Día 4 - 18/11/21 (Jueves)

Afrontando

Problemas de la vida real

Lic. Ronaldo Armando Canizales Turcios

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas

Agenda Día 4

Bloque A

- Ingeniería de características: cancelación de préstamos.
- Datos desbalanceados: detección de fraude.
- Tratam. de series temporales: enfoques para predicción.

Bloque B

- Reducción de dimensionalidad: Introducción al aprendizaje no supervisado.
- Aplicando análisis de componentes principales. **[Práctica]**

Agenda Día 5

Bloque A

- Aprendizaje no supervisado: clusterización.

- Agrupación no supervisada de perfiles económicos: k-medias.

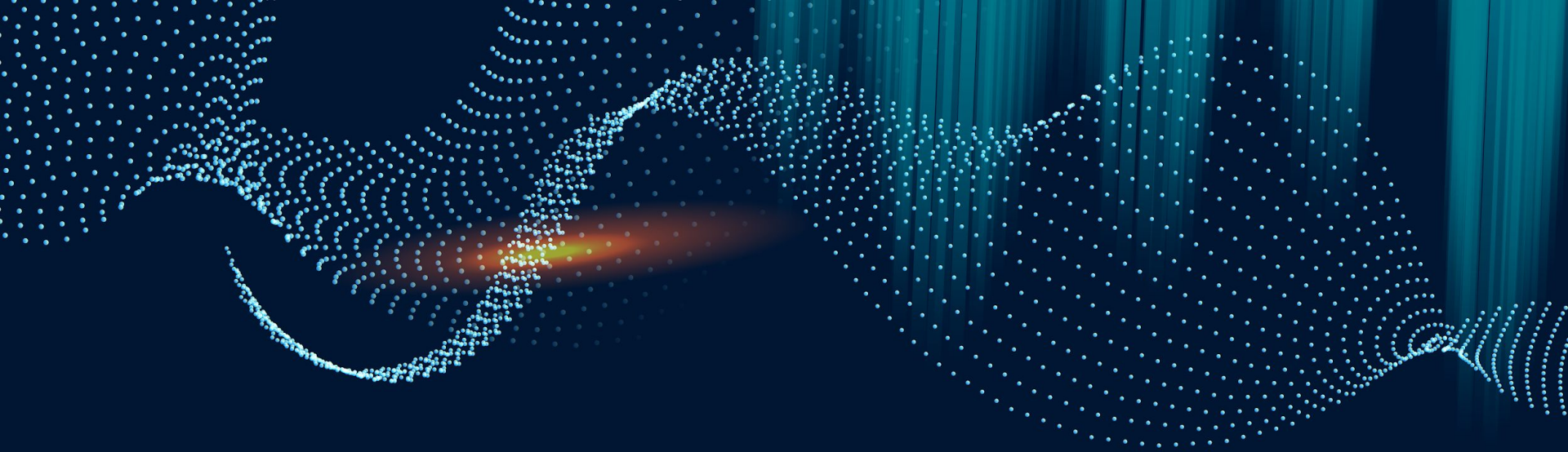
- Clusterización utilizando datos en tiempo real.

Bloque B

- Aplicando clusterización mediante k-medias. **[Práctica]**

- Métodos de ensamble: modelos y conceptos.

- Cierre del curso: concurso (rifa de libros) y siguientes pasos.



01 Ingeniería de características

Cancelación de préstamos

Se trata de un problema de clasificación.

Una cancelación es una deuda que un acreedor ha dejado de intentar cobrar después de que el deudor no haya realizado pagos durante varios meses.

La variable objetivo es “Charge-Off”, que tomará un valor de 1 en caso de cancelación de préstamo y 0 en caso contrario.

El conjunto de datos con el que trabajaremos contiene más de un millón de registros del año 2017.
Cada registro contiene 150 variables de diversa índole. Nuestro trabajo será aplicar ingeniería de características a estos datos.

Vistazo a la variable objetivo

```
dataset['loan_status'].value_counts(dropna=False)
```

Current	788,950
Fully Paid	**646,902**
Charged Off	**168,084**
Late (31-120 days)	23,763
In Grace Period	10,474
Late (16-30 days)	5,786
NaN	23

Fully Paid	0.793758
Charged Off	0.206242

```
dataset = dataset.loc[dataset['loan_status'].isin(['Fully Paid', 'Charged Off'])]

dataset['loan_status'].value_counts(dropna=False)

dataset['loan_status'].value_counts(normalize=True, dropna=False)
```

Eliminando carac. que tienen más de 30% de valores faltantes (NaN)

```
missing_fractions = dataset.isnull().mean().sort_values(ascending=False)

missing_fractions.head(10)
```

```
next_pymnt_d                                  1.000000
member_id                                     1.000000
orig_projected_additional_accrued_interest    0.999876
sec_app_mths_since_last_major_derog           0.999628
hardship_dpd                                  0.999275
hardship_reason                               0.999275
hardship_status                               0.999275
deferral_term                                 0.999275
hardship_amount                               0.999275
hardship_start_date                           0.999275
```

```
drop_list = sorted(list(missing_fractions[missing_fractions > 0.3].index))
print(drop_list)
```

```
len(drop_list)
```

```
58
```

```
dataset.drop(labels=drop_list, axis=1, inplace=True)
dataset.shape
```

```
(814986, 92)
```

Eliminando carac. basándose en la intuición y el criterio experto

```
print(sorted(dataset.columns))
```

```
['acc_now_delinq', 'acc_open_past_24mths', 'addr_state', 'annual_inc', 'application_type', 'avg
_cur_bal', 'bc_open_to_buy', 'bc_util', 'charged_off', 'chargeoff_within_12_mths', 'collection_
recovery_fee', 'collections_12_mths_ex_med', 'debt_settlement_flag', 'delinq_2yrs', 'delinq_amn
t', 'disbursement_method', 'dti', 'earliest_cr_line', 'emp_length', 'emp_title', 'fico_range_hi
gh', 'fico_range_low', 'funded_amnt', 'funded_amnt_inv', 'grade', 'hardship_flag', 'home_owners
hip', 'id', 'initial_list_status', 'inq_last_6mths', 'installment', 'int_rate', 'issue_d', 'las
t_credit_pull_d', 'last_fico_range_high', 'last_fico_range_low', 'last_pymnt_amnt', 'last_pymnt
_d', 'loan_amnt', 'mo_sin_old_il_acct', 'mo_sin_old_rev_tl_op', 'mo_sin_rcnt_rev_tl_op', 'mo_si
n_rcnt_tl', 'mort_acc', 'mths_since_recent_bc', 'mths_since_recent_inq', 'num_accts_ever_120_p
d', 'num_actv_bc_tl', 'num_actv_rev_tl', 'num_bc_sats', 'num_bc_tl', 'num_il_tl', 'num_op_rev_t
l', 'num_rev_accts', 'num_rev_tl_bal_gt_0', 'num_sats', 'num_tl_120dpd_2m', 'num_tl_30dpd', 'nu
m_tl_90g_dpd_24m', 'num_tl_op_past_12m', 'open_acc', 'out_prncp', 'out_prncp_inv', 'pct_tl_nvr_
dlq', 'percent_bc_gt_75', 'policy_code', 'pub_rec', 'pub_rec_bankruptcies', 'purpose', 'pymnt_p
lan', 'recoveries', 'revol_bal', 'revol_util', 'sub_grade', 'tax_liens', 'term', 'title', 'tot_
coll_amt', 'tot_cur_bal', 'tot_hi_cred_lim', 'total_acc', 'total_bal_ex_mort', 'total_bc_limi
t', 'total_il_high_credit_limit', 'total_pymnt', 'total_pymnt_inv', 'total_rec_int', 'total_rec
_late_fee', 'total_rec_prncp', 'total_rev_hi_lim', 'verification_status', 'zip_code']
```

```
keep_list = ['charged_off','funded_amnt','addr_state',

len(keep_list)
```

```
40
```

```
drop_list = [col for col in dataset.columns if col not in keep_list]

dataset.drop(labels=drop_list, axis=1, inplace=True)

dataset.shape
```

```
(814986, 39)
```

Eliminando carac. que tienen una correlación menor al 3%

```
correlation = dataset.corr()
correlation_chargeOff = abs(correlation['charged_off'])

correlation_chargeOff.sort_values(ascending=False)
```

charged_off	1.000000
last_pymnt_amnt	0.381359
int_rate	0.247815
fico_range_low	0.139430
fico_range_high	0.139428
dti	0.123031
acc_open_past_24mths	0.098985
bc_open_to_buy	0.086896
avg_cur_bal	0.085777
num_actv_rev_tl	0.077211
bc_util	0.077132
mort_acc	0.077086
revol_util	0.072185
funded_amnt	0.064258
loan_amnt	0.064139
mo_sin_rcnt_rev_tl_op	0.053469
mo_sin_old_rev_tl_op	0.048529
annual_inc	0.046685

```
drop_list_corr = sorted(list(correlation_chargeOff[correlation_chargeOff < 0.03].index))
print(drop_list_corr)
```

```
['pub_rec', 'pub_rec_bankruptcies', 'revol_bal', 'total_acc']
```

```
dataset.drop(labels=drop_list_corr, axis=1, inplace=True)
dataset.shape
```

```
(814986, 35)
```

Un vistazo a las variables categóricas

```
dataset[['id','emp_title','title','zip_code']].describe()
```

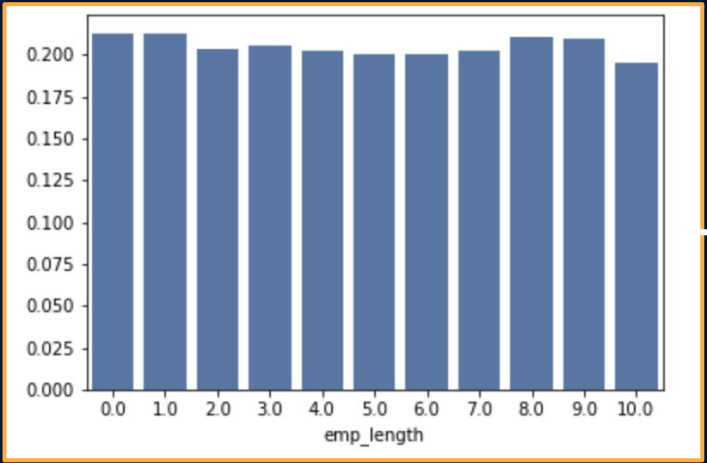
	id	emp_title	title	zip_code
count	814986	766415	807068	814986
unique	814986	280473	60298	925
top	14680062	Teacher	Debt consolidation	945xx
freq	1	11351	371874	9517

```
dataset.drop(['id','emp_title','title','zip_code'], axis=1, inplace=True)
```

¿Variables que aparentemente no tienen mucha relevancia?

```
dataset['term'] = dataset['term'].apply(lambda s: np.int8(s.split()[0]))

dataset.groupby('term')['charged_off'].value_counts(normalize=True).loc[:,1]
```

```
term
36    0.165710
60    0.333793
```



¿Variables que aparentemente sí tienen bastante relevancia?

```
dataset.drop(['emp_length'], axis=1, inplace=True)
```

```
dataset[['annual_inc']].describe()
```

	annual_inc
count	8.149860e+05
mean	7.523039e+04
std	6.524373e+04
min	0.000000e+00
25%	4.500000e+04
50%	6.500000e+04
75%	9.000000e+04
max	9.550000e+06

¿Variables que necesiten alguna transformación en particular?

```
dataset['log_annual_inc'] = dataset['annual_inc'].apply(lambda x: np.log10(x+1))
dataset.drop('annual_inc', axis=1, inplace=True)
```

¿Variables que vienen “en pareja”?

	fico_range_low	fico_range_high
fico_range_low	1.0	1.0
fico_range_high	1.0	1.0

```
dataset[['fico_range_low','fico_range_high']].corr()
```

```
dataset['fico_score'] = 0.5*dataset['fico_range_low'] + 0.5*dataset['fico_range_high']

dataset.drop(['fico_range_high', 'fico_range_low'], axis=1, inplace=True)
```

```
dataset['charged_off'].value_counts()
```

```
0    646902
1    168084
```

¿Alguna alternativa a los dummies?

```
from sklearn.preprocessing import LabelEncoder

# Categorical boolean mask
categorical_feature_mask = dataset.dtypes==object
# filter categorical columns using mask and turn it into a list
categorical_cols = dataset.columns[categorical_feature_mask].tolist()
```

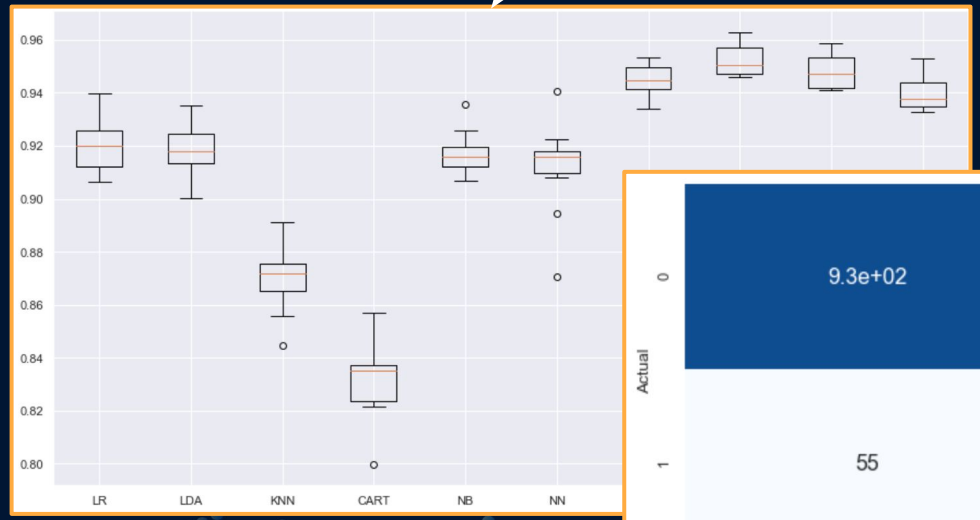
```
categorical_cols

['grade',
 'sub_grade',
 'home_ownership',
 'verification_status',
 'purpose',
 'addr_state',
 'initial_list_status',
 'application_type']
```

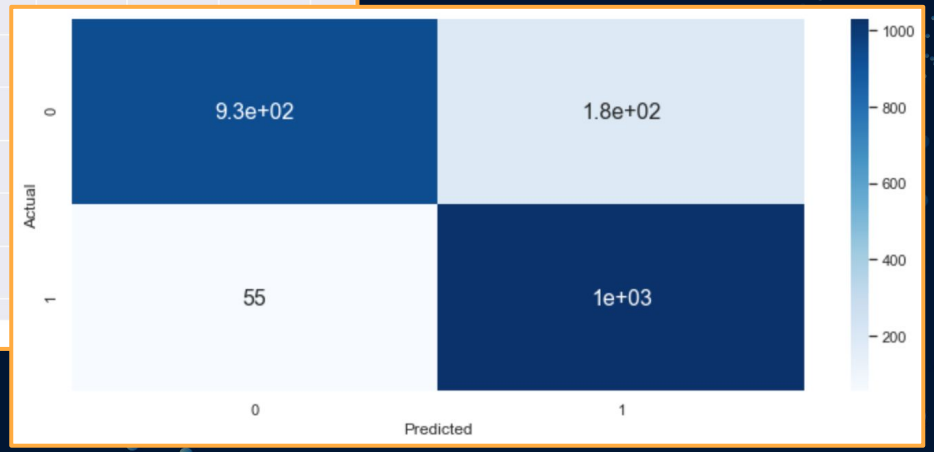
	grade	sub_grade	home_ownership	verification_status	purpose	addr_state	initial_list_status	application_type
0	2	10	5	1	2	45	1	0
1	0	2	1	0	1	4	1	0
2	3	15	5	1	1	24	1	0
4	2	12	5	1	2	3	0	0
5	2	12	5	1	2	31	0	0

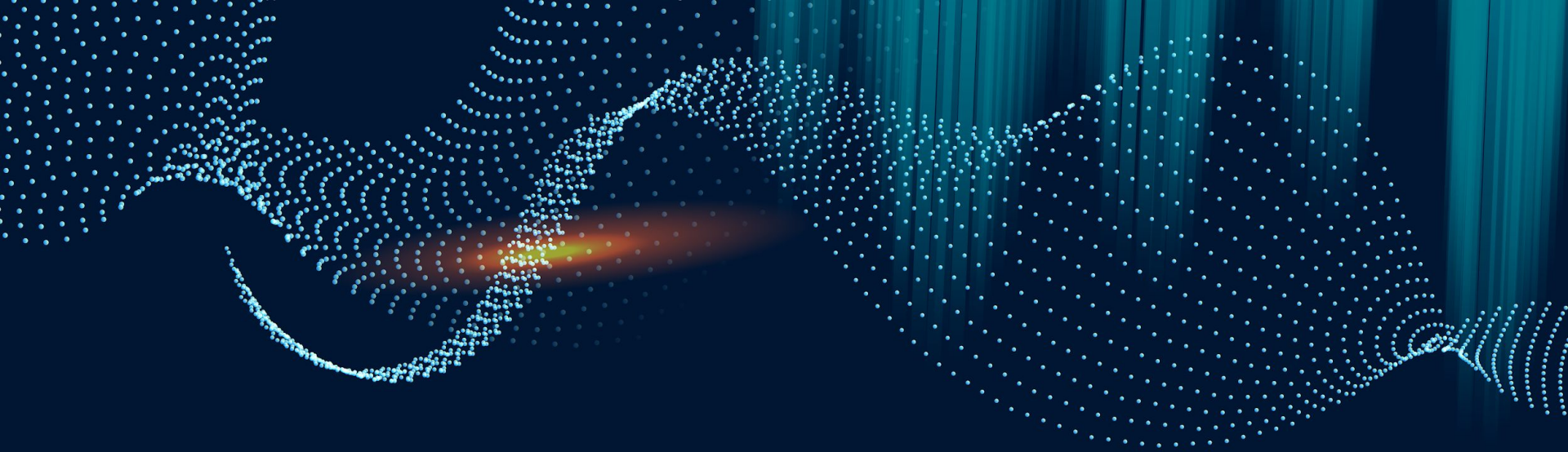
```
le = LabelEncoder()
# apply le on categorical feature columns
dataset[categorical_cols] = dataset[categorical_cols].apply(lambda col: le.fit_transform(col))
dataset[categorical_cols].head(10)
```

Pasos siguientes, lo que ya conocemos

```
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
#Neural Network
models.append(('NN', MLPClassifier()))
#Ensable Models
# Boosting methods
models.append(('AB', AdaBoostClassifier()))
models.append(('GBM', GradientBoostingClassifier()))
# Bagging methods
models.append(('RF', RandomForestClassifier()))
models.append(('ET', ExtraTreesClassifier()))
```







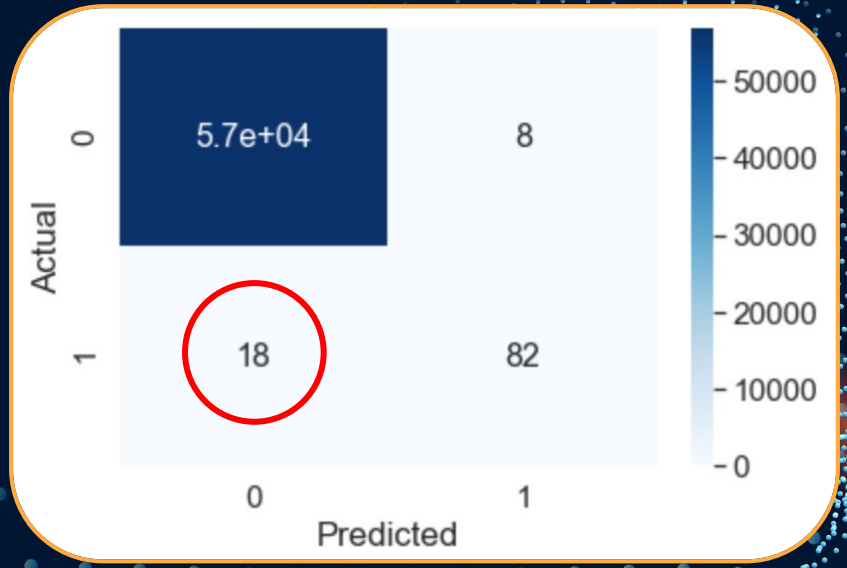
02 Datos desbalanceados

Detección de fraude

Transacciones:

- **No fraudulentas = 284,315**
- **Fraudulentas = 492**

```
0.9995435553526912
[[56854     8]
 [   18    82]]
              precision    recall

           0       1.00      1.00
           1       0.91      0.82

    accuracy
   macro avg       0.96      0.91
weighted avg       1.00      1.00
```



Aplicando undersampling

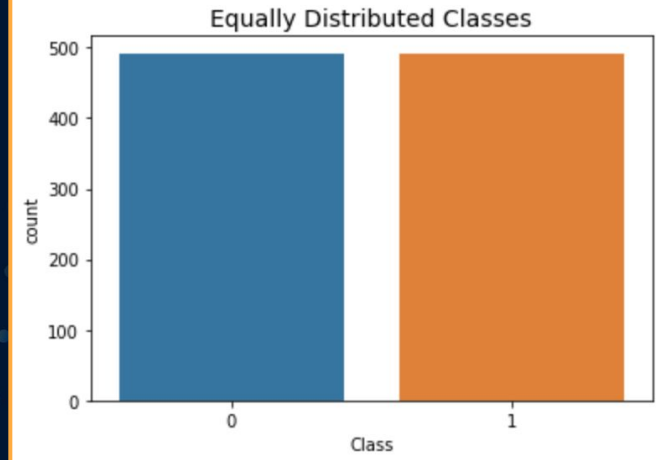
```
# amount of fraud classes 492 rows.
fraud_df = df.loc[df['Class'] == 1]
non_fraud_df = df.loc[df['Class'] == 0][:492]
```

```
normal_distributed_df = pd.concat([fraud_df, non_fraud_df])
```

```
# Shuffle dataframe rows
df_new = normal_distributed_df.sample(frac=1, random_state=42)
# split out validation dataset for the end
Y_train_new= df_new["Class"]
X_train_new = df_new.loc[:, dataset.columns != 'Class']
```

```
print('Distribution of the Classes in the subsample dataset')
print(df_new['Class'].value_counts()/len(df_new))
sns.countplot('Class', data=df_new)
pyplot.title('Equally Distributed Classes', fontsize=14)
pyplot.show()
```

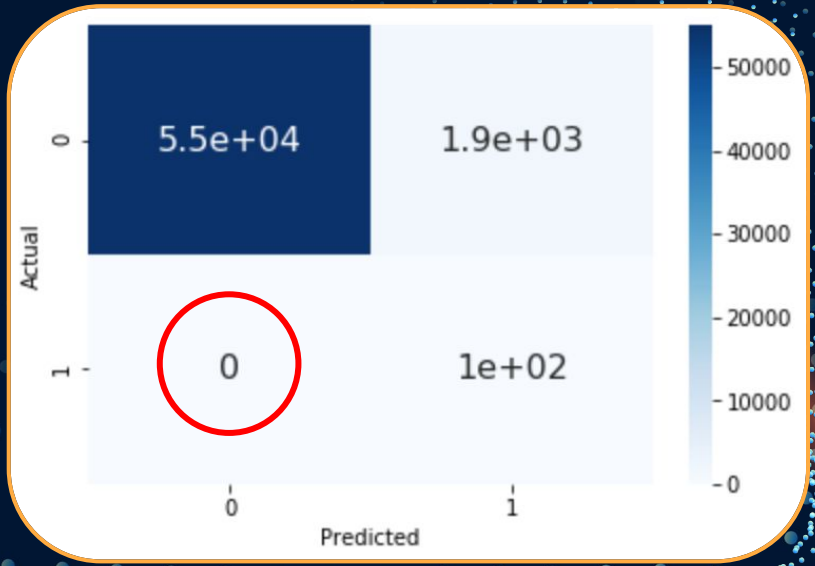
```
Distribution of the Classes in the subsample dataset
1    0.5
0    0.5
Name: Class, dtype: float64
```

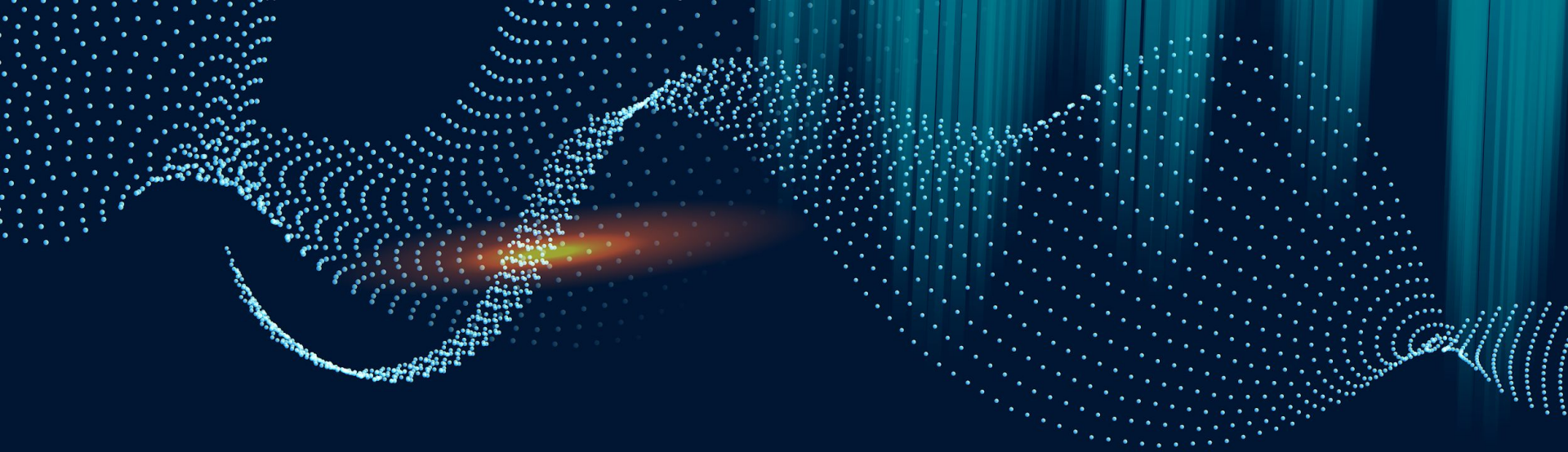


Transacciones:

- **No fraudulentas = 492**
- **Fraudulentas = 492**

```
0.9668199852533268
[[54972  1890]
 [    0   100]]
              precision    recall

           0       1.00      0.97
           1       0.05      1.00

    accuracy
   macro avg       0.53      0.98
weighted avg       1.00      0.97
```





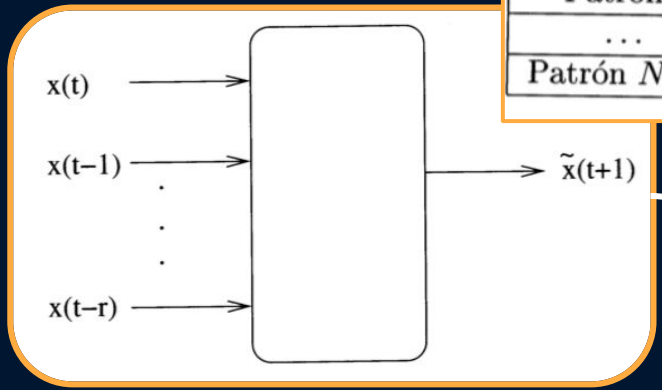
03 Series temporales

Enfoques para el Aprendizaje supervisado

Predicción de series temporales: un paso en el futuro

$$\tilde{x}(t+1) = \tilde{F}\big(x(t), x(t-1), ..., x(t-r)\big)$$

	Entrada	Salida deseada
Patrón 1	$x(r), x(r-1), ..., x(1), x(0)$	$x(r+1)$
Patrón 2	$x(r+1), x(r), ..., x(2), x(1)$	$x(r+2)$
Patrón 3	$x(r+2), x(r+1), ..., x(3), x(2)$	$x(r+3)$
Patrón 4	$x(r+3), x(r+2), ..., x(4), x(3)$	$x(r+4)$
...	...	...
Patrón $N-r$	$x(N-1), x(N-2), ..., x(N-r), x(N-(r+1))$	$x(N)$

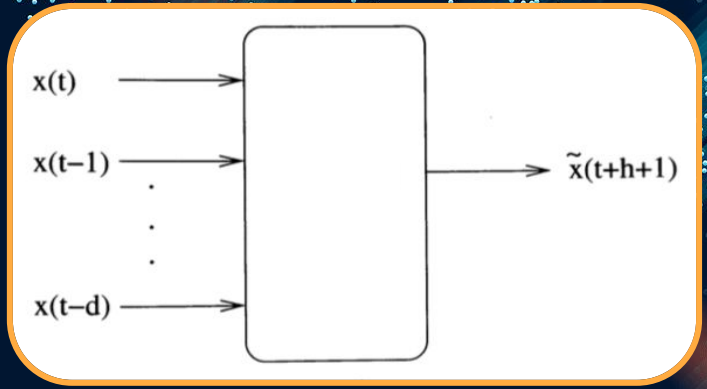


Time step	Value
1	10
2	11
3	18
4	15
5	20

X	Y
?	10
10	11
11	18
18	15
15	20
20	?

Con r = 1

Predicción de series temporales: múltiples pasos en el futuro



$$\tilde{x}(t+h+1) = \tilde{G}\big(x(t), x(t-1), ..., x(t-d)\big)$$

	Entrada	Salida deseada
Patrón 1	$x(d), x(d-1), ..., x(1), x(0)$	$x(d+h+1)$
Patrón 2	$x(d+1), x(d), ..., x(2), x(1)$	$x(d+h+2)$
Patrón 3	$x(d+2), x(d+1), ..., x(3), x(2)$	$x(d+h+3)$
Patrón 4	$x(d+3), x(d+2), ..., x(4), x(3)$	$x(d+h+4)$
...	...	...
Patrón N-d-h	$x(N-h-1), x(N-h-2), ..., x(N-1-h-d)$	$x(N)$

Ejemplo: predecir el precio de las acciones de Microsoft

Y	• Valor actual acciones de Microsoft
X	• Valor acciones **GOOGL** hace 5 días. • Valor acciones **IBM** hace 5 días. • Proporción **USD/JPY** hace 5 días. • Proporción **GBP/USD** hace 5 días. • Índice **S&P 500** hace 5 días. • Índice **Dow Jones** hace 5 días. • Valor acciones **Microsoft** hace 5 días. • Valor acciones **Microsoft** hace 15 días. • Valor acciones **Microsoft** hace 30 días. • Valor acciones **Microsoft** hace 60 días.

FRED
ECONOMIC DATA | ST. LOUIS FED

```
# Load libraries
import numpy as np
import pandas as pd
import pandas_datareader.data as web
```

```
stk_tickers = ['MSFT', 'IBM', 'GOOGL']
ccy_tickers = ['DEXJPUS', 'DEXUSUK']
idx_tickers = ['SP500', 'DJIA', 'VIXCLS']

stk_data = web.DataReader(stk_tickers, 'yahoo')
ccy_data = web.DataReader(ccy_tickers, 'fred')
idx_data = web.DataReader(idx_tickers, 'fred')
```

```
Y = np.log(stk_data.loc[:, ('Adj Close', 'MSFT')])

X1 = np.log(stk_data.loc[:, ('Adj Close', ('GOOGL', 'IBM'))]).shift(-return_period)
X2 = np.log(ccy_data).shift(-return_period)
X3 = np.log(idx_data).shift(-return_period)

X4 = pd.concat([np.log(stk_data.loc[:, ('Adj Close', 'MSFT')]).shift(-i)
                for i in [return_period, return_period*3, return_period*6, return_period*12]],
                axis=1).dropna()
X4.columns = ['MSFT_DT', 'MSFT_3DT', 'MSFT_6DT', 'MSFT_12DT']

X = pd.concat([X1, X2, X3, X4], axis=1)

dataset = pd.concat([Y, X], axis=1).dropna().iloc[:, :]
```

```
return_period = 5
```

```
>>> df
            Col1  Col2  Col3
2020-01-01    10    13    17
2020-01-02    20    23    27
2020-01-03    15    18    22
2020-01-04    30    33    37
2020-01-05    45    48    52
```

```
>>> df.shift(periods=3)
            Col1  Col2  Col3
2020-01-01   NaN   NaN   NaN
2020-01-02   NaN   NaN   NaN
2020-01-03   NaN   NaN   NaN
2020-01-04  10.0  13.0  17.0
2020-01-05  20.0  23.0  27.0
```

Preparando el conjunto de datos

Comparando varios modelos

Regression and Tree Regression algorithms

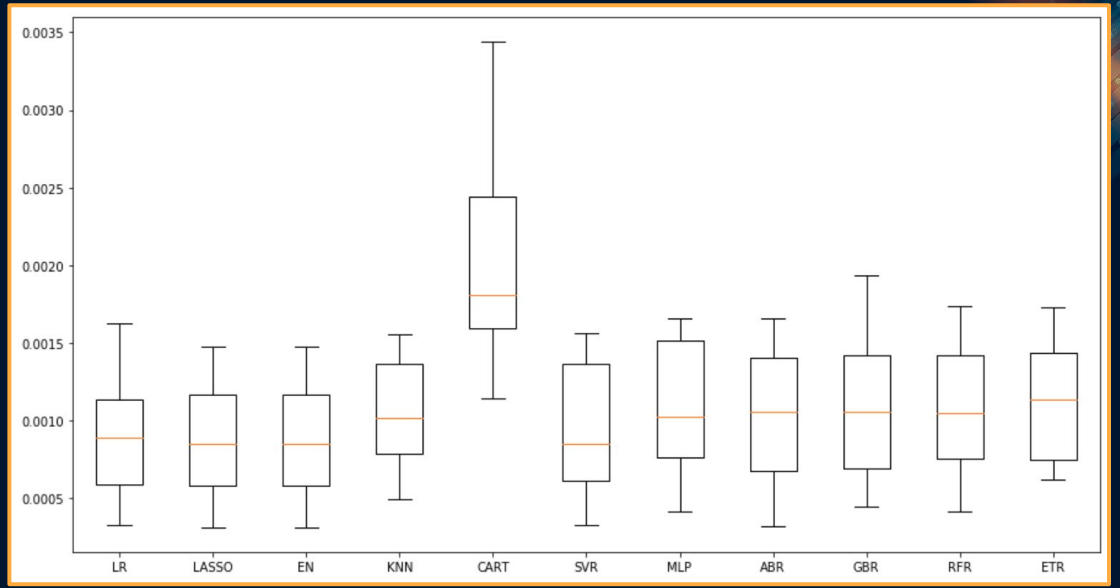
```
models = []
models.append(('LR', LinearRegression()))
models.append(('LASSO', Lasso()))
models.append(('EN', ElasticNet()))
models.append(('KNN', KNeighborsRegressor()))
models.append(('CART', DecisionTreeRegressor()))
models.append(('SVR', SVR()))
```

Neural Network algorithms

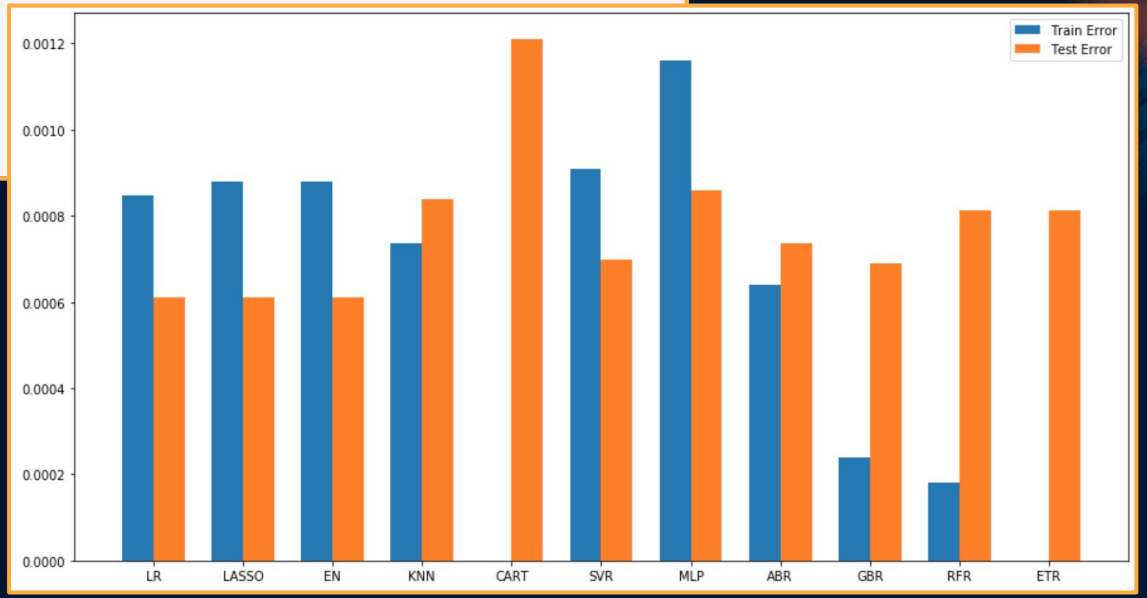
```
models.append(('MLP', MLPRegressor()))
```

Ensamble Models

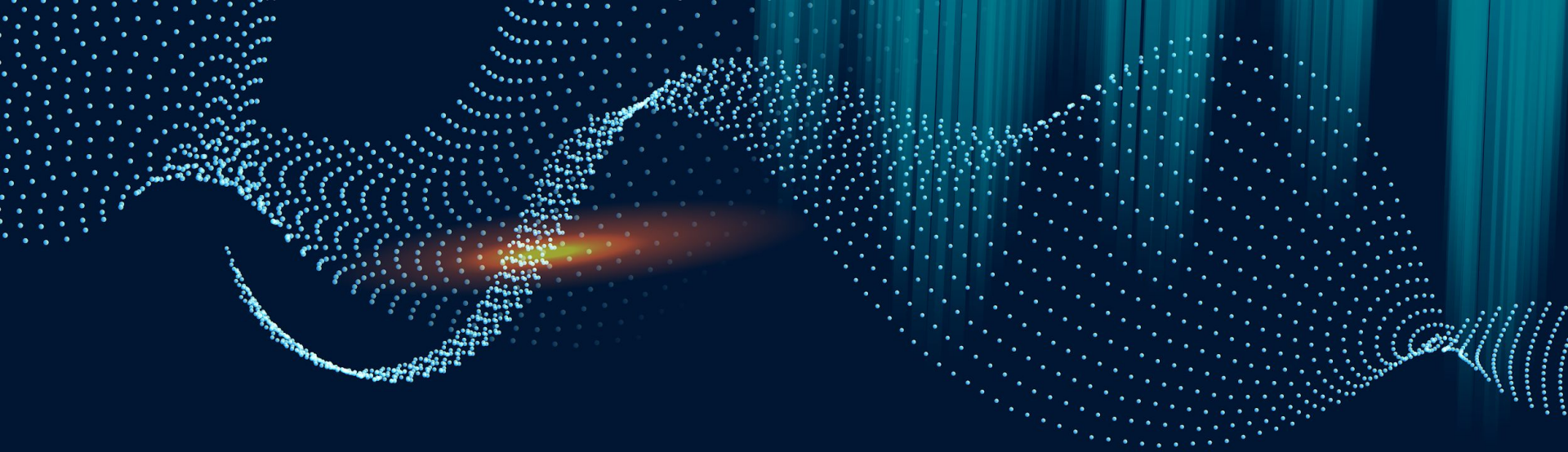
```
# Boosting methods
models.append(('ABR', AdaBoostRegressor()))
models.append(('GBR', GradientBoostingRegressor()))
# Bagging methods
models.append(('RFR', RandomForestRegressor()))
models.append(('ETR', ExtraTreesRegressor()))
```



```
fig = pyplot.figure()

ind = np.arange(len(names))  # the x locations for the groups
width = 0.35  # the width of the bars

fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.bar(ind - width/2, train_results,  width=width, label='Train Error')
pyplot.bar(ind + width/2, test_results, width=width, label='Test Error')
fig.set_size_inches(15,8)
pyplot.legend()
ax.set_xticks(ind)
ax.set_xticklabels(names)
pyplot.show()
```



Verificando si ha habido sobreajuste

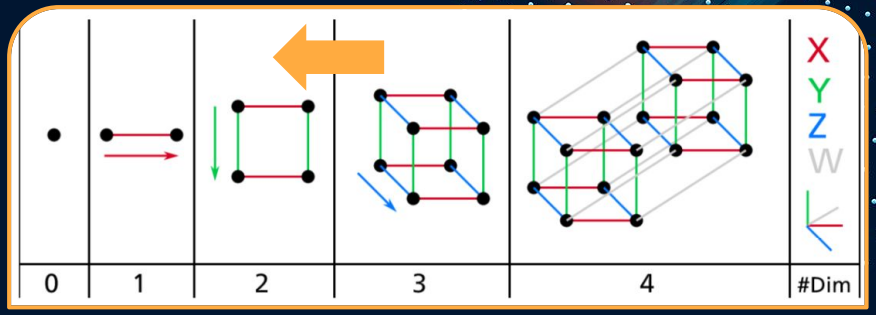


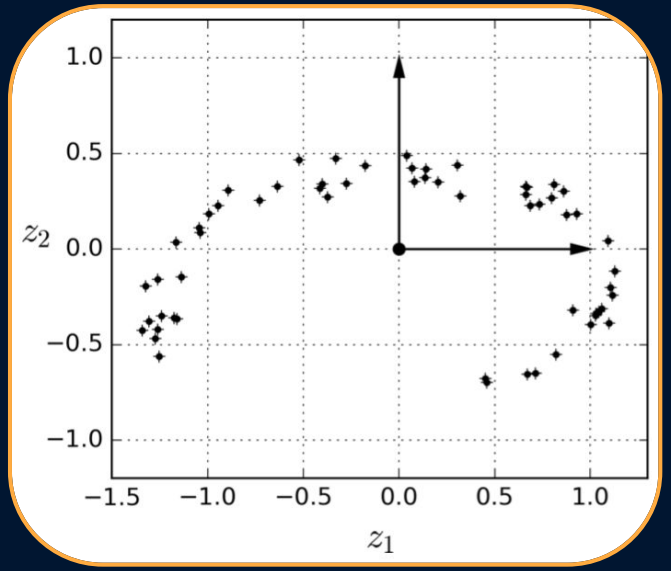
04

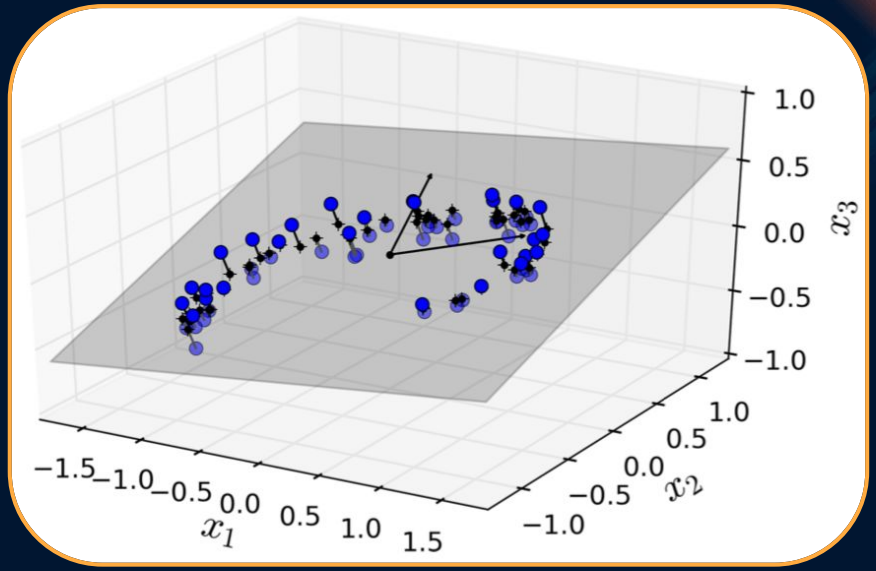
Reducción de la dimensionalidad

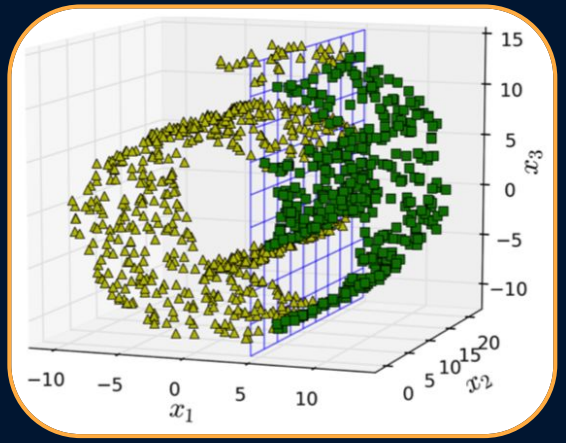
Introducción al uso de PCA en ML

¿A qué nos referimos?



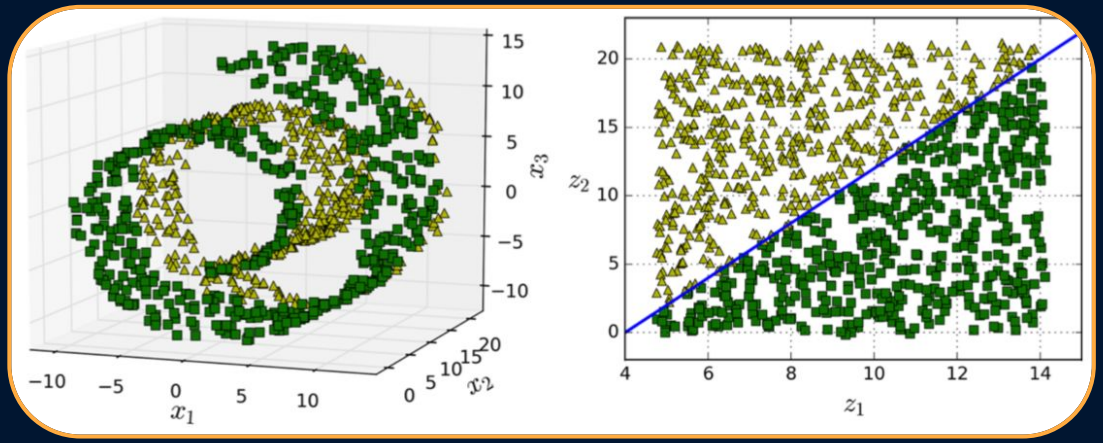






A veces no será necesario

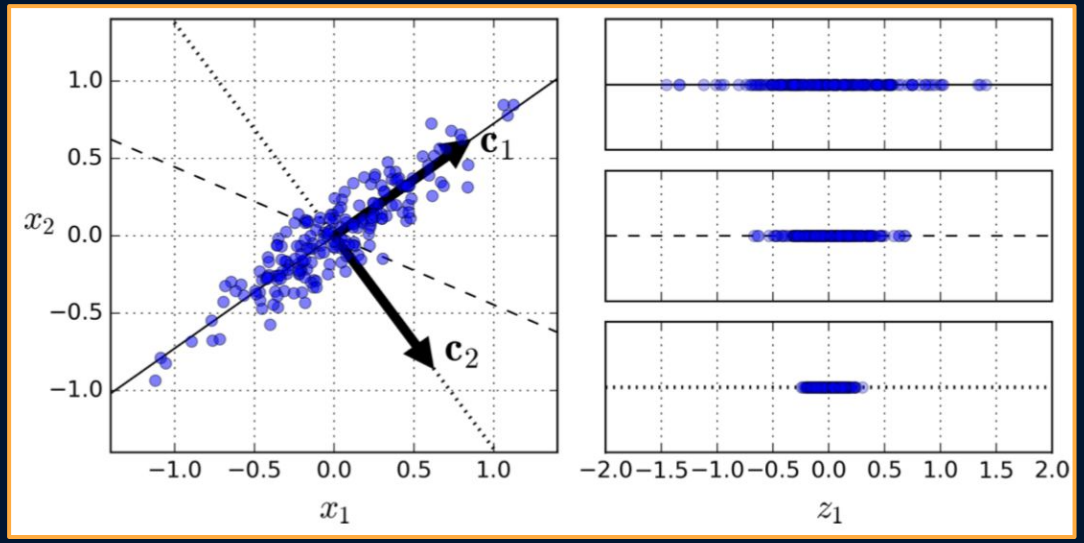
¿Por qué hacerlo en ML?

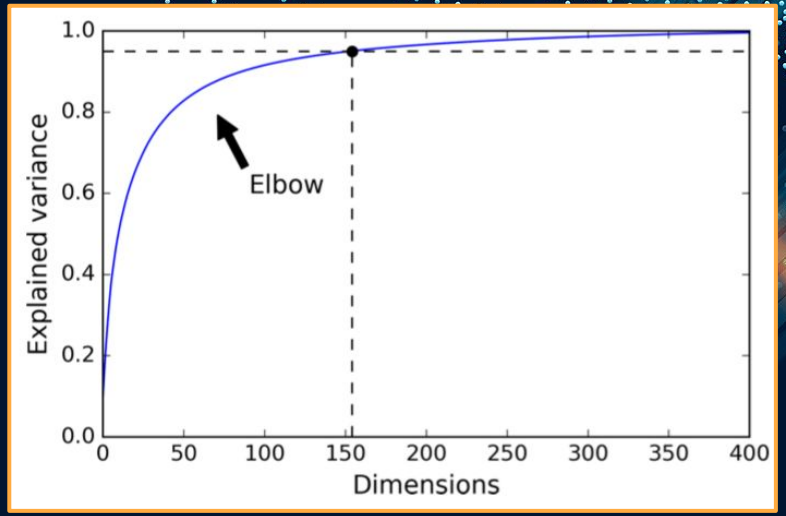


A veces nos ayudará para encontrar un modelo más preciso (o más sencillo).

No basta con reducir dimensiones

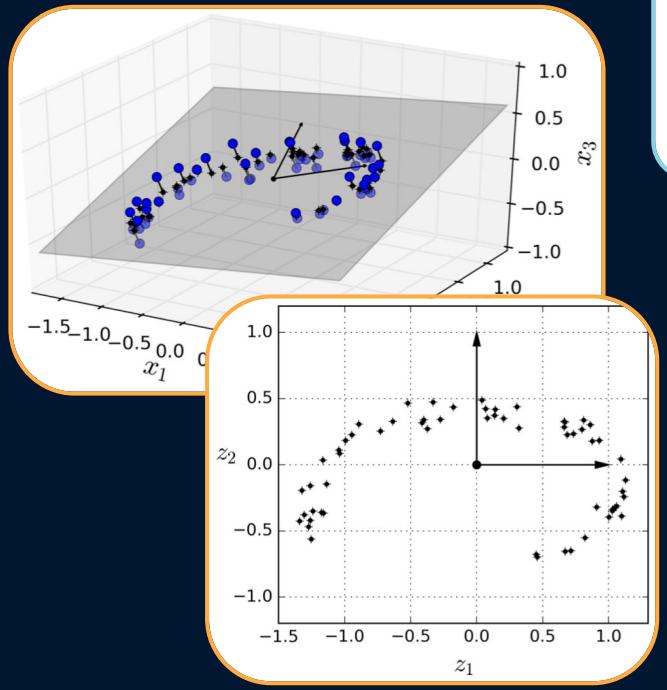
El algoritmo se encarga de encontrar la dimensión que retenga la mayor varianza entre los datos. Siempre se perderá información, pero se desea perder la menor cantidad posible.





Se desea seleccionar la menor cantidad de dimensiones, sin perder mucha información.

Hiperparámetro n_components




```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit_transform(X)
```

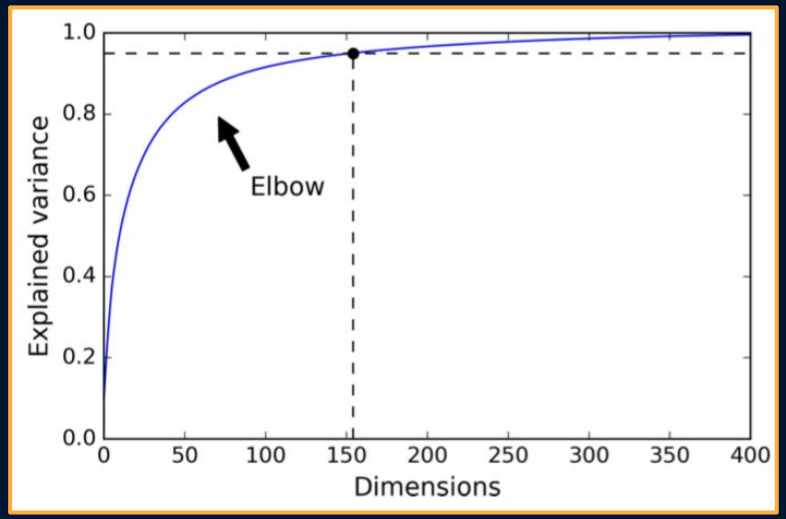
```
>>> print(pca.explained_variance_ratio_)
array([ 0.84248607,  0.14631839])
```

Para fijar una cantidad de dimensiones, hay que proveer un número natural mayor a uno

El 1.2% restante es el “precio a pagar” por buscar un modelo más sencillo de entrenar

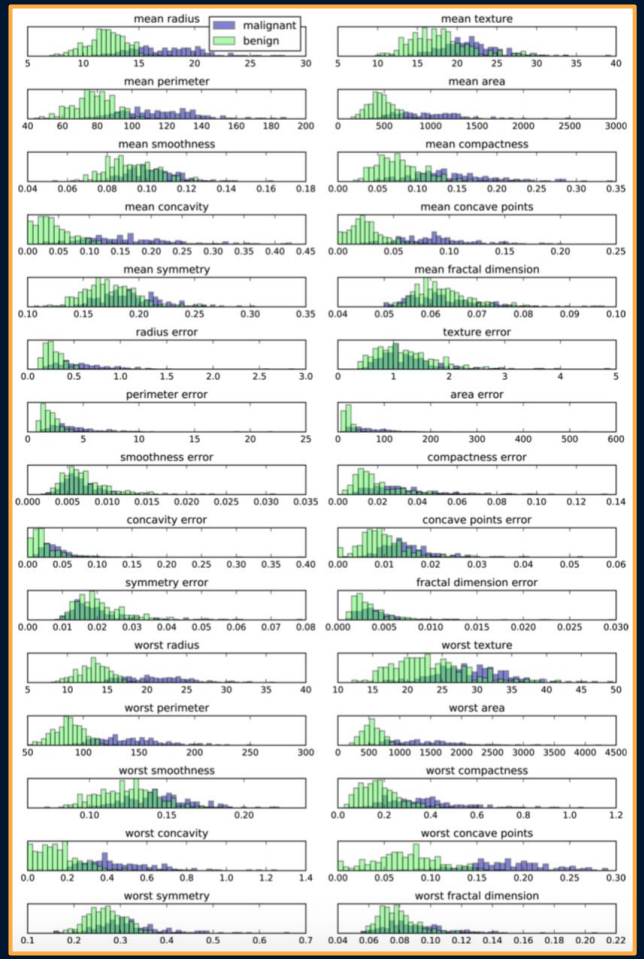
Hiperparámetro n_components

En cambio, si lo que nos interesa es preservar cierto porcentaje de varianza (lo más utilizado)




```
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X)
```

Entonces se debe proveer un número real en el rango entre cero y uno



30 variables de entrada

```
from sklearn.datasets import load_breast_cancer
cancer = load_breast_cancer()

scaler = StandardScaler()
scaler.fit(cancer.data)
X_scaled = scaler.transform(cancer.data)
```

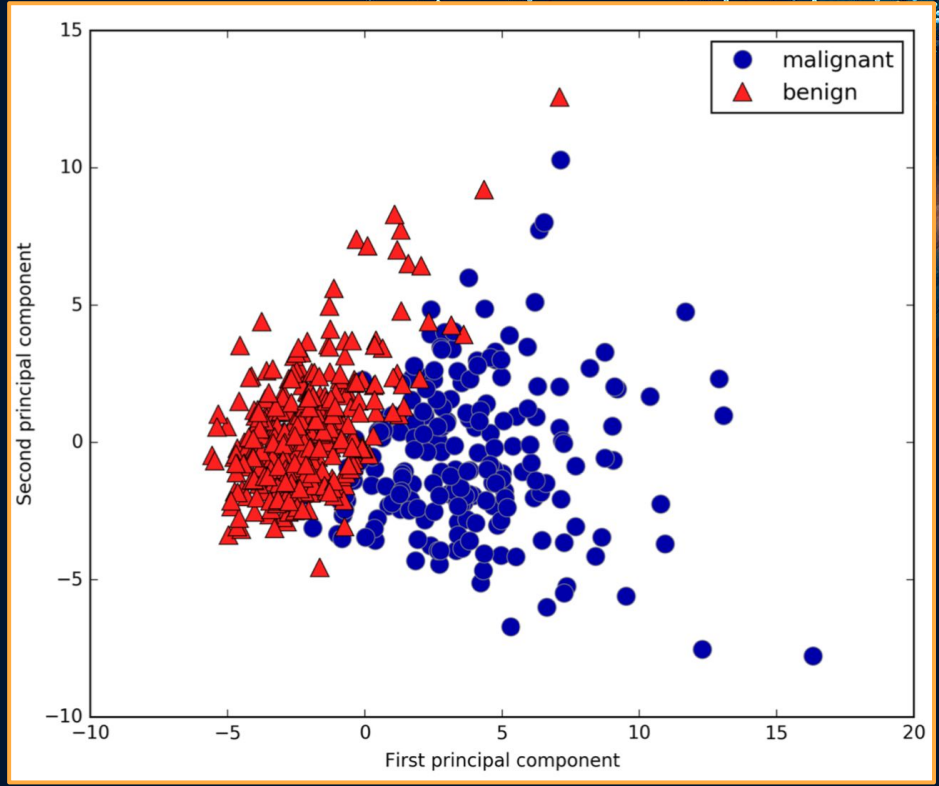
Ejemplo de aplicación: detección de cáncer de mama

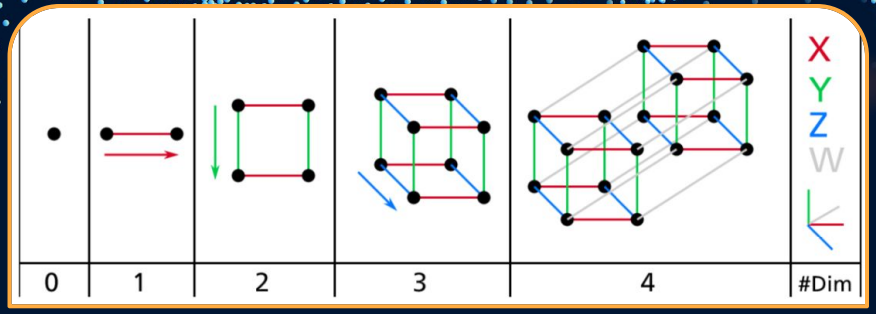
¡Con solo 2 componentes, tenemos un muy buen conjunto de datos!

```
from sklearn.decomposition import PCA
# keep the first two principal components of the data
pca = PCA(n_components=2)
# fit PCA model to breast cancer data
pca.fit(X_scaled)

# transform data onto the first two principal components
X_pca = pca.transform(X_scaled)
print("Original shape: {}".format(str(X_scaled.shape)))
print("Reduced shape: {}".format(str(X_pca.shape)))
```

```
Original shape: (569, 30)
Reduced shape: (569, 2)
```

```
# plot first vs. second principal component, colored by class
plt.figure(figsize=(8, 8))
mglearn.discrete_scatter(X_pca[:, 0], X_pca[:, 1], cancer.target)
plt.legend(cancer.target_names, loc="best")
plt.gca().set_aspect("equal")
plt.xlabel("First principal component")
plt.ylabel("Second principal component")
```





05

Aplicando reducción de la dimensionalidad

Características de flores **[Práctica]**

¿Consultas,

dudas o comentarios?

Muchas gracias por su asistencia y atención

UNIVERSIDAD DE GRANADA

Universidad Centroamericana José Simeón Cañas