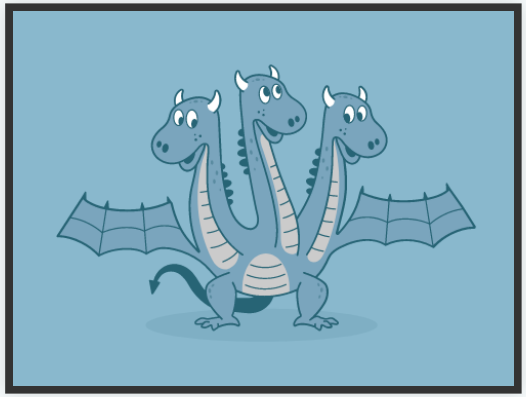
FALLANDO HASTA EL ÉXITO CON HYDRA



datacubeR.cl
github.com/datacubeR

```
Hola, soy Alfonso!
Head de Data Science en Jooycar
```

ACERCA DE MÍ

ACERCA DE MÍ

- Estoy casado con Valentina.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.
- Me encanta el Machine Learning, no sólo como trabajo sino como **Hobbie**.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.
- Me encanta el Machine Learning, no sólo como trabajo sino como **Hobbie**.
- Tengo un **Blog** en el que comparto material de **Data Science** en Español. datacubeR.cl

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.
- Me encanta el Machine Learning, no sólo como trabajo sino como **Hobbie**.
- Tengo un **Blog** en el que comparto material de **Data Science** en Español. datacubeR.cl
- Gané el Desafío **Itaú-Binnario 2020**.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.
- Me encanta el Machine Learning, no sólo como trabajo sino como **Hobbie**.
- Tengo un **Blog** en el que comparto material de **Data Science** en Español. datacubeR.cl
- Gané el Desafío **Itaú-Binnario 2020**.
- **No ocupé nada** de lo que les voy a presentar ahora en la competencia desafío, y casi me costó caro.

ACERCA DE MÍ

- Estoy casado con Valentina.
- Tengo 2 Gatitas y una Perrita.
- Llevo 6 años trabajando como *Data Scientist*.
- Soy Docente de la Carrera de Data Science en la Academia Desafio Latam.
- Me encanta el Machine Learning, no sólo como trabajo sino como **Hobbie**.
- Tengo un **Blog** en el que comparto material de **Data Science** en Español. datacubeR.cl
- Gané el Desafío **Itaú-Binnario 2020**.
- **No ocupé nada** de lo que les voy a presentar ahora en la competencia desafío, y casi me costó caro.
- Pero a pesar de no usar esto, me di cuenta de la necesidad de ser ordenado en la etapa de Experimentación.

EXPECTATIVAS DE LA CHARLA

QUÉ APRENDEREMOS

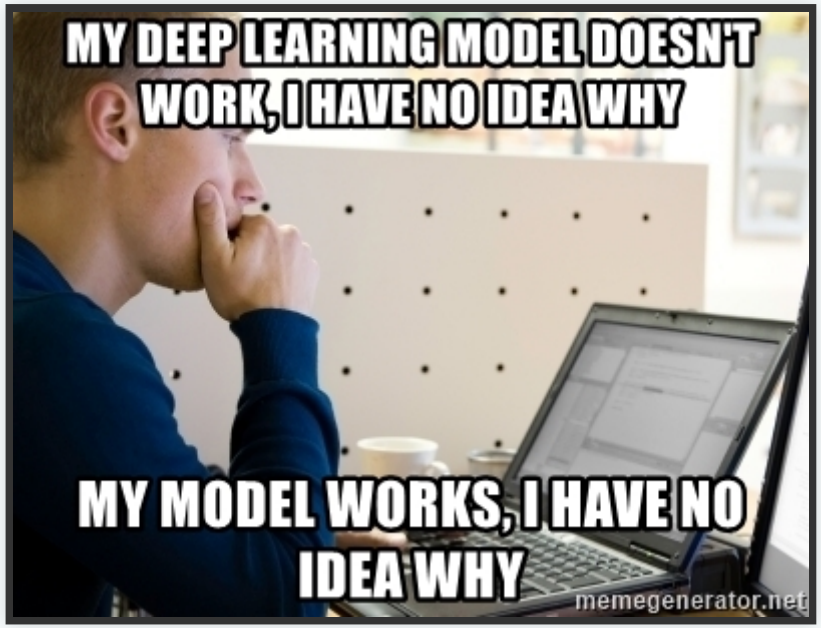
- **Hydra**, un **framework** de para generar scripts con **Configuraciones Complejas**.
- Cómo entrenar **varios modelos** de Machine Learning de manera ordenada usando **Hydra**.
- Generar **Scripts de Experimentación** para la búsqueda del **Modelo Óptimo**.
- Aprender cómo **reutilizar un código** para múltiples modelos distintos.

QUÉ **NO** APRENDEREMOS

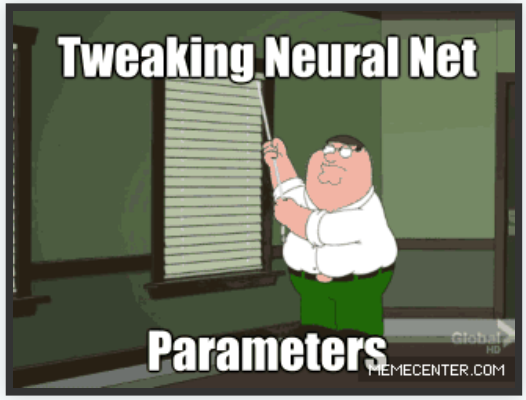
- Qué es el Machine Learning.
- Qué es un archivo YAML.
- Cómo entrenar un modelo en **Scikit-Learn**.
- Conceptos propios del Machine Learning como **Preprocesamiento**, **Cross Validation**, **Hiperparámetros**, **Evaluación de un Modelo**.
- Cómo elegir el mejor modelo y ponerlo en **producción**.
- Cómo evitar el código.

ENTRENANDO UN MODELO DE MACHINE LEARNING

datacubeR


MY DEEP LEARNING MODEL DOESN'T WORK, I HAVE NO IDEA WHY
MY MODEL WORKS, I HAVE NO IDEA WHY
memegenerator.net

MY MODEL COMPILES PERFECTLY
BUT ALL IT PREDICTS IS 0.0
imgflip.com



MODELO ÓPTIMO

- Un Algoritmo
- un Set de Datos
- Set de Hiperparámetros

HAY QUE FALLAR MUCHAS VECES HASTA ENCONTRAR ESE MODELO “EXITOSO”.

EL PROBLEMA

QUEREMOS ENTRENAR MUCHOS MODELOS PARA RESOLVER LO SIGUIENTE:

Un Modelo de Clasificación Binaria.

Set de Entrenamiento y Test en dos tamaños:

- Small (sólo features numéricas)
- Large (incluye también features categóricas)

Es necesario hacer un preprocesamiento:

- Variables numéricas y categóricas contienen NAs.
- Es necesario usar Encoding en Variables Categóricas.

Queremos Probar 2 modelos distintos para resolver el problema:

- Regresión Logística
- Random Forest

VAMOS A RESOLVER EL PROBLEMA UTILIZANDO SCIKIT-LEARN.

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Usamos **pandas** para importar la data.

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Importamos el Modelo que nos interesa

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Algunas Clases amigas para trabajar en Scikit-Learn.

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Una Clase de Feature-Engine para imputar.

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Leemos la data y separamos predictores y Targets

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Preprocesamiento

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Entrenamiento del Modelo

UNA REGRESIÓN LOGÍSTICA EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train = train_df_small.drop(columns = 'Survived')
y_train = train_df_small.Survived
X_test,  = test_df_small.drop(columns = 'Survived')
y_test =  test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Medición de la Performance

UN RANDOM FOREST EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', RandomForestClassifier(
        n_estimators = 100,
        max_depth = 5,
        min_samples_split = 10,
        random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

UN RANDOM FOREST EN SCIKIT-LEARN

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', RandomForestClassifier(
        n_estimators = 100,
        max_depth = 5,
        min_samples_split = 10,
        random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Importamos otro modelo

UN RANDOM FOREST EN SCIKIT-LEARN

```python
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from feature_engine.imputation import MeanMedianImputer

train_df_small = pd.read_csv('data/data-train-small.csv')
test_df_small = pd.read_csv('data/data-test-small.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', RandomForestClassifier(
        n_estimators = 100,
        max_depth = 5,
        min_samples_split = 10,
        random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

El modelo es distinto

Y SI QUISIERA CAMBIAR MI DATASET EN LA REGRESIÓN LOGÍSTICA?

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder
from sklearn.metrics import accuracy_score

train_df_small = pd.read_csv('data/data-train-large.csv')
test_df_small = pd.read_csv('data/data-test-large.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('imp_cat', CategoricalImputer(imputation_method='frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Y SI QUISIERA CAMBIAR MI DATASET EN LA REGRESIÓN LOGÍSTICA?

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder
from sklearn.metrics import accuracy_score

train_df_small = pd.read_csv('data/data-train-large.csv')
test_df_small = pd.read_csv('data/data-test-large.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('imp_cat', CategoricalImputer(imputation_method='frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Cambio en la Data

Y SI QUISIERA CAMBIAR MI DATASET EN LA REGRESIÓN LOGÍSTICA?

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder
from sklearn.metrics import accuracy_score

train_df_small = pd.read_csv('data/data-train-large.csv')
test_df_small = pd.read_csv('data/data-test-large.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('imp_cat', CategoricalImputer(imputation_method='frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Cambio en el Preprocesamiento

Y SI QUISIERA CAMBIAR MI DATASET EN LA REGRESIÓN LOGÍSTICA?

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder
from sklearn.metrics import accuracy_score

train_df_small = pd.read_csv('data/data-train-large.csv')
test_df_small = pd.read_csv('data/data-test-large.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('imp_cat', CategoricalImputer(imputation_method='frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Y si quisieramos cambiar los Hiperparámetros?

Y SI QUISIERA CAMBIAR MI DATASET EN LA REGRESIÓN LOGÍSTICA?

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder
from sklearn.metrics import accuracy_score

train_df_small = pd.read_csv('data/data-train-large.csv')
test_df_small = pd.read_csv('data/data-test-large.csv')
X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

preprocess = Pipeline(steps = [
    ('imp_num', MeanMedianImputer(imputation_method='mean')),
    ('imp_cat', CategoricalImputer(imputation_method='frequent')),
    ('ohe', OneHotEncoder()),
    ('sc', StandardScaler())
])

model_pipe = Pipeline(steps = [
    ('prep', preprocess),
    ('model', LogisticRegression(random_state=123))
])

model_pipe.fit(X_train,y_train)

y_pred = model_pipe.predict(X_test)
print("El accuracy obtenido es:", accuracy_score(y_test, y_pred))
```

Distintos Modelos tienen distintos Hiperparámetros

HYDRA

- Utiliza archivos de Configuración para poder cambiar rápidamente el código.
    - Instanciar clases.
    - Invocar Funciones
    - Llamar parámetros

- Entregará todos los outputs del código en una carpeta aparte del tipo:
    - outputs/fecha/hora
    - Dentro de cada carpeta se puede guardar lo que sea además de que se almacena la configuración utilizada en el proceso.

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   └── config.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

2 directories, 5 files
```

CONFIG.YAML

```yaml
name: Logistic Regression
data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
C: 1.0
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Agregamos Hydra y OmegaConf

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Envolvemos todo en una función

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Decoramos la función función

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Corregimos los Paths

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Podemos modificar parámetros del Modelo

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path

import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from feature_engine.imputation import MeanMedianImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method='mean')),
        ('sc', StandardScaler())
    ])

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(C = cfg.C, random_state=123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.name} es:", accuracy_score(y_test, y_pred))

if __name__ == '__main__':
    train_model()
```

Llamamos la función en nuestro Script

EJECUCIÓN DEL SCRIPT

```
python main.py
```

++ genera un override del parámetro C.

```
python main.py ++C=0.1
```

++ genera un override del parámetro C y los datasets.

```
python main.py ++C=0.1 ++data_train=data/data-train-large.csv ++data_test=data/data-test-large.csv
```

PROBLEMA

El procesamiento no es el adecuado para el dataset Large

INVOCAR FUNCIONES

Sabemos que debemos utilizar distintos preprocesamientos dependiendo de la data a utilizar, para ellos modularizaremos los preprocesamientos de la siguiente manera:

PREPROCESSING.PY

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder

def simple_preprocess(imputation_method):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_method)),
        ('sc', StandardScaler())
    ])
    return preprocess

def complex_preprocess(imputation_num, imputation_cat):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_num)),
        ('imp_cat', CategoricalImputer(imputation_method=imputation_cat)),
        ('ohe', OneHotEncoder()),
        ('sc', StandardScaler())
    ])
    return preprocess
```

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   ├── config.yaml
│   └── preprocess
│       ├── complex.yaml
│       └── simple.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

3 directories, 7 files
```

INVOCAR FUNCIONES

Sabemos que debemos utilizar distintos preprocesamientos dependiendo de la data a utilizar, para ellos modularizaremos los preprocesamientos de la siguiente manera:

PREPROCESSING.PY

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder

def simple_preprocess(imputation_method):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_method)),
        ('sc', StandardScaler())
    ])
    return preprocess

def complex_preprocess(imputation_num, imputation_cat):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_num)),
        ('imp_cat', CategoricalImputer(imputation_method=imputation_cat)),
        ('ohe', OneHotEncoder()),
        ('sc', StandardScaler())
    ])
    return preprocess
```

Generará un Preprocesamiento Simple

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   ├── config.yaml
│   └── preprocess
│       ├── complex.yaml
│       └── simple.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

3 directories, 7 files
```

INVOCAR FUNCIONES

Sabemos que debemos utilizar distintos preprocesamientos dependiendo de la data a utilizar, para ellos modularizaremos los preprocesamientos de la siguiente manera:

PREPROCESSING.PY

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder

def simple_preprocess(imputation_method):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_method)),
        ('sc', StandardScaler())
    ])
    return preprocess

def complex_preprocess(imputation_num, imputation_cat):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_num)),
        ('imp_cat', CategoricalImputer(imputation_method=imputation_cat)),
        ('ohe', OneHotEncoder()),
        ('sc', StandardScaler())
    ])
    return preprocess
```

Generará un Preprocesamiento más Complejo

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   ├── config.yaml
│   └── preprocess
│       ├── complex.yaml
│       └── simple.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

3 directories, 7 files
```

INVOCAR FUNCIONES

Sabemos que debemos utilizar distintos preprocesamientos dependiendo de la data a utilizar, para ellos modularizaremos los preprocesamientos de la siguiente manera:

PREPROCESSING.PY

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder

def simple_preprocess(imputation_method):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_method)),
        ('sc', StandardScaler())
    ])
    return preprocess

def complex_preprocess(imputation_num, imputation_cat):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=imputation_num)),
        ('imp_cat', CategoricalImputer(imputation_method=imputation_cat)),
        ('ohe', OneHotEncoder()),
        ('sc', StandardScaler())
    ])
    return preprocess
```

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   ├── config.yaml
│   └── preprocess
│       ├── complex.yaml
│       └── simple.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

3 directories, 7 files
```

LOS ARCHIVOS DE CONFIGURACIÓN TOMAN OTRO ROL:

- `config.yaml` ahora corresponderá a toda la configuración general. Elementos que afecten al proceso completo.
- Y crearemos una subcarpeta llamada `preprocess` que contendrá configuraciones específicas para cada preprocesamiento.

simple.yaml

```yaml
type:
  _target_: main.simple_preprocess
  imputation_method: mean

data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
```

complex.yaml

```yaml
type:
  _target_: main.complex_preprocess
  imputation_num: mean
  imputation_cat: frequent

data_train: data/data-train-large.csv
data_test: data/data-test-large.csv
```

config.yaml

```yaml
C: 1.0
```

LOS ARCHIVOS DE CONFIGURACIÓN TOMAN OTRO ROL:

- `config.yaml` ahora corresponderá a toda la configuración general. Elementos que afecten al proceso completo.
- Y crearemos una subcarpeta llamada `preprocess` que contendrá configuraciones específicas para cada preprocesamiento.

simple.yaml

```yaml
type:
  _target_: main.simple_preprocess
  imputation_method: mean

data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
```

Invocaremos la función y sus parámetros

complex.yaml

```yaml
type:
  _target_: main.complex_preprocess
  imputation_num: mean
  imputation_cat: frequent

data_train: data/data-train-large.csv
data_test: data/data-test-large.csv
```

config.yaml

```yaml
C: 1.0
```

LOS ARCHIVOS DE CONFIGURACIÓN TOMAN OTRO ROL:

- `config.yaml` ahora corresponderá a toda la configuración general. Elementos que afecten al proceso completo.
- Y crearemos una subcarpeta llamada `preprocess` que contendrá configuraciones específicas para cada preprocesamiento.

simple.yaml

```yaml
type:
  _target_: main.simple_preprocess
  imputation_method: mean

data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
```

Ojo los Dataset dependen ahora del Preprocesamiento

complex.yaml

```yaml
type:
  _target_: main.complex_preprocess
  imputation_num: mean
  imputation_cat: frequent

data_train: data/data-train-large.csv
data_test: data/data-test-large.csv
```

config.yaml

```yaml
C: 1.0
```

LOS ARCHIVOS DE CONFIGURACIÓN TOMAN OTRO ROL:

- `config.yaml` ahora corresponderá a toda la configuración general. Elementos que afecten al proceso completo.
- Y crearemos una subcarpeta llamada `preprocess` que contendrá configuraciones específicas para cada preprocesamiento.

simple.yaml

```yaml
type:
  _target_: main.simple_preprocess
  imputation_method: mean

data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
```

complex.yaml

```yaml
type:
  _target_: main.complex_preprocess
  imputation_num: mean
  imputation_cat: frequent

data_train: data/data-train-large.csv
data_test: data/data-test-large.csv
```

config.yaml

```yaml
C: 1.0
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Importamos los preprocesamientos

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Modificamos los paths de la data

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Invocamos el preprocesamiento

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Modificar Hiperparámetros

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

MI SCRIPT EN HYDRA

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', LogisticRegression(cfg.C, random_state = 123))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

El operador + indica qué configuración usar:

```
python main.py +preprocess=simple
```

```
python main.py +preprocess=complex ++C=0.1
```

INSTANCIAR OBJETOS

Por ejemplo si quisieramos tener un Script que permita cambiar entre Regresión Logística y Random Forest necesitaremos instanciar distintas clases. Esto porque ambos modelos poseen distintos parámetros:

```yaml
name: Logistic Regression
type:
  _target_: main.LogisticRegression
  C: 1
  random_state: 123
```

```yaml
name: Random Forest
type:
  _target_: main.RandomForestClassifier
  n_estimators: 100
  max_depth: 5
  min_samples_split: 10
  n_jobs: -1
  random_state: 123
```

```
~/Desktop/hydra_tutorial
tree
.
├── conf
│   ├── config.yaml
│   ├── models
│   │   ├── lr.yaml
│   │   └── rf.yaml
│   └── preprocess
│       ├── complex.yaml
│       └── simple.yaml
└── data
    ├── data-test-large.csv
    ├── data-test-small.csv
    ├── data-train-large.csv
    └── data-train-small.csv

4 directories, 9 files
```

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', hydra.utils.instantiate(cfg.models.type))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.models.name} es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

EJECUTAR EL SCRIPT

```
python main.py +preprocess=simple +models=lr
```

```
python main.py +preprocess=complex +models=rf
```

```
python main.py +preprocess=complex +models=lr ++models/C=0.1
```

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', hydra.utils.instantiate(cfg.models.type))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.models.name} es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Es necesario importar ambos Modelos

EJECUTAR EL SCRIPT

```
python main.py +preprocess=simple +models=lr
```

```
python main.py +preprocess=complex +models=rf
```

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', hydra.utils.instantiate(cfg.models.type))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.models.name} es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

Es necesario importar ambos Modelos

EJECUTAR EL SCRIPT

```
python main.py +preprocess=simple +models=lr
```

```
python main.py +preprocess=complex +models=rf
```

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import pandas as pd
from preprocessing import simple_preprocess, complex_preprocess
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score

@hydra.main(config_path ='conf', config_name = 'config')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    preprocess = hydra.utils.call(cfg.preprocess.type)

    model_pipe = Pipeline(steps = [
        ('prep', preprocess),
        ('model', hydra.utils.instantiate(cfg.models.type))
    ])

    model_pipe.fit(X_train,y_train)
    y_pred = model_pipe.predict(X_test)

    print(f"El Accuracy obtenido por {cfg.models.name} es:", accuracy_score(y_test, y_pred))


if __name__ == '__main__':
    train_model()
```

EJECUTAR EL SCRIPT

```
python main.py +preprocess=simple +models=lr
```

```
python main.py +preprocess=complex +models=rf
```

```
python main.py +preprocess=complex +models=lr ++models/C=0.1
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

MUCHAS COMBINACIONES CON OPTUNA

- **Optuna** es una librería de Opttimización Bayesiana ultra rápida.
- Basa su búsqueda en sus **resultados anteriores**.

```python
from omegaconf import DictConfig
import hydra
from hydra.utils import to_absolute_path
import optuna
import pandas as pd
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from models import random_forest, log_reg
from preprocessing_multirun import simple_preprocess, complex_preprocess
from optuna.samplers import TPESampler

@hydra.main(config_name='config', config_path='conf')
def train_model(cfg: DictConfig):
    train_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_train))
    test_df_small = pd.read_csv(to_absolute_path(cfg.preprocess.data_test))
    X_train, y_train = train_df_small.drop(columns = 'Survived'), train_df_small.Survived
    X_test, y_test = test_df_small.drop(columns = 'Survived'), test_df_small.Survived

    def optimize_model(trial):
        preprocess = hydra.utils.call(cfg.preprocess.type, trial = trial)

        model_pipe = Pipeline(steps = [
            ('prep', preprocess),
            ('model', hydra.utils.call(cfg.models.type, trial = trial))
        ])

        model_pipe.fit(X_train,y_train)
        y_pred = model_pipe.predict(X_test)
        return accuracy_score(y_test, y_pred)

    sampler = TPESampler(seed=123)
    study = optuna.create_study(sampler = sampler, direction="maximize")
    study.optimize(optimize_model, n_trials=cfg.n_trials)

    print(f'El mejor accuracy conseguido fue: {study.best_value}')
    print(f'usando los siguientes parámetros: \n \t \t{study.best_params}')
```

PREPROCESSING.PY

```python
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from feature_engine.imputation import CategoricalImputer, MeanMedianImputer
from feature_engine.encoding import OneHotEncoder

def simple_preprocess(imp_method_values, trial=None):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=trial.suggest_categorical("imp_num", imp_method_values))),
        ('sc', StandardScaler())
    ])
    return preprocess

def complex_preprocess(imputation_num, imputation_cat, trial=None):
    preprocess = Pipeline(steps = [
        ('imp_num', MeanMedianImputer(imputation_method=trial.suggest_categorical("imp_num", imputation_num))),
        ('imp_cat', CategoricalImputer(imputation_method=trial.suggest_categorical("imp_cat", imputation_cat))),
        ('ohe', OneHotEncoder()),
        ('sc', StandardScaler())
    ])
    return preprocess
```

MODELS.PY

```python
def random_forest(n_estimators, max_depth, min_samples_split, trial = None):
    return RandomForestClassifier(n_estimators = trial.suggest_int('n_estimators', **n_estimators),
                                  max_depth = trial.suggest_int('max_depth', **max_depth),
                                  min_samples_split = trial.suggest_discrete_uniform('min_samples_split', **min_samples_sp
                                  n_jobs = -1,
                                  random_state = 123
                                  )

def log_reg(C, fit_intercept, trial = None):
    return LogisticRegression(C = trial.suggest_loguniform('C', **C),
                              fit_intercept=trial.suggest_categorical('fit_intercept', fit_intercept),
                              n_jobs = -1,
                              random_state = 123
                              )
```

LR.YAML

```yaml
name: Multi Logistic Regression
type:
  _target_: main.log_reg
  C:
    low: 0.001
    high: 100
  fit_intercept: [false, true]
```

RF.YAML

```yaml
name: Multi Random Forest
type:
  _target_: main.random_forest
  n_estimators:
    low: 10
    high: 600
    step: 10
  max_depth:
    low: 1
    high: 10
    step: 1
  min_samples_split:
    low: 0.1
    high: 1
    q: 0.1
```

SIMPLE.YAML

```yaml
type:
  _target_: main.simple_preprocess
  imp_method_values: [mean, median]

data_train: data/data-train-small.csv
data_test: data/data-test-small.csv
```

COMPLEX.YAML

```yaml
type:
  _target_: main-multirun.complex_preprocess
  imputation_num: [mean, median]
  imputation_cat: [frequent, missing]

data_train: data/data-train-large.csv
data_test: data/data-test-large.csv
```

CONFIG.YAML

```yaml
n_trials: ???
```

EJECUCIÓN MULTIRUN

```
python main.py -m +preprocess=simple,complex +models=lr,rf +n_trials=100
```

400 Modelos con un sólo Script.

MUCHAS GRACIAS

Los desafío a que me den el mejor accuracy que se puede lograr en este problema y el set de Hiperparámetros que lo logra.

github.com/datacubeR/pycon2021