

# INTRODUCCIÓN PRÁCTICA A MACHINE LEARNING CON PYTHON Y SPARK

ESPECIALIZACIÓN EN CIENCIA DE DATOS - ITBA

Workshop de Big Data

7 de Octubre de 2021



① **Un breve repaso del (necesario) contexto teórico**

② Modelos de clasificación en Python

③ Árboles y ensamble de modelos

④ Conclusión

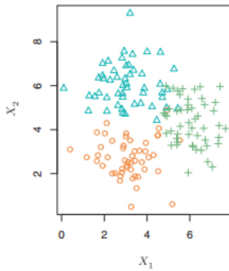
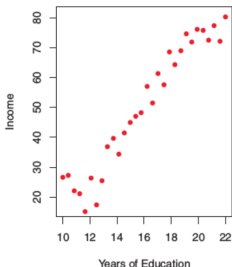
- Subcampo o técnica de la **minería de datos** (o *data mining*)
  - Proceso de negocios que explora grandes volúmenes de datos con el fin de descubrir **patrones y reglas significativas**.
    - Ejemplos: encontrar características de clientes que se dan de baja; productos que se venden previos a catástrofe natural, etc.
- Definición clásica [Mitchell (1997)]:

*“Un **programa de computadora** se dice que aprende de la **experiencia**  $E$  con respecto a una clase de **tareas**  $T$  y **medida de performance**  $P$ , si su performance en las tareas en  $T$ , medidas con  $P$ , mejoran con la experiencia  $E$ .”*

  - Donde:
    - Programa de computadora  $\simeq$  modelo estadístico
    - Experiencia  $E \simeq$  datos
    - Ejemplo: predecir la altura de un niño a 6 meses vista ( $T$ )

# TIPOS O FAMILIAS DE APRENDIZAJE

- Existen tres grandes familias de algoritmos de aprendizaje automático
  1. **Supervisado:** los datos son *instancias etiquetadas*, pares de objetos dados por observaciones y respuestas asociadas
    - Objetivo: generar una función que relacione los datos de entrada con los de salida y pueda ser utilizada con datos desconocidos
    - Ejemplos/Aplicaciones: filtros de Spam, tasa de conversión en marketing online
  2. **No Supervisado:** los datos son observaciones sin respuestas asociadas
    - Objetivo: comprender las relaciones subyacentes entre las distintas observaciones
    - Ejemplos/Aplicaciones: clustering
  3. **Por Refuerzos:** los datos definen un entorno, un conjunto de estados y acciones y funciones de pago (premios/castigos)
    - Objetivo: encontrar una regla o política de acción que maximice el beneficio esperado
    - Ejemplos/Aplicaciones: agentes económicos, Pacman.



# APRENDIZAJE SUPERVISADO Y TERMINOLOGÍA

- Objetivo es predecir una variable, la *etiqueta*. Puede ser de dos tipos:
  - Si la variable es **categorica** (o discreta) se dice que el problema es de **clasificación** (binaria o multiclase)
    - Construir un filtro de correo basura con mensajes anotados como *spam/no-spam*
  - Si la variable es **continua** se dice que el problema es de **regresión**
    - Predecir cantidad de ventas de un call center un día determinado
- ¿Cómo aprendemos? Seamos levemente más formales...
- Dado un vector de entrada  $X$  y un vector de salida  $Y$  asumimos que existe una función (desconocida) tal que  $f : X \rightarrow Y$ :

$$Y = f(X) + \varepsilon$$

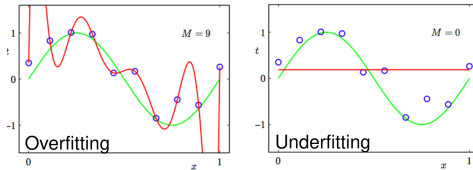
- Aquí  $f$  representa la información *sistemática* que  $X$  provee de  $Y$  mientras que  $\varepsilon$  es el *error irreducible*.
- la literatura de ML llama *target* a la variable explicada y *features o atributos* a las explicativas.
- Para encontrar (una aproximación) de  $f$ , **aprendemos por ejemplos**:
  1. Construimos un **conjunto de entrenamiento** compuesto por  $N$  ejemplos de la forma  $\{(x_1, y_1), \dots, (x_N, y_N)\}$  donde  $x_i$  e  $y_i$  son respectivamente un **vector de atributos** (o features) y su etiqueta asociada.
  2. Los valores de entrada  $x_i$  alimentan algún *algoritmo de aprendizaje* que produce resultado  $\hat{y} = \hat{f}(x_i)$
  3. El algoritmo de aprendizaje modifica  $\hat{f}$  minimizando las diferencias entre los resultados originales y los propiamente generados:  $y_i - \hat{f}(x_i)$ .



- **Paramétricos:** asumen que  $f$  toma una determinada forma funcional y utilizan el conjunto de entrenamiento para estimar los parámetros de dicha caracterización.
  - Ejemplos: “modelo lineal”,  $f(X) = \beta_0 + \sum_{i=1}^k \beta_i X_i$  (otros: LASSO, GAM, etc.)
  - Ventajas:
    - más fáciles de estimar ya que se reduce a estimar un conjunto de parámetros (por ejemplo con mínimos cuadrados ordinarios)
    - más fáciles interpretar (hacer inferencia)
  - Desventajas: al ser más rígidos no son tan buenos para hacer predicciones
- **No Paramétricos:** no hacen supuestos sobre la forma funcional de  $f$ 
  - Ejemplos: árboles de decisión (bagging, boosting), vecinos más cercanos, SVM, etc.
  - Ventajas: mejores para hacer predicciones
  - Desventajas: más costosos de ajustar y difíciles de interpretar

**¿Los modelos más flexibles son siempre mejores para predecir?**

# OVERFITTING Y UNDERFITTING



## Objetivo final: obtener buenas predicciones en datos no observados previamente

- La mayoría de los algoritmos ajusta utilizando observaciones  $(x_i, y_i)$  del conjunto de entrenamiento pero la performance es evaluada con observaciones y etiquetas de un *conjunto de test*!
  - **Overfitting**: el modelo se ajusta en exceso a la data de entrenamiento y no generalizá adecuadamente a datos nuevos
    - Propia de modelos más flexibles que memorizan el “ruido” y no propiedades verdaderas de la función a estimar
  - **Underfitting**: el modelo es demasiado rígido y no captura relevantes con la consecuente mala performance en entrenamiento y test.
- Algunas preguntas pendientes:
  - ¿Qué forma funcional tiene una métrica de performance?
  - ¿Es fácil obtener data de test?
  - ¿Por qué ocurre el fenómeno de overfitting?

# SELECCIÓN DE MODELOS: CONJUNTO DE VALIDACIÓN

- Repitamos nuevamente el objetivo de todo esto... **queremos predecir bien sobre datos desconocidos!**
- ¿Cómo hacerlo? Simulamos la división entre datos conocidos y desconocidos
  - *Enfoque de validation/holdout set*: entrenamos el modelo con datos conocidos y validamos las predicciones con el **conjunto de validación** ("desconocido")
    - El conjunto de validación es una submuestra al azar de observaciones del conjunto de entrenamiento (usualmente 20 %)
    - Problemas: i) predicción de performance tiene alta variabilidad ya que depende de la participación de observaciones en entrenamiento/validación y ii) al achicar el conjunto de entrenamiento podemos estar sobrestimando el error de test



¿Definido el modelo final: qué observaciones usamos para entrenarlo?



- ① Un breve repaso del (necesario) contexto teórico
- ② **Modelos de clasificación en Python**
- ③ Árboles y ensamble de modelos
- ④ Conclusión

## Data Scientist



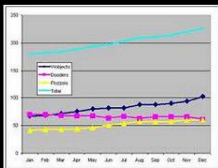
What my friends think I do



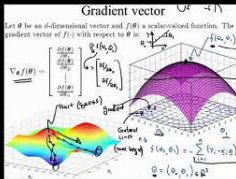
What my mom thinks I do



What society thinks I do



What my boss thinks I do



What I think I do



What I actually do

# EXTRACCIÓN DE DATOS PARA CLASIFICACIÓN



- Objetivo: predecir (un subconjunto) sobrevivientes del Titanic (usando machine learning). Necesitamos datos!

## 1. *Ejercicio 1: extracción de datos del hundimiento del Titanic*

- 1.a Escriba una función que obtenga, a partir de [este enlace](#) y utilizando Pandas, el conjunto de datos de pasajeros involucrados en el hundimiento del Titanic y lo guarde en un archivo `titanic_local.csv`.<sup>†</sup>
- 1.b Agregue lógica de cache que reuse la copia local del conjunto de datos, en caso que esta exista, y evite de ese modo consultar repetidas veces el enlace de arriba.<sup>†</sup>
- 1.c Renombre la columna 'home.dest' a 'home\_dest'.<sup>†</sup>
- 1.d Parta este conjunto de datos de forma aleatoria en dos partes de manera tal que un subconjunto tenga el 70 % de los registros.
- 1.e ¿Cómo puede garantizar que lo hecho en el inciso anterior sea replicable? Fije la semilla del generador de números aleatorios en 1234.



- EDA: Análisis exploratorio de datos (Tukey (1977))
  - Data mining en su acepción pura: técnicas para analizar los datos, encontrar patrones y generar insights
    - Visualizaciones: univariadas (para un mismo feature), bi-variadas (entre features y target), multivariadas (entre distintos features)
    - Técnicas de estadística clásica: correlaciones, test de hipótesis, ANOVA
    - Reducción de dimensionalidad: PCA, SVD
    - Clustering
  - Objetivos:
    - Comprender el dataset
    - Definir y refinar la selección e ingeniería de atributos que alimentan los modelos
- ¿Que es realmente hacer data science en la industria?
  - **E**xtract **T**ransform **F**it **L**oad
  - Distribución de tiempos: E 30 % T 50 % F 10 %, L 10 %

# EDA E INGENIERÍA DE ATRIBUTOS

- El aprendizaje del algoritmo de aprendizaje dependerá de los atributos
  - *Garbage in - garbage out*
- Técnicas diversas para generar/modificar atributos
  - Transformaciones logarítmicas
    - Tiene sentido cuando variables siguen una distribución asimétrica positiva (masa concentrada en valores pequeños y grandes con poca densidad)
  - Reescalamiento de variables numéricas
    - Si el modelo es sensible a la escala del atributo es deseable hacer transformaciones tipo *estandarización*
    - Propenso a hacer **data leakage**: incorporar en los modelos información que no debería estar disponible al momento de puesta en producción
  - Binning
    - Agrupación en *bins* ordenados
  - Transformación de variables categóricas en variables continuas (*one-hot encoding*)

Row Number	Direction
1	North
2	North-West
3	South
4	East
5	North-West



Row Number	Direction_N	Direction_S	Direction_W	Direction_E	Direction_NW
1	1	0	0	0	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0
5	0	0	0	0	1

# DESCRIPCIÓN DE VARIABLES Y PREPROCESAMIENTO

Variable	Descripción	Notas
survived	Condición de supervivencia	1 = Si, 0 = No
pclass	Tipo de ticket	1= Alto, 2= Medio, 3 = Bajo
name	Nombre del pasajero	
sex	Sexo	
age	Edad en años	Fracción si < 1 y xx.5 si estimada
sibsp	# hermanos y conyuges a bordo	
parch	# padres e hijos a bordo	Hijos con niñera tiene parch=0
ticket	# de boleto	
fare	Precio del boleto	
cabin	# de cabina	
embarked	Puerto de embarque	C=Cherbourg, Q=Queenstown, S=Southampton
boat	# bote de rescate	
home.dest	Ciudad de origen	
body	# Número de identificación cadáver	

- En base a esta tabla:

- *Ejercicio 2: Primer preproceso / EDA*

1. Escriba una función que permita convertir (*castear*) múltiples columnas de un tipo de dato a otro y utilícela para asegurar que sus datos sean consistentes con la tabla arriba.<sup>†</sup>
2. ¿Qué variables puede usted ignorar? ¿Por qué? Escriba una función que reciba un dataframe y una lista de columnas a eliminar y devuelva un el data frame sin estas columnas. Logguee cuáles columnas fueron eliminadas.<sup>†</sup>
3. Una los conjuntos de entrenamiento y validación en único dataframe con una columna de booleanos indicando pertenencia al conjunto de entrenamiento. ¿Por qué tiene sentido trabajar con un set de datos de esta forma?

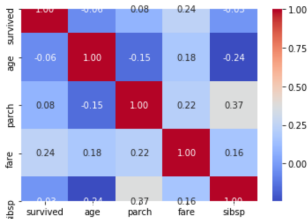
4. ¿Tiene clases desbalanceadas? Haga un gráfico de barras para responder esta pregunta.



# UN POCO MÁS DE EDA

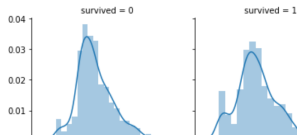
- Vamos a utilizar Seaborn para hacer algunas visualizaciones estadísticas.
  - Veamos primero correlaciones de los atributos numéricos

```
import seaborn as sns
g = sns.heatmap(df.select('survived', 'age', 'parch', 'fare', 'sibsp')
                .toPandas().corr(), annot=True, fmt = ".2f", cmap = "coolwarm")
```



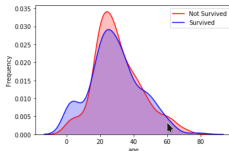
- Solamente el precio del boleto parece estar correlacionado con la probabilidad de supervivencia... ¿debemos descartar las otras variables?

```
g = sns.FacetGrid(df, col='survived')
g = g.map(sns.distplot, 'age')
```



# AUN MÁS EDA (Y EJERCICIOS)

```
survived_age = df.filter((f.col('survived') == 1) &  
                          (f.col('age').isNotNull())).select('age').toPandas()  
  
...  
g = sns.kdeplot(not_survived_age.squeeze(), color='Red', shade=True)  
g = sns.kdeplot(survived_age.squeeze(), color='Blue', shade=True)  
g.set_xlabel('age')  
g.set_ylabel('Frequency')  
g.legend(['Not Survived', 'Survived'])
```



## ● Ejercicio 3: Visualizaciones como herramienta de EDA

1. Use la función `distplot` para graficar la distribución del precio de boletos.
  - 1.a ¿Es la distribución resultante asimétrica/sesgada? En caso afirmativo aplique alguna transformación sobre la variable y grafique la nueva distribución.
2. ¿Son los hombres o las mujeres más propensos a sobrevivir? Grafique la probabilidad de supervivencia para cada caso y compute el valor exacto de las mismas.<sup>†</sup>
3. ¿Hay alguna clase del barco que garantice mayor probabilidad de supervivencia? ¿Es este resultado robusto a controlar por sexo? *Hint*: use la función `catplot`.





- Valores faltantes, repetidos, constantes y extremos
  - ¿Cuál es la frecuencia de valores nulos o faltantes? ¿Vale la pena descartar por esto motivo alguna variable? ¿Imputamos valores? ¿Qué valor usar?
  - ¿Tienen contenido informativo aquellas variable con nula o cuasi-nula varianza?
  - ¿Tiene sentido reemplazar valores extremos?
- *Ejercicio 4: Trabajando con valores nulos y constantes*
  1. Escriba una función que i) para cada atributo compute la proporción de valores nulos y ii) en caso que esta supere un determinado umbral elimine dicha columna
    - 1.a Potencialmente podría borrar información importantes (como el target!). ¿Cómo puede modificar la función anterior para proteger a esta columnas?
  2. En espíritu similar al punto anterior escriba una función que elimine, si las hay, columnas con nula o cuasi nula varianza.<sup>†</sup>
  3. Escriba una función o lógica que permita rellenar valores de atributos numéricos por su mediana. ¿Cómo puede extender esto a atributos categóricos?
    - Hint: le puede primero servir escribir una función auxiliar que identifique las columnas por tipo (categórica o numérica)
  4. Sugiera/piense mejoras sobre el procesamiento hecho en los puntos anteriores.

*Applied Machine Learning is basically Feature Engineering (Andrew Ng)*

- Hay muchas opciones para hacer ingeniería de atributos (por ejemplo extraer el título del nombre)
  - *Ejercicio 5: Ingeniería de atributos*
    1. Cree un nuevo atributo `family_size` con el tamaño de la familia de cada pasajero (incluyendo a el mismo).<sup>†</sup>
    2. Dado este nuevo atributo genere 3 atributos adicionales que remitan a familias de un único miembro, de 2 a 4 miembros y mayor o igual a 5.<sup>†</sup>
    3. Haga uno (o varios) gráficos comparando la probabilidad de supervivencia de cada una de estas familias.
    4. Bonus: Extraiga un prefijo a partir de la columna de boletos siempre y cuando el valor de esa columna no sea numerico. Reemplace esta columna por dicho prefijo.

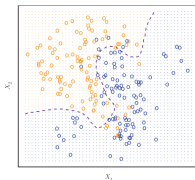
# CLASIFICACIÓN VERSUS REGRESIÓN

- En general los conceptos del universo de problemas de regresión se trasladan al caso de clasificación con pequeñas diferencias
  - Nuestro *target* es ahora una variable discreta o categórica de forma que el objetivo es clasificar etiquetas desconocidas en distintas clases
  - Numerosos problemas pueden modelarse de este modo
    - Rotulado de correo basura, churn, probabilidad de default, etc
  - Queremos que nuestro clasificador estimado,  $\hat{f}$ , minimice ahora la siguiente función

$$\frac{1}{p} \sum_{i=1}^p I(y_i \neq \hat{y}_i)$$

donde  $I(\cdot)$  es una función que vale 1 si  $y_i \neq \hat{y}_i$  y 0 en caso contrario.

- Básicamente deseamos reducir la proporción de errores que cometemos
- Esencialmente estamos intentando buscar una frontera de decisión óptima (en el sentido que minimice el error de arriba)
  - Algunos “clasificadores” conocidos: Bayes ingenuo, k-vecinos más cercanos, regresión logística, etc



# REGRESIÓN LOGÍSTICA

- Supongamos que queremos modelar la probabilidad que un cliente cancele o no su línea de celular

- Podríamos pensar en un modelo de regresión lineal como los ya vistos:

$$Y_t = \beta_0 + \beta_1 X_{t-k} + \varepsilon_t$$

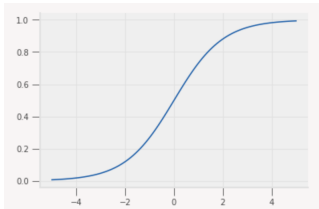
- En base a esto definir un cliente “cancelador” si  $\hat{Y} > 0.5$
- Pero ... el target no caerá necesariamente en el intervalo  $[0, 1]$ .

- Tiene sentido en cambio formular el problema como

$$\Pr(Y_t = 1) = F(\beta_0 + \beta_1 X_{t-k} + \varepsilon_t)$$

donde  $F(\cdot)$  es la función logística dada por  $F(\cdot) = \frac{1}{1+e^{-x}}$

```
>>> x = np.linspace(-5, 5, 100)
>>> y = 1 / (1 + np.exp(-x))
>>> plt.plot(x, y)
```



# MÉTRICAS DE PERFORMANCE

- En general estos modelos devuelven estimaciones de *probabilidades condicionales*.  
¿Cómo sabemos si fueron buenas?
  - Necesitamos alguna métrica de evaluación. Una **matriz de confusión** nos permite definir algunas

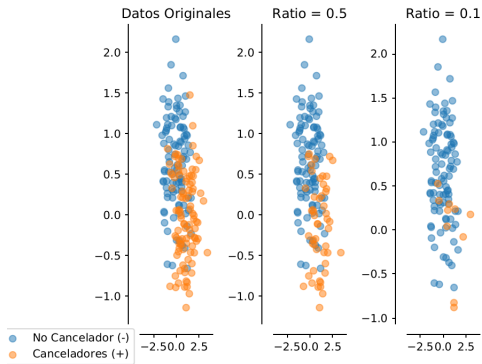
		Valor real y	
		P(1)	N(0)
Valor predicho $\hat{y}$	$\hat{P}(1)$	Verdadero Positivo (TP)	Falso Positivo (FP)
	$\hat{N}(0)$	Falso Negativo (FN)	Verdadero Negativo (TN)

- Hay dos tipos de errores: **falsos positivos** y **falsos negativos**
  - Podemos derivar métricas en base a estos errores
  - La más “natural” se conoce como exactitud o **accuracy** definida por el ratio de clasificaciones correctas sobre el total realizado:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

- ¿Es una buena métrica? Puede ser engañosas...



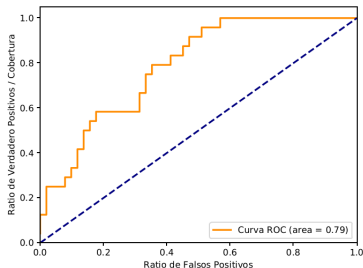


- Imaginemos el caso último panel con una proporción de 1 caso positivo cada 100
  - Un clasificador trivial que prediga siempre la clase negativa tiene una exactitud del 99 %.
  - ¿Podemos construir una métrica para evitar esta situación? Hay que tratar de evitar mezclar los verdaderos positivos y negativos...

- Para sortear los problemas anteriores es frecuente utilizar las siguiente dos métricas
  - **Precisión:** cantidad de casos correctamente rotulados como positivos sobre el total de predicciones positivas,  $TP/(TP + FP)$ .
  - **Cobertura o Recall:** proporción de instancias positivas que el algoritmo logra identificar sobre el total de casos positivos ( $TP/(TP + FN)$ )
  - Depende del contexto puede ser deseable maximizar una o la otra
    - Si es una enfermedad rara, por caso, nos interesa más la cobertura que la precisión. ¿Para un filtro de spam?
  - En determinadas situaciones ambas métricas son importantes y tiene sentido combinarlas

$$F_1 = 2 \frac{p \cdot r}{p + r}$$

- Pesa por igual a ambas métricas (media armónica) y el valor suele estar cerca del mínimo de las métrica
- Esta acotado al intervalo  $[0, 1]$  y vale 1 únicamente para un clasificador perfecto
- Valores superiores a 0.7 son propios de un “buen” clasificador
- Desventaja: no hace un juicio sobre como el modelo clasifica las instancias negativas



## ● Características:

- Para definir pertenencia a una clase hay un umbral,  $p$ , tal que si  $p = 0$  (1) siempre (nunca) predigo que las observaciones son positivas entonces ambos TPR (TP / P) y FPR (FP / N) valen 1 (0)..
- Curva 45 grados define a un clasificador aleatorio y la coordenada (0,1) a un clasificador perfecto
- Intuitivamente si elegimos aleatoriamente una observacion positiva y una negativa el area bajo la curva es la probabilidad de que el clasificador rankee más alto al positivo

**Conviene utilizar una única métrica para guiar las decisiones**



- Estamos casi listos para predecir sobrevivientes usando una regresión logística
  - *Ejercicio 6: Fit Regresión Logística*
    1. Alguno algoritmos, en Python, soportan exclusivamente atributos numéricos. Escriba entonces una función que utilice *StringIndexer* para convertir todos los atributos categóricos.  
Bonus: ¿Tiene sentido usar esta codificación para atributos que carecen de orden? En caso contrario pruebe emplear *One-Hot-Encoding*.
    2. Fitee una regresión logística sobre la data de entrenamiento y genere predicciones (y probabilidades) para la data de test/validación.
    3. Evalúe estas predicciones usando métricas de *accuracy* y ROC para el conjunto de entrenamiento y ROC para el conjunto de validación.<sup>†</sup>
    4. Bonus: utilice *scikit-learn* o escriba una función que compute la métrica F1 y con ello evalúe nuevamente los resultados para ambos conjuntos.

# SOBRE TRAIN, VALIDATION, TEST SPLIT

- ¿Podemos overfittear el conjunto de validación?
  - Si realizamos muchas pruebas posiblemente si
  - Por eso en la práctica trabajamos con 3 conjuntos: entrenamiento, validación y testeo

Data Permitting:



- En general se reserva 20 % para testing y el remanente se divide en 80 % entrenamiento y 20 % validación
- Con muchos datos los porcentajes son menores: es una cuestión de representatividad y no de números



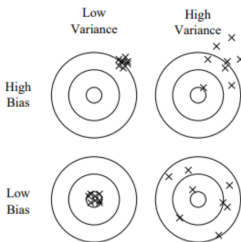
# TRADEOFF DE SESGO-VARIANZA

- ¿Cuál es el valor esperado de ECM en una observación de testeo?
  - Es decir cuál es el ECM promedio si uno estima sucesivas veces  $f$  usando muchos conjuntos de entrenamiento y evaluando en cada  $x_0$  del conjunto de test:

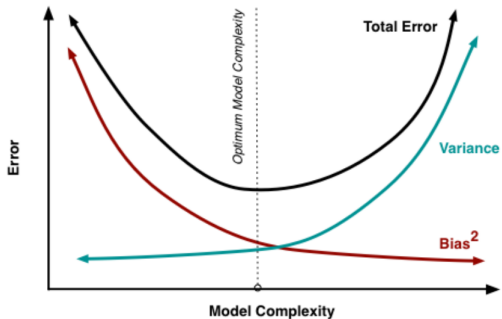
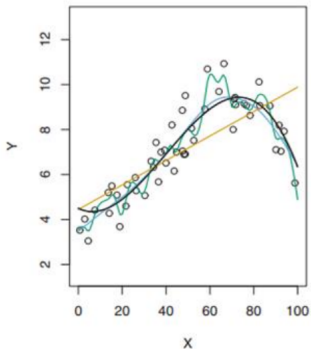
$$E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + [\text{Sesgo}(\hat{f}(x_0))]^2 + \text{Var}(\varepsilon)$$

donde

- La varianza remite a cuánto cambiaría  $\hat{f}$  si estimáramos con otro conjunto de entrenamiento (idealmente queremos que sea poco)
- El sesgo alude a si  $\hat{f}$  está errando consistentemente en las predicciones ( $\text{Sesgo}(\hat{f}(x_0)) = E[\hat{f}(x_0)] - y_0$ ).
- El ideal: tener un sesgo bajo y una varianza baja. ¿Es esto posible?



# TRADEOFF SESGO-VARIANZA II



- ¿Por qué hay un tradeoff?

- Es fácil obtener un método con bajo sesgo y alta varianza (dibujando una curva que pase por todos los puntos)
- Es fácil obtener un método con bajo o nula varianza y alto sesgo (fitteando una constante)

- En general vale lo siguiente

- Métodos más complejos tienen alta varianza y bajo sesgo
- El fenómeno de *overfitting* se asocia a escenarios justamente de alta varianza y bajo sesgo

# AGENDA

---

① Un breve repaso del (necesario) contexto teórico

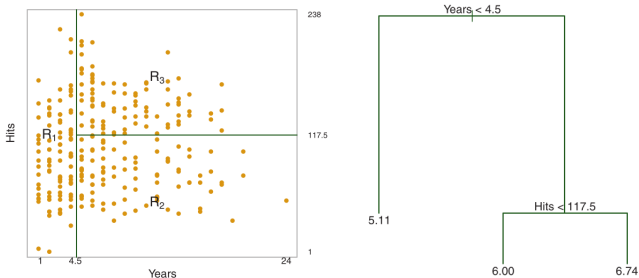
② Modelos de clasificación en Python

③ Árboles y ensamble de modelos

④ Conclusión

# ÁRBOLES DE DECISIÓN

- Característica central: particionan el espacio de atributos en regiones
  - Cada región está definida por el cumplimiento de alguna regla
  - Particionar de esta manera define una **estructura de árbol** (con nodos terminales o hojas, nodos internos y ramas (aristas)) por el cumplimiento de alguna regla



- ¿Cómo podemos utilizar esto para problemas de clasificación?
  - Una observación dentro de una determinada región pertenecerá a la clase más frecuente en esa región: ¿cómo elegimos las regiones?

# PARTICIONAMIENTO RECURSIVO BINARIO

- Proceso iterativo que funciona del siguiente modo

1. Se fija un criterio para hacer cortes

- En clasificación resulta natural usar el *error de clasificación* de cada región: para una región  $m$  con  $N_m$  observaciones la proporción de observaciones que no pertenecen a la clase  $k$  resulta ser

$$EC = \frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k)$$

2. Para el primer nodo seleccionamos el predictor y el punto de corte tal que particionar el espacio de atributos en dos regiones, genere la mayor reducción en el error de clasificación.
3. Para los nodos de decisión siguientes repetimos el proceso en cada (sub)región resultante hasta llegar a algún criterio de parada

- Ventajas:

- Fáciles de interpretar si no son muy profundos
- Excelente manejo de no linealidades
- Sencillo obtener un resumen de la importancia de cada atributo

- Desventajas:

- Poco sesgo (decreciente en profundidad) pero altísima varianza (basta considerar partir el conjunto de entrenamiento de forma aleatoria en mitades)

- Bagging:

- En general vale que agregar observaciones reduce la varianza
  - Podemos crear  $B$  conjuntos de entrenamiento muestreando de forma aleatoria (y con reposición) del conjunto entrenamiento original (*bootstrapping*).
- Si luego entrenamos sobre cada conjunto y promediamos reducimos la varianza (y tenemos bajo sesgo)

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

- La predicción final es la clase más votada por los  $B$  árboles (regla mayoritaria)
- Random Forests: ideado por Breiman con el objeto de mejorar aun más la performance predictiva
  1. Al igual que en bagging se crean  $B$  conjuntos de entrenamiento muestreando de forma aleatoria (y con reposición) del conjunto entrenamiento original.
  2. Para cada conjunto se hace particionamiento recursivo pero al elegir los cortes se emplea un subconjunto aleatorio de la totalidad de atributos (*descorrelacionar*)



# EJERCICIOS ÁRBOLES





## ● Ejercicio 7: Árboles de decisión y aleatorios

1. Fittee un árbol de decisión en vez de una regresión logística y compare la performance de los modelos.<sup>†</sup>
  - Bonus: Grafique el árbol de decision resultante
2. Repita el punto anterior pero ahora con un bosque aleatorio.<sup>†</sup>
  - Bonus: Pruebe aumentar la profundidad del árbol. ¿Mejora su métrica de performance?
3. Compute y grafique la importancia de atributos.



- ① Un breve repaso del (necesario) contexto teórico
- ② Modelos de clasificación en Python
- ③ Árboles y ensamble de modelos
- ④ **Conclusión**

- Tenemos una mejor idea de data science y en particular machine learning con Python y Spark.
- Sin embargo... hay muchas cosas que (obviamente) no vimos
  - No vimos aprendizaje no supervisado ni reinforcement learning
  - ... ni tampoco un ejemplo de regresión
  - Prácticamente no hemos trabajado con datos fechados!
    - El mecanismo de train/test split debe tener una noción temporal
  - Hay otros métodos de selección de atributos
    - Árboles no profundos y hacer feature importance para identificar features o Lasso sobre regresión.
  - Técnicas de reducción de dimensionalidad
  - Técnicas de corrección de desbalance e clases
  - Tuneo de hiperparámetros

-  EFRON, B. y T. HASTIE (2016), *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*, New York, NY, USA: Cambridge University Press
-  JAMES, G., D. WITTEN, T. HASTIE y R. TIBSHIRANI (2013), *Introduction to Statistical Learning : with Applications in R*, New York, NY: Springer
-  MITCHELL, T. M. (1997), *Machine Learning*, New York, NY: McGraw-Hill. [Vea p. 3]
-  TUKEY, J. W. (1977), *Exploratory Data Analysis*, 7.<sup>a</sup> ed., Reading, Massachusetts: Addison Wesley. [Vea p. 12]