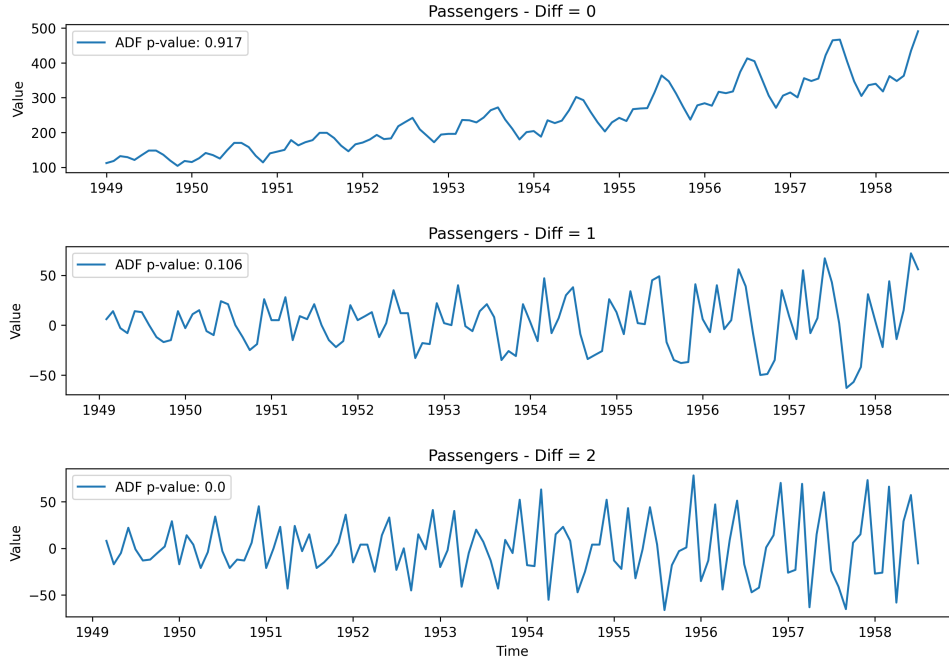


Figure 12: Time Series Differentiation Plot

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

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 lags** anteriores.

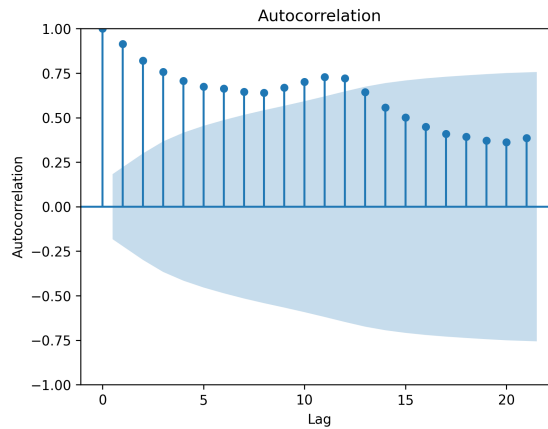


Figure 13: Autocorrelation Plot

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

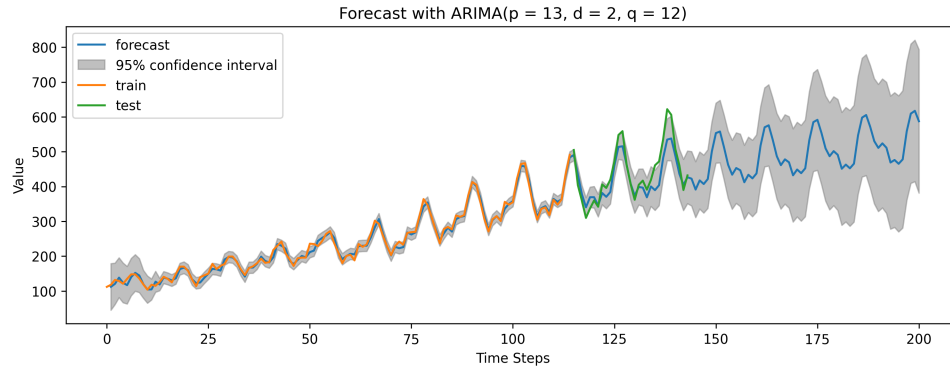


Figure 14: ARIMA over Flight Passengers Forecasting

### 3 Others

#### 3.1 Performance Metrics

## 4 REMOVE

**Theorem 4.1.** This is a theorem.

**Proposition 4.2.** This is a proposition.

**Principle 4.3.** This is a principle.

### 4.1 Pictures

### 4.2 Citation

This is a citation [1].

## References

- [1] H. Ren, “Template for math notes,” 2021.