

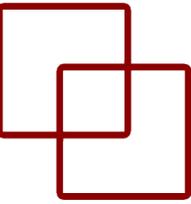
# Tema 6. Fase de optimización y forecasting



**Dr. Manuel Castillo-Cara**  
[www.manuelcastillo.eu](http://www.manuelcastillo.eu)

**Departamento de Inteligencia Artificial**  
**Escuela Técnica Superior de Ingeniería Informática**  
**Universidad Nacional de Educación a Distancia (UNED)**

# Índice

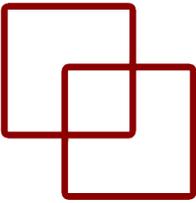


- Background
- Pipelines.
- Preprocesamiento Avanzado.
- Fase de Optimización.
- Fase de Forecasting.



# Background

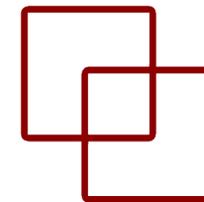
# 1. Background



	<u>TYPE</u>	<u>NAME</u>	<u>DESCRIPTION</u>	<u>ADVANTAGES</u>	<u>DISADVANTAGES</u>
Linear		Linear regression	The “best fit” line through all data points. Predictions are numerical.	Easy to understand – you clearly see what the biggest drivers of the model are.	<ul style="list-style-type: none"> <li>X Sometimes too simple to capture complex relationships between variables.</li> <li>X Tendency for the model to “overfit”.</li> </ul>
		Logistic regression	The adaptation of linear regression to problems of classification (e.g., yes/no questions, groups, etc.)	Also easy to understand.	<ul style="list-style-type: none"> <li>X Sometimes too simple to capture complex relationships between variables.</li> <li>X Tendency for the model to “overfit”.</li> </ul>
Tree-based		Decision tree	A graph that uses a branching method to match all possible outcomes of a decision.	Easy to understand and implement.	<ul style="list-style-type: none"> <li>X Not often used on its own for prediction because it's also often too simple and not powerful enough for complex data.</li> </ul>
		Random Forest	Takes the average of many decision trees, each of which is made with a sample of the data. Each tree is weaker than a full decision tree, but by combining them we get better overall performance.	A sort of “wisdom of the crowd”. Tends to result in very high quality models. Fast to train.	<ul style="list-style-type: none"> <li>X Can be slow to output predictions relative to other algorithms.</li> <li>X Not easy to understand predictions.</li> </ul>
		Gradient Boosting	Uses even weaker decision trees, that are increasingly focused on “hard” examples.	High-performing.	<ul style="list-style-type: none"> <li>X A small change in the feature set or training set can create radical changes in the model.</li> <li>X Not easy to understand predictions.</li> </ul>
Neural networks		Neural networks	Mimics the behavior of the brain. Neural networks are interconnected neurons that pass messages to each other. Deep learning uses several layers of neural networks put one after the other.	Can handle extremely complex tasks - no other algorithm comes close in image recognition.	<ul style="list-style-type: none"> <li>X Very, very slow to train, because they have so many layers. Require a lot of power.</li> <li>X Almost impossible to understand predictions.</li> </ul>



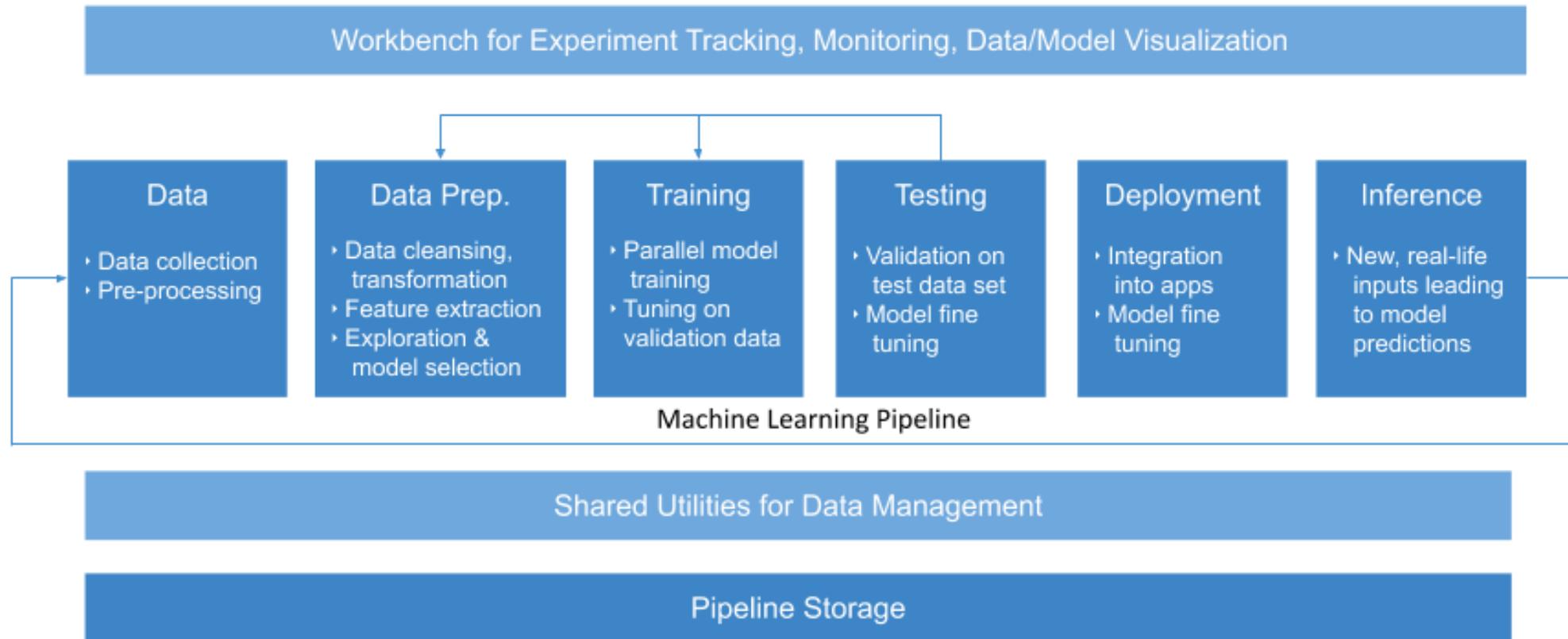
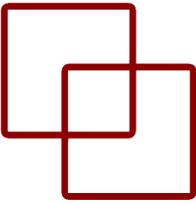
# Pipelines

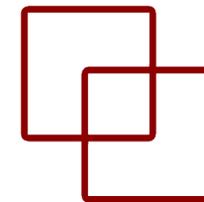


# 1. Definición (I)

- Un Pipeline consiste en una combinación de estimadores que se ejecutan como si fuesen uno.
  - Más concretamente se trata de una secuencia de transformaciones y el último es un estimador de cualquier tipo (transformador, clasificador, etc.).
- Pipeline se encarga de ir llamando a las funciones *fit* y *transform* de cada uno de los transformadores hasta que llega al último de ellos, siendo la entrada de la función *fit* el resultado del *transform* anterior.
- Pipeline tendrá aquellas funciones correspondientes a su último estimador, es decir, si al final hay un clasificador, el Pipeline tendrá las funciones *fit*, *predict* y *score*, si es un transformador, *fit* y *transform*.
- Se trata de una herramienta muy útil que permite reducir el tamaño del código y ayuda a la reproducibilidad de diferentes experimentos.

# 1. Definición (II)





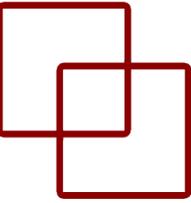
## 2. Pipeline

- Observe cómo creamos una lista de pasos de Python que se proporcionan a Pipeline para procesar los datos.
- Observe también cómo el Pipeline en sí mismo se trata como un estimador y se evalúa en su totalidad mediante el procedimiento de validación cruzada *k-fold*.
- Ejecutar el ejemplo proporciona un resumen del Accuracy de la configuración en el conjunto de datos.

```
# Create a pipeline that standardizes the data then creates a model
# create pipeline
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('lda', LinearDiscriminantAnalysis()))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)")
```

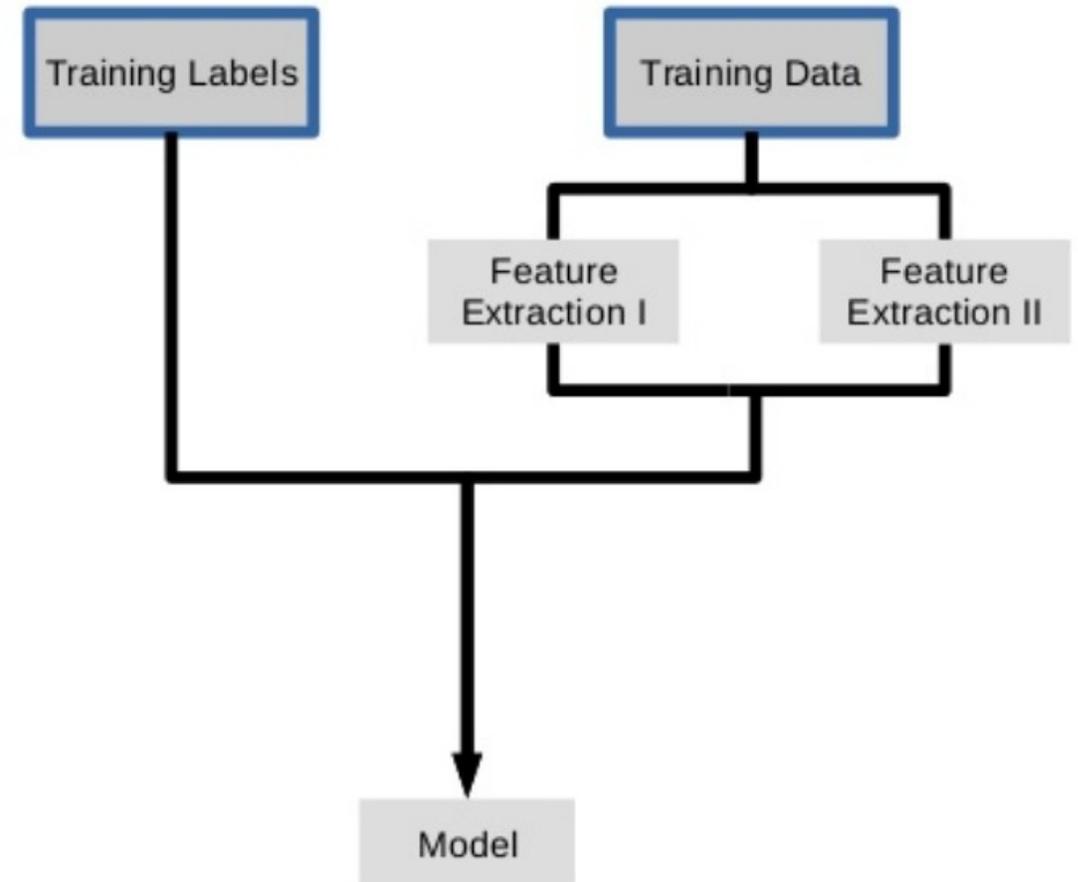
Accuracy: 77.35% (5.16%)

# 3. FeatureUnion

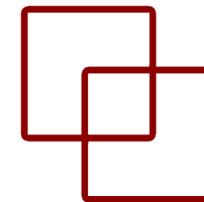


- *FeatureUnion* permite combinar los resultados de múltiples procedimientos de selección y extracción de características.
  - Toda la extracción de características y la unión de características se produce dentro de cada *fold* del procedimiento de validación cruzada.

## Feature Union



# 3. FeatureUnion



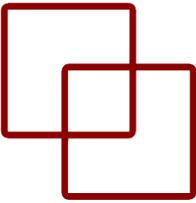
- *FeatureUnion* permite combinar los resultados de múltiples procedimientos de selección y extracción de características.
  - toda la extracción de características y la unión de características se produce dentro de cada *fold* del procedimiento de validación cruzada.
- El ejemplo muestra el Pipeline en cuatro pasos:
  - Extracción de características con PCA (3 características).
  - Extracción de características con selección estadística (6 características).
  - Unión de características.
  - Evaluación con un modelo LoR.
- Observe cómo *FeatureUnion* en resumidas cuentas "es un Pipeline dentro de otro".

```
features = []
features.append(('pca', PCA(n_components=3)))
features.append(('select_best', SelectKBest(k=6)))
feature_union = FeatureUnion(features)
# create pipeline
estimators = []
estimators.append(('feature_union', feature_union))
estimators.append(('logistic', LogisticRegression(
    solver='lbfgs', max_iter=1000)))
model = Pipeline(estimators)
# evaluate pipeline
kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}%
      ({results.std()*100.0:,.2f}%)|")
```

Accuracy: 77.60% (5.16%)



# Preprocesamiento Avanzado

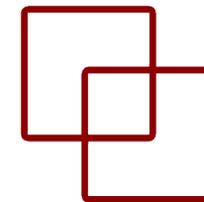


# 1. Valores NaN

- **Importante:** muchos algoritmos no son capaces de manejarlos y nos dá error en la ejecución.
- Una de las formas de realizar esta sustitución de valores perdidos consiste en utilizar la media (para valores continuos) o la moda (caso discreto).
- La función *SimpleImputer* se encarga de calcular y modificar los datos de entrada.

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(strategy='median')
imp.fit(wisconsin_data)
wisconsin_trans = imp.transform(wisconsin_data)
wisconsin_trans = pd.DataFrame(wisconsin_trans,
                               columns = wisconsin_data.columns)
print(np.sum(np.isnan(wisconsin_trans)))
```

```
patientId          0
clumpThickness     0
cellSize           0
CellShape          0
marginalAdhesion  0
epithelialSize     0
bareNuclei         0
blandChromatin     0
normalNucleoli     0
mitoses            0
dtype: int64
```

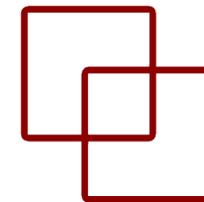


## 2. Escalar el atributo clase

- En problemas de regresión también puede ser crítico escalar y realizar otras transformaciones de datos en la variable objetivo.
- Esto se puede lograr en Python usando la clase *TransformedTargetRegressor*.
- Si tenemos un resultado Mean MAE inicial de 3.191 y le aplicamos una transformación Yeo-Johnson al target podemos ver como mejora considerablemente.

```
from sklearn.preprocessing import PowerTransformer
# prepare the model with input scaling
pipeline = Pipeline(steps=[('power', PowerTransformer()), ('model', HuberRegressor())])
# prepare the model with target scaling
model = TransformedTargetRegressor(regressor=pipeline, transformer=PowerTransformer())
# evaluate model
cv = KFold(n_splits=10, shuffle=True, random_state=1)
scores = cross_val_score(model, X, Y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
# convert scores to positive
scores = np.absolute(scores)
# summarize the result
s_mean = np.mean(scores)
print('Mean MAE: %.3f' % (s_mean))
```

Mean MAE: 2.926



# 3. One-Hot Encoding

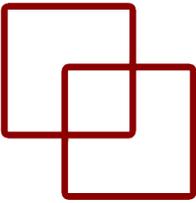
- Puede ser un desafío cuando tiene un conjunto de datos con tipos mixtos y desea aplicar transformaciones de datos selectivamente a algunas, pero no a todas, las características de entrada.
- Scikit-learn proporciona el *ColumnTransformer* que le permite aplicar transformaciones de datos de forma selectiva a diferentes columnas de su conjunto de datos.

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

# 3. One-Hot Encoding - código

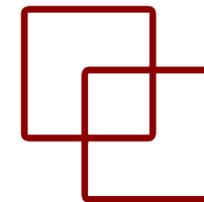


```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVR
# load dataset
filename = 'data/abalone.csv'
dataframe = pd.read_csv(filename, header=None)
array = dataframe.values
# split into inputs and outputs
last_ix = len(dataframe.columns) - 1
X, y = dataframe.drop(last_ix, axis=1), dataframe[last_ix]
# determine categorical and numerical features
numerical_ix = X.select_dtypes(include=['int64', 'float64']).columns
categorical_ix = X.select_dtypes(include=['object', 'bool']).columns
# define the data preparation for the columns
t = [('cat', OneHotEncoder(), categorical_ix), ('num', MinMaxScaler(), numerical_ix)]
col_transform = ColumnTransformer(transformers=t)
model = SVR(kernel='rbf', gamma='scale', C=100)
# define the data preparation and modeling pipeline
pipeline = Pipeline(steps=[('prep', col_transform), ('m', model)])
# define the model cross-validation configuration
cv = KFold(n_splits=10, shuffle=True, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
scores = np.absolute(scores)
print('MAE: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))
```

MAE: 1.465 (0.047)

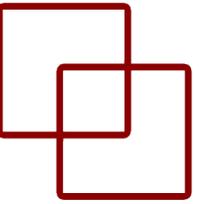


**Fase de optimización**



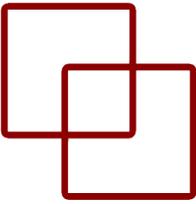
# 1. Introducción

- Mejorar la métrica de los algoritmos mediante la configuración de sus hiperparámetros.
- Es importante en esta fase tener 3 ó 4 algoritmos candidatos.
- FAQs
  - ¿Qué hiperparámetros afinar?
    - Cada algoritmo tiene sus propios hiperparámetros.
  - ¿Qué método de búsqueda utilizar para localizar buenos parámetros de algoritmo?
    - Tenemos dos técnicas principales, grid search y random search
  - ¿Qué opciones de prueba usar para limitar el ajuste excesivo de los datos de entrenamiento?
    - Tenemos que cuidarnos en no caer en overfitting.



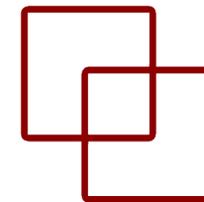
## 2. Procedimiento

- Utilizar el paquete scikit-learn de Python.
- Utilizar herramientas propias que vienen con el algoritmo.
- Diseñar una propia búsqueda de hiperparámetros.
- Usaremos el conjunto de datos de Pima Indians Diabetes.



# 3. Probar el algoritmo

- Probar hiperparámetros de Random Forest.
  - ‘*mtry*’: Número de variables muestreadas aleatoriamente como candidatos en cada división. Valores predeterminados
    - Clasificación: ‘*sqrt(p)*’
    - Regresión:  $p / 3$ .
    - Donde ‘*p*’ es el número de atributos en ‘*x*’ (60 para el conjunto de datos Sonar)
  - ‘*ntree*’: Número de árboles para crecer.
    - No debe establecerse en un número pequeño, para garantizar que cada instancia se predice al menos unas cuantas veces.

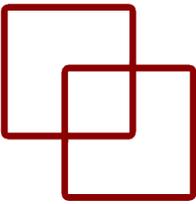


## 4. Modelo de línea base

- Antes de buscar los resultados de los mejores hiperparámetros debemos de conocer el resultado que nos dá el modelo que estemos utilizando como línea base.
- La idea de buscar hiperparámetros es mejorar el resultado predictivo que nos dé el modelo.
- En este caso estamos utilizando un algoritmo RiR que no tiene demasiados hiperparámetros por lo que no va a mejorar mucho.
- Sin embargo algoritmos de taxonomía no lineal mejoran muchísimo conforme configuramos sus hiperparámetros.

```
# RiR Classification
num_folds = 10
kfold = KFold(n_splits=5, random_state=7)
model = Ridge()
results = cross_val_score(model, X, Y, cv=kfold)
print(f"Accuracy: {results.mean()*100.0:,.2f}% ({results.std()*100.0:,.2f}%)"
```

Accuracy: 27.61% (1.61%)



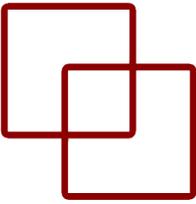
# 5. Grid Search

- Es un enfoque para el ajuste de parámetros que construirá y evaluará metódicamente un modelo para cada combinación de parámetros de algoritmo especificados en una cuadrícula (grid),
- Utiliza la clase *GridSearchCV*.
- El siguiente ejemplo evalúa diferentes valores alpha, siendo el óptimo el 1

```
# Grid Search for Algorithm Tuning
from sklearn.model_selection import GridSearchCV

alphas = np.array([1,0.1,0.01,0.001,0.0001,0])
param_grid = dict(alpha=alphas)
model = Ridge()
grid = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid.fit(X, Y)
print(f"Accuracy óptimo: {grid.best_score_.mean()*100.0:,.2f}%")
print(f"Valor de alpha óptimo: {grid.best_estimator_.alpha}")
```

```
Accuracy óptimo: 27.61%
Valor de alpha óptimo: 1.0
```



# 6. Random Search

- Realizar una búsqueda aleatoria de parámetros
- Utiliza la clase *RandomizedSearchCV*.
- El siguiente ejemplo evalúa diferentes valores alpha aleatorios entre 0 y 1, siendo el óptimo 0.97.

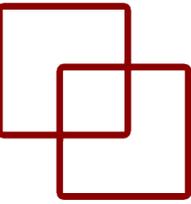
```
# Random Search for Algorithm Tuning
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV

param_grid = {'alpha': uniform()}
model = Ridge()
rsearch = RandomizedSearchCV(estimator=model, param_distributions=param_grid, n_iter=100, random_state=7, cv=5)
rsearch.fit(X, Y)
print(f"Accuracy óptimo: {rsearch.best_score_.mean()*100.0:,.2f}%")
print(f"Valor de alpha óptimo: {rsearch.best_estimator_.alpha}")
```

```
Accuracy óptimo: 27.61%
Valor de alpha óptimo: 0.9779895119966027
```

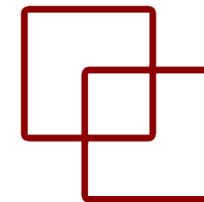


# Fase Forescasting



# 1. Introducción

- Conceptos:
  - Cómo utilizar su modelo entrenado para hacer predicciones sobre datos no etiquetados.
  - Cómo recrear un modelo de buen desempeño desde un entorno Python como un modelo independiente.
  - Cómo guardar su modelo en un archivo, cargarlo más tarde y hacer predicciones sobre datos no etiquetados.



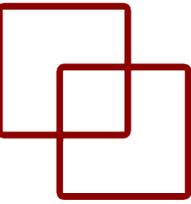
## 2. Pickle

- Pickle guarda el modelo en *model.sav* finalizado en su directorio de trabajo local.

```
# Save Model Using Pickle
import pickle as pkl
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
# Fit the model on 33%
model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
pkl.dump(model, open(filename, 'wb'))
```

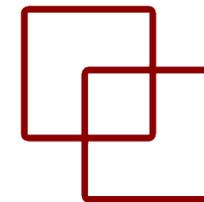
```
# load the model from disk
loaded_model = pkl.load(open(filename, 'rb'))
results = loaded_model.score(X_test, Y_test)
print(f"Accuracy: {results.mean()*100.0:,.2f}%")
```

Accuracy: 78.74%



## 3. Joblib (I)

- La biblioteca *Joblib* es parte del ecosistema *SciPy* y proporciona utilidades para canalizar trabajos de Python.
  - Proporciona utilidades para guardar y cargar objetos de Python que utilizan las estructuras de datos de NumPy de manera eficiente.
- La ejecución del ejemplo guarda el modelo en un archivo como *model.sav* finalizado y también crea un archivo para cada matriz NumPy en el modelo (cuatro archivos adicionales).
- Debemos instalar el paquete *joblib*



## 3. Joblib (II)

```
# Save Model Using joblib
import joblib as jbl
#from sklearn.externals.joblib import load

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=7)
# Fit the model on 33%
model = LogisticRegression(solver= 'lbfgs', max_iter=1000)
model.fit(X_train, Y_train)
# save the model to disk
filename = 'finalized_model.sav'
jbl.dump(model, filename)
```

```
['finalized_model.sav']
```

```
# load the model from disk
loaded_model = jbl.load(filename)
result = loaded_model.score(X_test, Y_test)
print(f"Accuracy: {result.mean()*100.0:,.2f}%")
```

Accuracy: 78.74%

# ¡GRACIAS!



**Dr. Manuel Castillo-Cara**  
**[www.manuelcastillo.eu](http://www.manuelcastillo.eu)**

**Departamento de Inteligencia Artificial**  
**Escuela Técnica Superior de Ingeniería Informática**  
**Universidad Nacional de Educación a Distancia (UNED)**