

clusterAI 2021

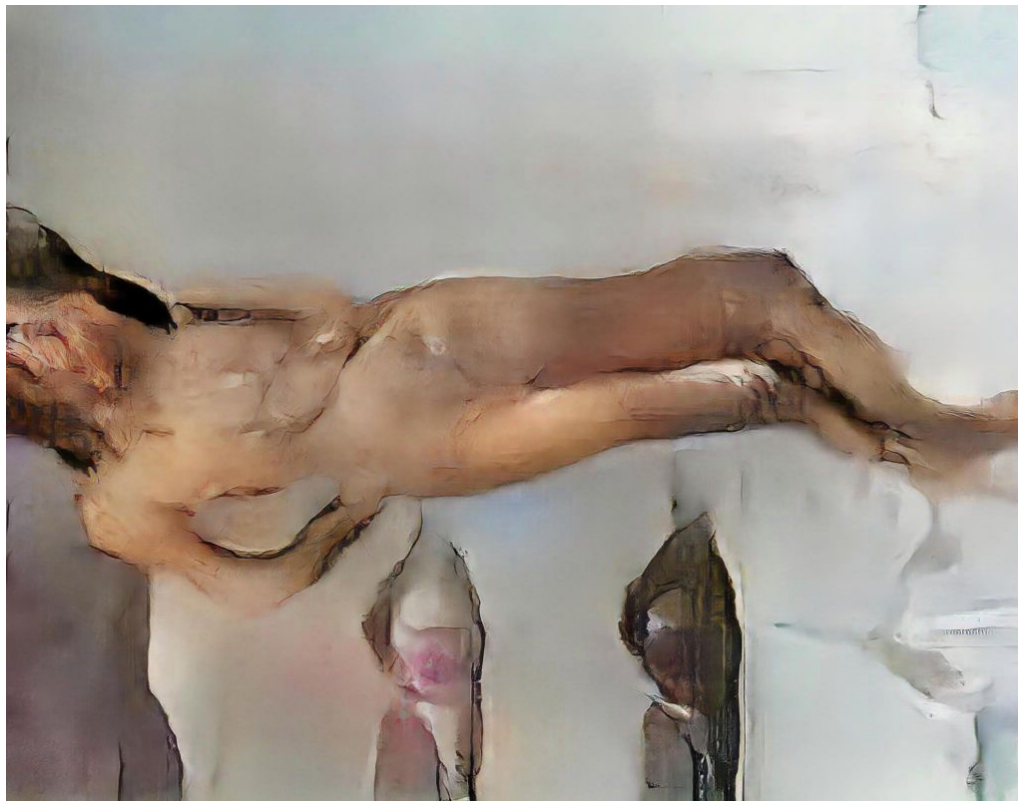
ciencia de datos en ingeniería industrial

UTN BA

curso I5521

clase_02: bayes theorem & pre-processing

AI & Art: Mario Klingemann



'My Artificial Muse'
<http://quasimondo.com/>

Probability density functions

- Bayes theorem
- Likelihood
- Maximum likelihood

Preprocessing

- Python: For, IF, functions
- Intro Scikit-Learn
- Categorical Variables: Dummies
- Feature Processing

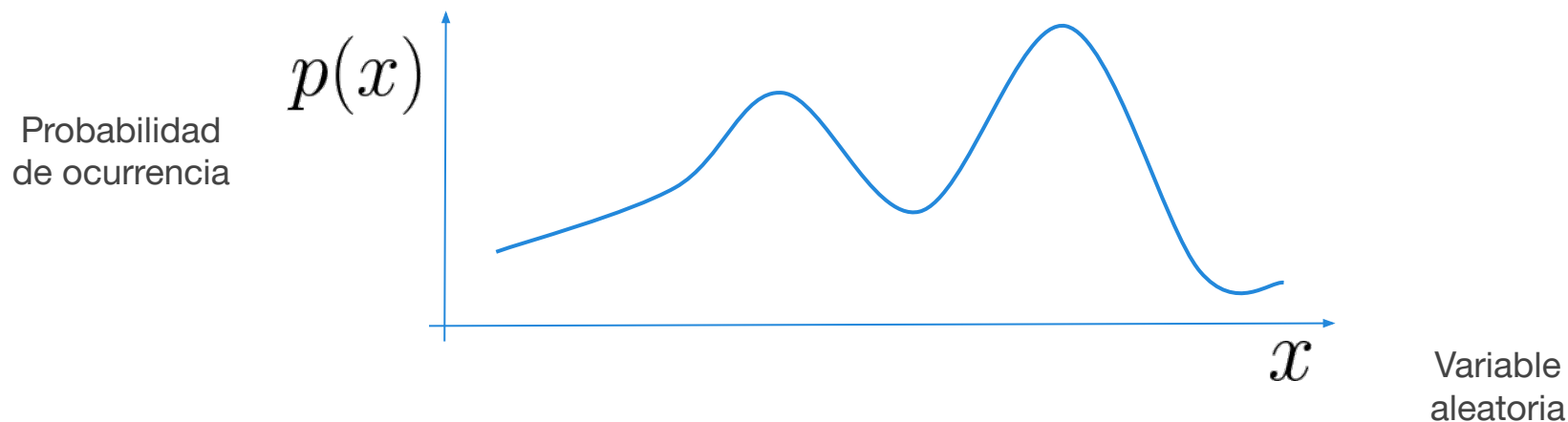
Python lab

- EDA aceros
- EDA subtes

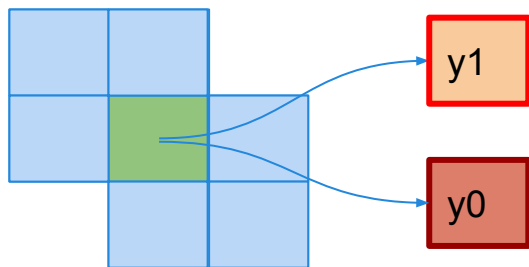
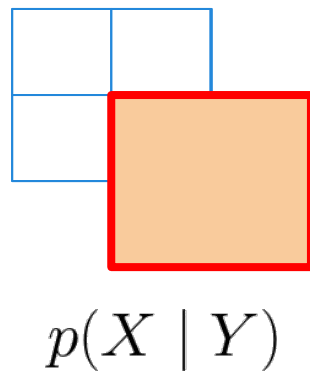
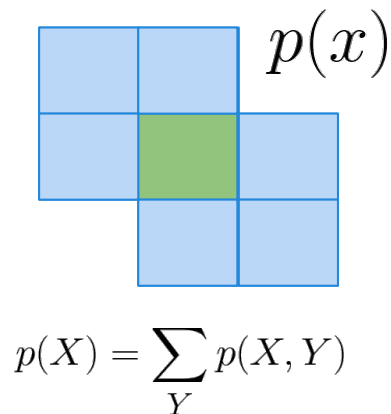
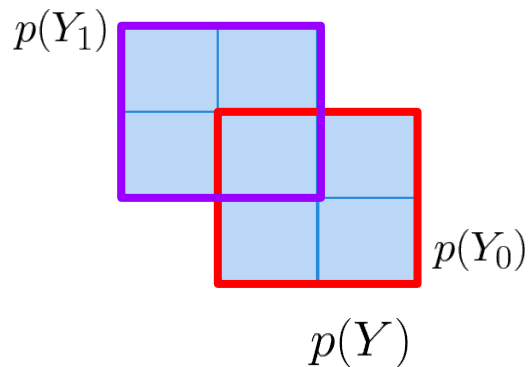
Probability density functions

Distribución de probabilidad

La distribución de probabilidad es la **función** que asigna probabilidades de ocurrencia a distintos estados posibles de un experimento [1]. Es la **descripción** de un fenómeno **aleatorio** en términos de un espacio de muestreos y probabilidades de eventos.



Teorema de Bayes



???

$$p(Y | X) = \frac{p(X | Y)p(Y)}{p(X)}$$

Supongamos un conjunto X compuesto por dos clases/conjuntos Y_1 e Y_0 .

Teorema de Bayes

$$p(Y \mid X) = \frac{p(X \mid Y)p(Y)}{p(X)}$$

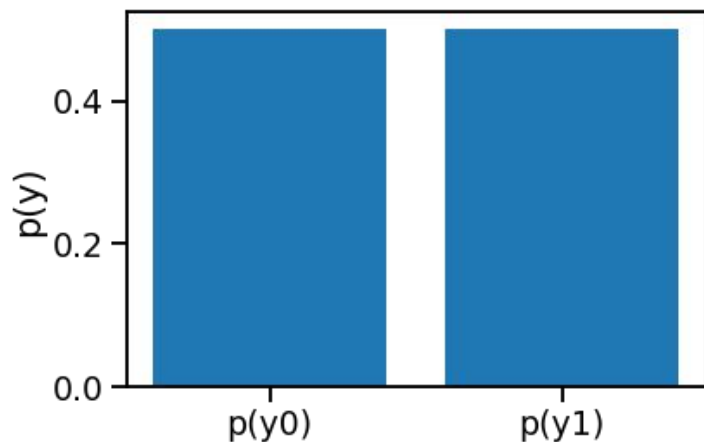
$p(Y)$: prior

$p(x \mid Y)$: likelihood

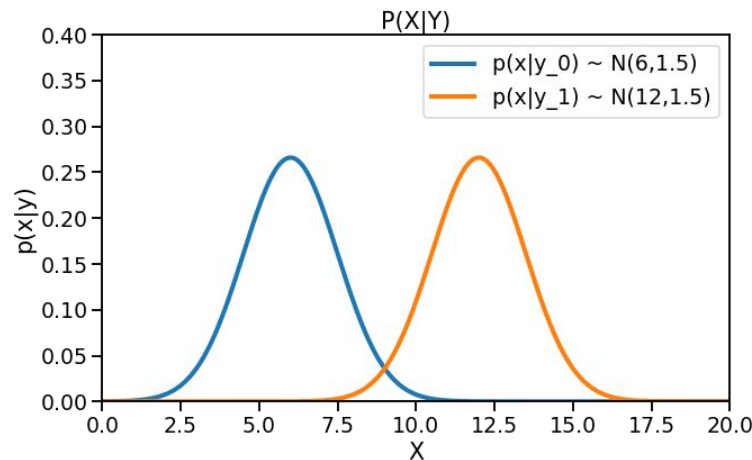
$p(Y \mid X)$: posterior

$p(X)$: Densidad de toda la muestra

Teorema de Bayes



$$p(Y)$$

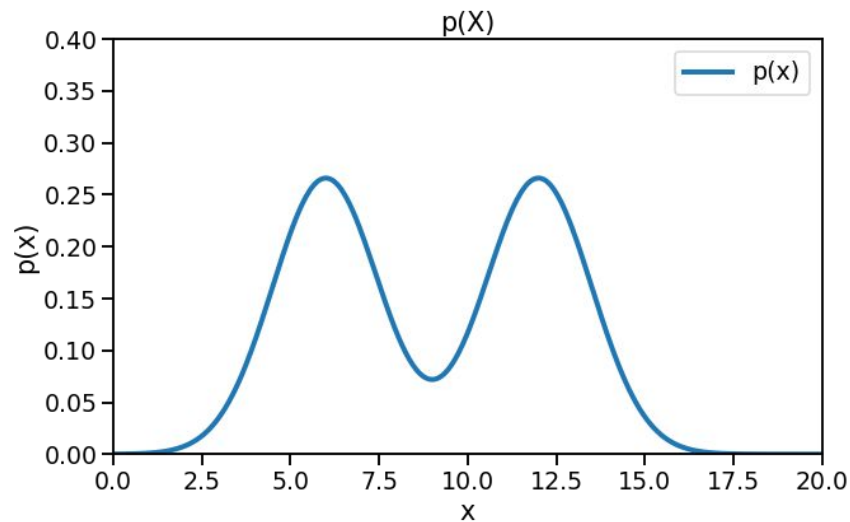


$$p(X | Y)$$

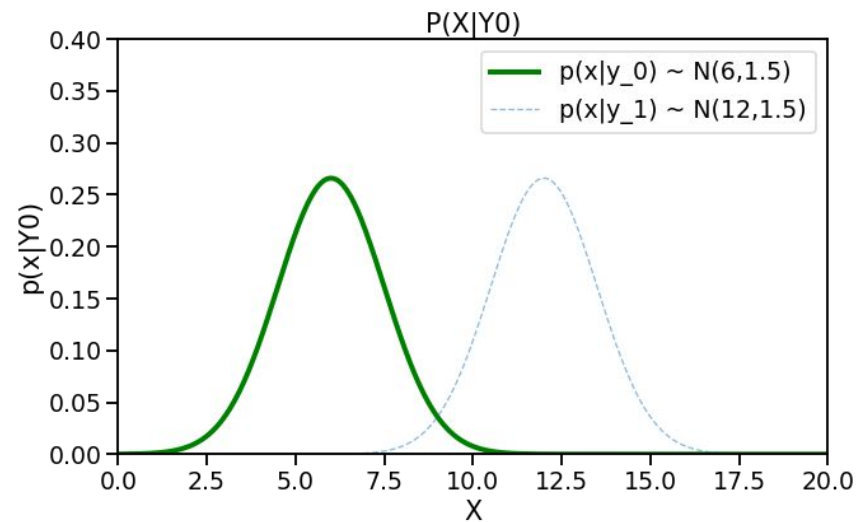
$$p(x | Y_i) \sim N(\mu, \sigma)$$

Supongamos que tenemos un set de muestras provenientes de $p(X)$ y es mismo esta compuesto por dos clases Y_1 & Y_2 , cada clase/conjunto generada por una distribución de probabilidad gaussiana.

Teorema de Bayes



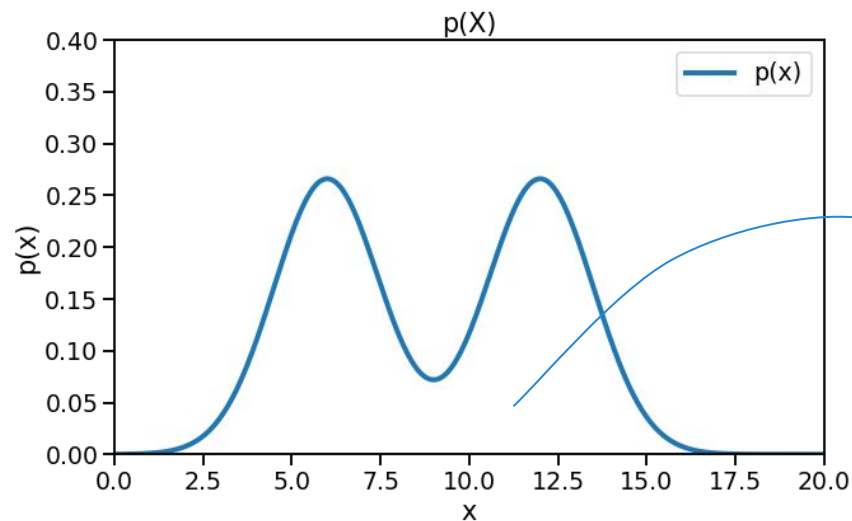
$$p(X) = \sum_Y p(X, Y)$$



$$p(X | Y)$$

Sabemos que $p(X)$ es la sumatoria/union de todas las densidades de probabilidades de intersección (fig. izq) y que podemos condicionar $p(X)$ con Y (fig. der).

Teorema de Bayes

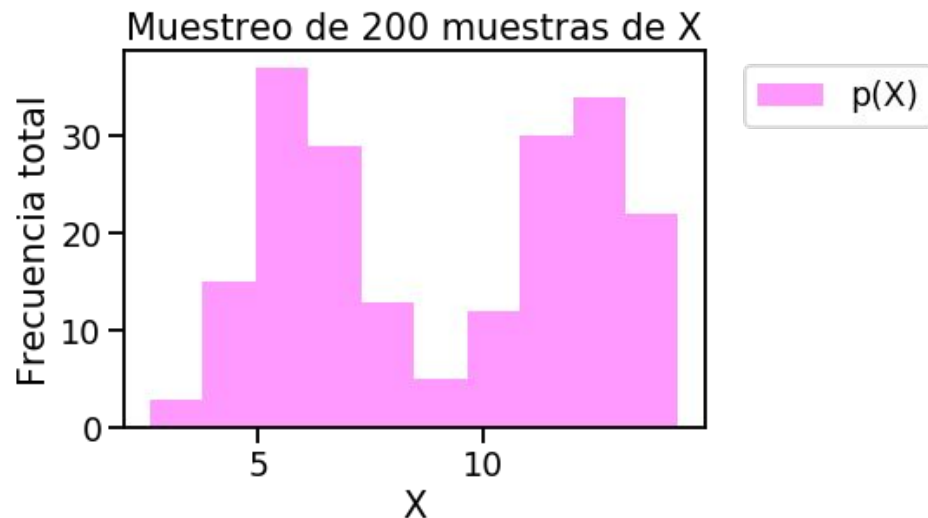
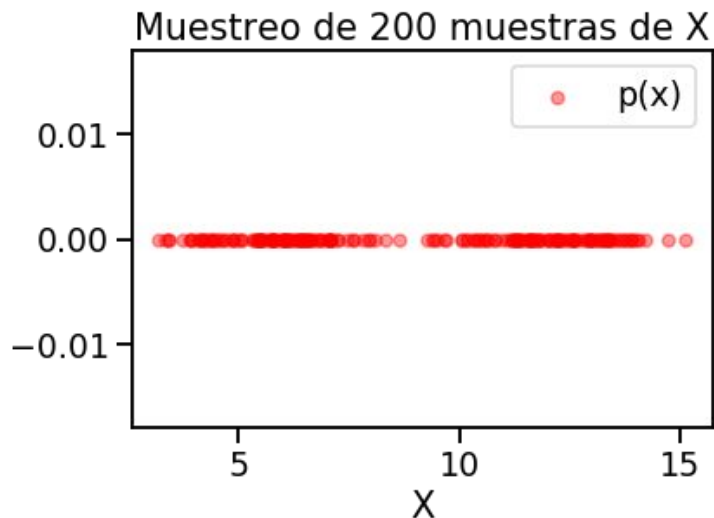


???

$$p(Y | X) = \frac{p(X | Y)p(Y)}{p(X)}$$

Conociendo la densidad de X , la probabilidad prior de cada clase y el likelihood (densidad) de cada clase podemos estimar la probabilidad posterior: la proba que dado X tengamos Y_1 o Y_2 .

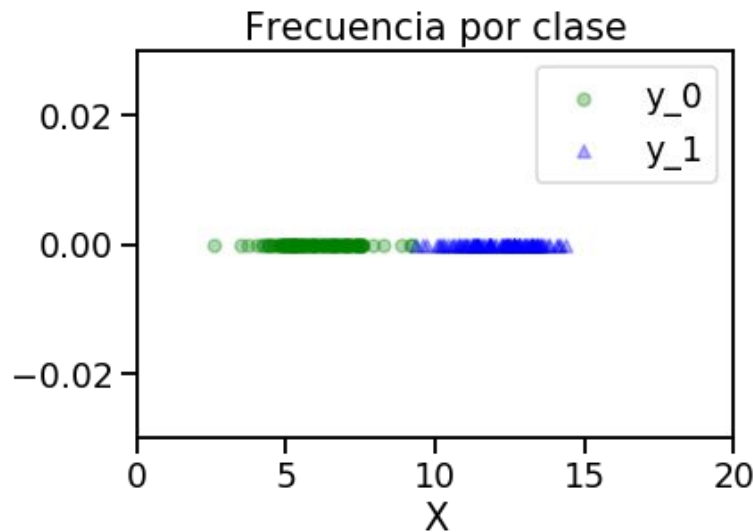
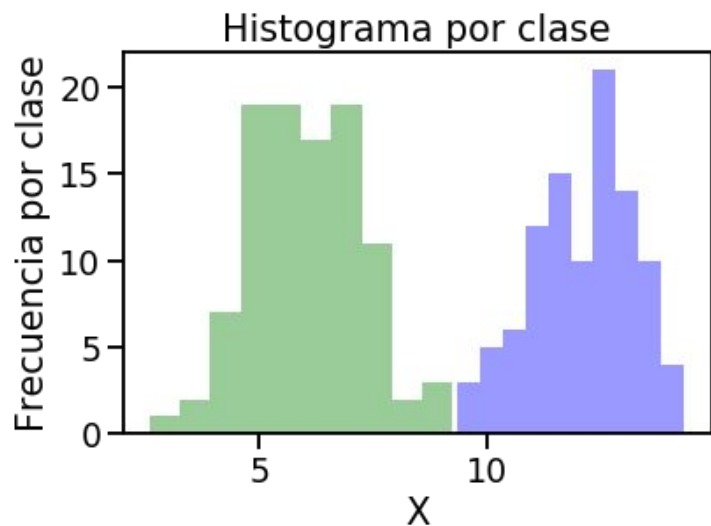
Teorema de Bayes



$$p(X)$$

Empíricamente si sampleamos/muestreamos 200 muestras de X obtendremos un set de datos que formará una densidad como se muestra en la figura. X esta compuesta por 200 instancias muestreadas de una distribución de probabilidad formada por la suma de dos gaussianas.

Teorema de Bayes



$$p(x | Y_0) \sim N(\mu = 6, \sigma = 1.5)$$

$$p(x | Y_1) \sim N(\mu = 12, \sigma = 1.5)$$

Empíricamente si condicionamos X con colores cada clase Y_i podremos realizar un histograma de frecuencias separado por cada una.

Teorema de Bayes

$$p(Y | X) = \frac{p(X | Y)p(Y)}{p(X)}$$

$$p(X | Y_i) \sim N(\mu_i, \sigma_i)$$

$p(Y)$: prior

$p(x | Y)$: likelihood

$p(Y | X)$: posterior

$p(X)$: Densidad de toda la muestra

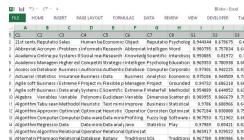
Si suponemos que conocemos la distribución de probabilidad de cada clase y su respectivo prior entonces podríamos obtener la probabilidad posterior: dada una muestra X cual es la probabilidad de que pertenezca a Y_1 o Y_2 . En este ejemplo el likelihood puede ser conocido como una distribución normal, aunque en muchos casos el likelihood no es paramétrico/conocido y podemos estimarlo con un histograma.

Preprocessing: cómo manejar datos tabulares

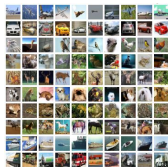
Pre-procesamiento de datos



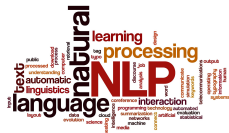
Grafos



Tablas



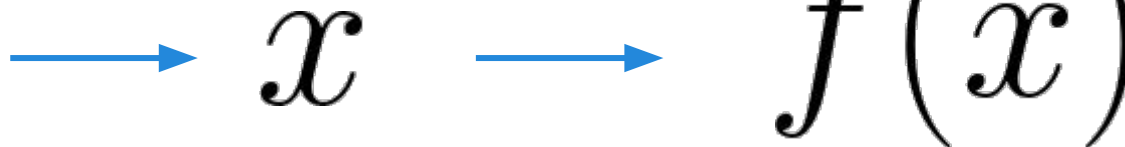
Imágenes



NLP

Pre-procesar para obtener x

Aprender f desde los datos x

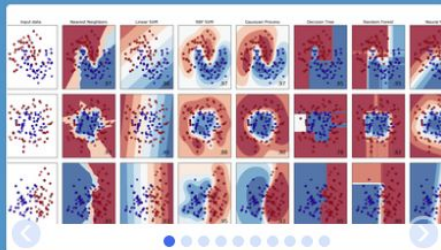


Los datos como son registrados y almacenados originalmente no siempre están en condiciones de ser utilizados para un análisis exploratorio ni tampoco para ser usados en un modelo de aprendizaje. Por eso los datos deben ser pre-procesados. Por el momento vamos a enfocarnos en el pre-procesamiento de datos tabulares (tablas).



Scikit Learn <https://scikit-learn.org/>

Intro scikit-learn



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

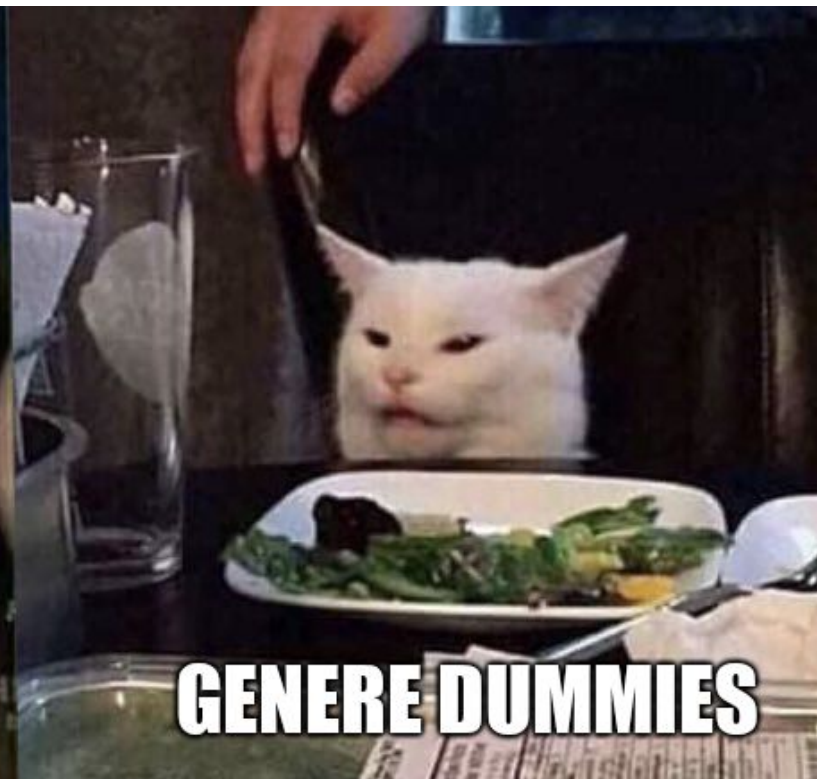
Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

Categorical Variables: Dummies



Categorical Variables (Dummies)


Edad	Altura	Sexo
18	1.70	Masculino
24	1.60	Femenino
30	1.90	Femenino
28	1.5	Masculino



Edad	Altura	Sexo	Masculino	Femenino
18	1.70	Masculino	1	0
24	1.60	Femenino	0	1
30	1.90	Femenino	0	1
28	1.5	Masculino	1	0

Cuando las variables/features/dimensiones toman valores categóricos podemos transformarlas para obtener una nueva variable que tome valores binarios por cada categoría existente. Estas nuevas variables son conocidas como dummies.

Categorical Variables (Dummies)



```
# 1 Creamos un dataframe
raw_data = {'edad': [18, 24, 30, 28],
            'altura': [1.7, 1.6, 1.9, 1.5],
            'sexo': ['masculino', 'femenino', 'femenino', 'masculino']}
data = pd.DataFrame(raw_data, columns = ['edad', 'altura', 'sexo'])

# 2 Creamos un dataframe de variables Dummies para columna "Sexo"
df_sexo = pd.get_dummies(data['sexo'])

# 3 Agregamos estas nuevas variables dummies a nuestro dataframe
df_new = pd.concat([df, df_sex], axis=1)
```

Feature scaling & normalization



Normalization

En muchas ocasiones las features pueden tener rangos muy distintos. Por ejemplo, si utilizamos metros cuadrados y temperatura para caracterizar las condiciones climáticas de un campo, la primer variable estará en el rango de decenas de miles y la segunda en decenas. Esta diferencia de escalas puede generar un problema a la hora de ‘aprender de datos’.

Para resolverlo, abordaremos dos estrategias de pre-procesamiento de features:

- Standardization [1]
- Min-Max normalization [2]

[1] van den Berg, R. A., Hoefsloot, H. C., Westerhuis, J. A., Smilde, A. K., & van der Werf, M. J. (2006). Centering, scaling, and transformations: improving the biological information content of metabolomics data. *BMC genomics*, 7(1), 142.


[2] Jain, Y. K., & Bhandare, S. K. (2011). Min max normalization based data perturbation method for privacy protection. *International Journal of Computer & Communication Technology*, 2(8), 45-50.

Feature Engineering: Standardization

El método de “standardization” (o “Z-score normalization”) transforma una feature para que tenga media $\mu=0$ y desviación standard $\sigma=1$. Por esta razón, en el caso que los datos tengan una distribución gaussiana, los datos transformados tendrán una distribución normal standard. Esta última es una propiedad que puede mejorar significativamente la performance de un modelo.

$$x_i' = \frac{(x_i - \mu)}{\sigma}$$

Auto-Scaling



```
from sklearn.preprocessing import StandardScaler
data = [[0, 0], [0, 0], [1, 1], [1, 1]]
scaler = StandardScaler()
print(scaler.fit(data))
    "StandardScaler(copy=True, with_mean=True, with_std=True)"
print(scaler.mean_)
    "[0.5 0.5]"
print(scaler.var_)
    "[0.25 0.25]"
print(scaler.transform(data))
    "[[-1. -1.] [-1. -1.] [ 1.  1.] [ 1.  1.]]"
```



Feature Engineering: Min-Max normalization

El método de “Min-Max normalization” afecta al valor de la feature en cada sample por el mínimo de la feature y lo divide por el rango entre máximo y mínimo.

$$x'_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

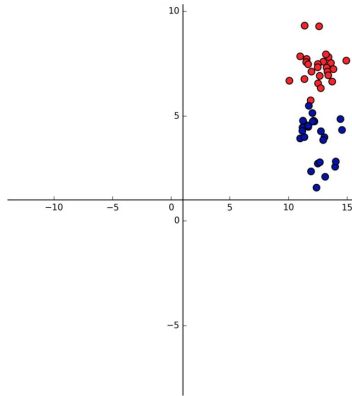
Cada feature después de pre-procesarla quedará un mínimo en 0 y un máximo en 1.

Feature Engineering: Min-Max normalization

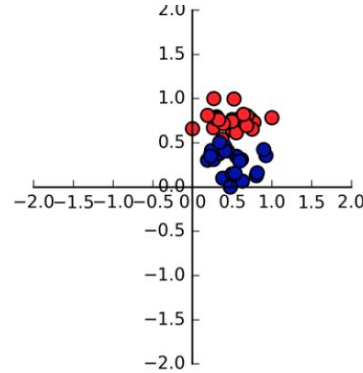


```
from sklearn.preprocessing import MinMaxScaler
data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
print(scaler.fit(data))
    "MinMaxScaler(copy=True, feature_range=(0, 1))"
print(scaler.data_max_)
    "[ 1. 18.]"
print(scaler.transform(data))
    "[[0. 0.], [0.25 0.25], [0.5 0.5], [1. 1.]]"
print(scaler.transform([[2, 2]]))
    "[[1.5 0.  ]]"
```

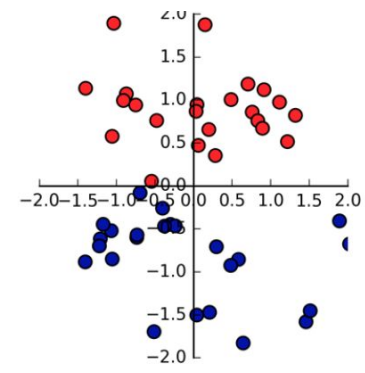
Datos originales



Min-Max Scaler



Standard Scaler



Preprocessing: Min-Max scaling

IMPORTANTE: cuando pre-procesamos las features de un dataset debemos conservar el objeto “scaler” que contiene la información para transformar features. Esto quiere decir que a nuevos datos debemos transformarlos con el scaler ajustado con los datos iniciales y evitar realizar todo el proceso de nuevo con los datos viejos y nuevos.

Pandas: Concat, Join, Merge

Pandas nos da la opción de poder combinar dataframes de distintas formas:

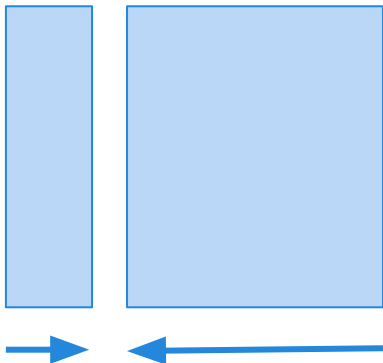
- **Concat**, unir dos dataframes por columnas o filas
- **Join & Merge** (vlookup)

Pandas: Concat

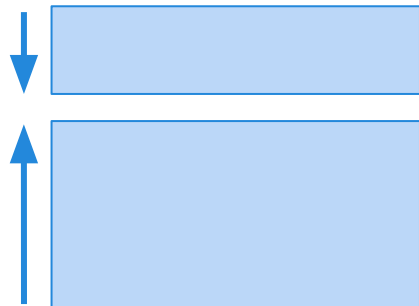
Podremos concatenar dos dataframes por columnas o por filas. Esto quiere decir que si concatenamos por:

- Columnas: la cantidad de filas de ambos tiene que ser igual
- Filas: la cantidad de columnas de ambos tiene que ser igual

Concatenar
por columnas



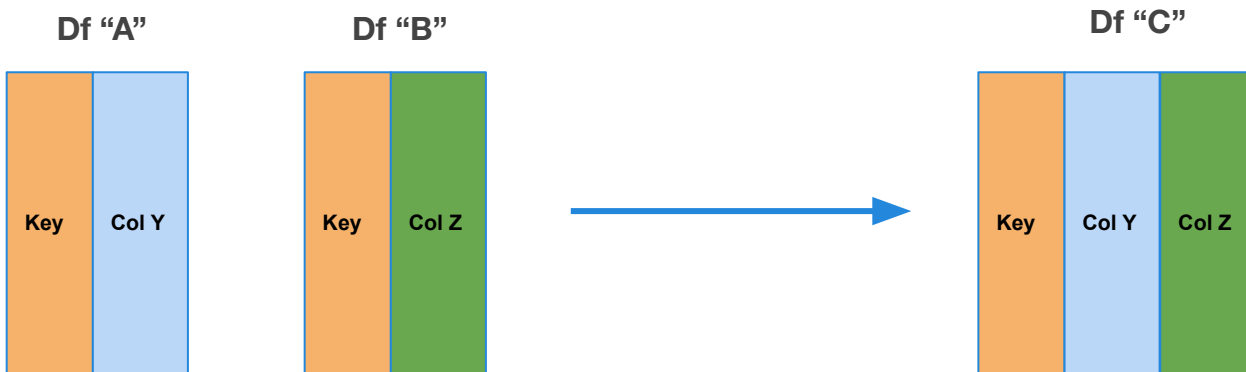
Concatenar
por filas



Si la cantidad de filas o columnas no son iguales dependiendo el caso pandas generará nuevas columnas y filas para satisfacer la desigualdad y estas estarán llenas con NaNs.

Pandas: Join & Merge

Es lo más cercano al “vlookup” en excel. Esto permite poder tener una columna “key” de referencia en dos tablas (A y B). Permite llevar los datos de B asociados a “key” a la tabla A asociándolos a “key” también.



Pandas: Merge

```
In [39]: left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
....:                        'A': ['A0', 'A1', 'A2', 'A3'],
....:                        'B': ['B0', 'B1', 'B2', 'B3']})
....:

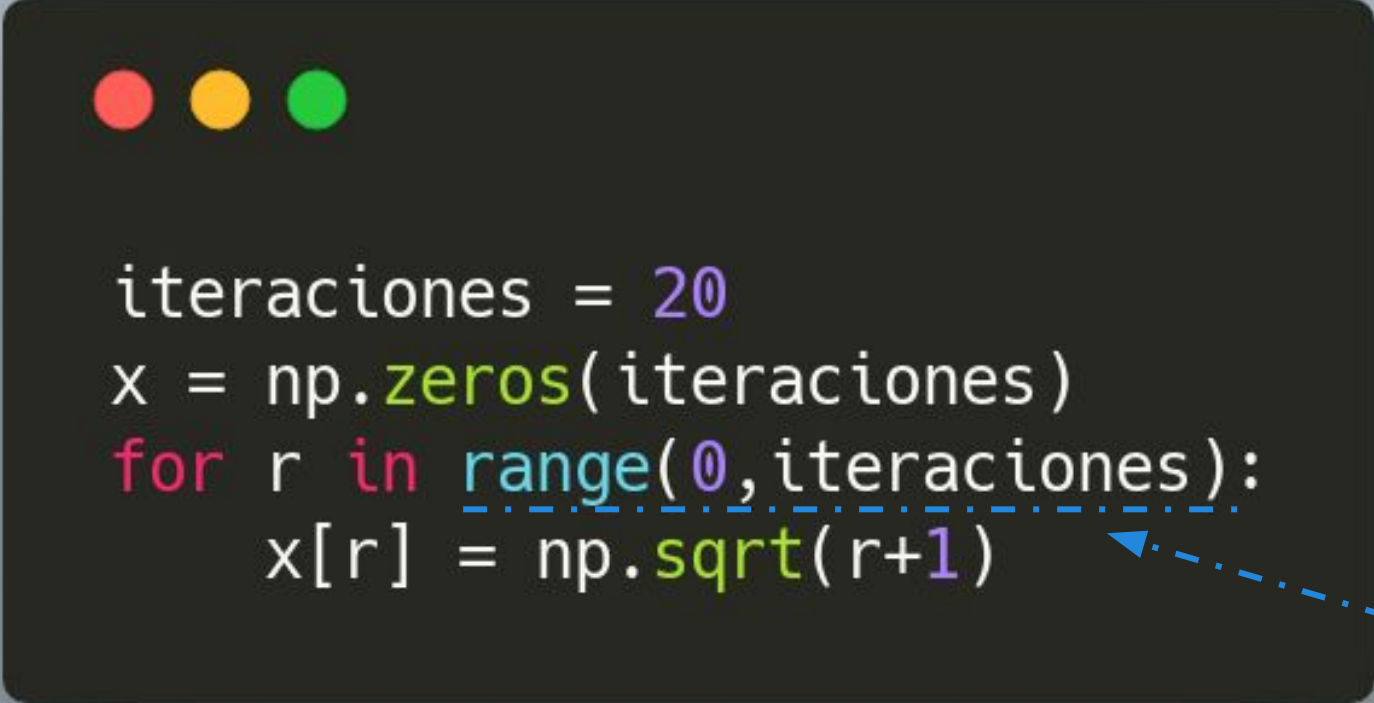
In [40]: right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'],
....:                          'C': ['C0', 'C1', 'C2', 'C3'],
....:                          'D': ['D0', 'D1', 'D2', 'D3']})
....:

In [41]: result = pd.merge(left, right, on='key')
```

left				right				Result					
	key	A	B		key	C	D		key	A	B	C	D
0	K0	A0	B0	0	K0	C0	D0	0	K0	A0	B0	C0	D0
1	K1	A1	B1	1	K1	C1	D1	1	K1	A1	B1	C1	D1
2	K2	A2	B2	2	K2	C2	D2	2	K2	A2	B2	C2	D2
3	K3	A3	B3	3	K3	C3	D3	3	K3	A3	B3	C3	D3

Ciclos for, funciones y condiciones IF.

'for' loops in python



```
iteraciones = 20
x = np.zeros(iteraciones)
for r in range(0, iteraciones):
    x[r] = np.sqrt(r+1)
```

iterador
(iterator)



'if' statements in python



```
if x > 1 :  
    x = pd.concat([data1,data2])  
  
else:  
    x = data1
```

'if' statements in python



```
if y == "mean":  
    mean = np.mean(data.distance)  
  
elif y == "Preproc":  
    nans = data.isnull().any()  
  
elif y == "std dev":  
    std_dev = np.std(data.distance)
```

functions in python



```
def dot_product(x1,x2):  
    "Esta función calcula el producto interno entre dos vectores"  
    dotprod = np.dot(x1,x2.T)  
    return dotprod  
  
a = dot_product(x_febrero,x_marzo)
```

A agarrar la PyLA

