

# Practica usando SAFE (Surrogate Assisted Feature Extraction)

Sistemas de apoyo para la toma de  
decisiones

Prof. Jessica Carmin Mendiola Fuentes

Ingeniería en Datos e  
Inteligencia Organizacional

Presenta:

Meneses Ballón Oscar Enrique

Fecha de entrega:  
31/03/2023

## Introducción

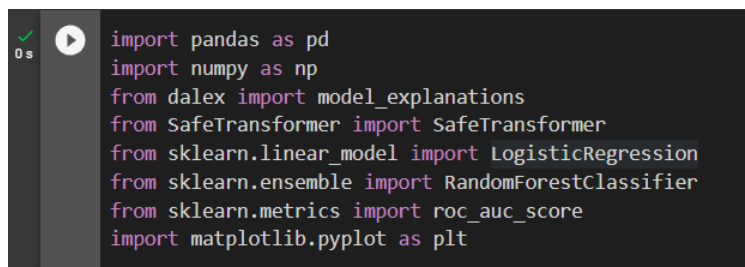
La implementación de SAFE ML, por sus siglas en español Extracción de Características Asistida por Surrogado para el Aprendizaje de Modelos, es una técnica de procesamiento de datos útil para evitar uno de los problemas más comunes que pueden ocurrir al estar creando modelos y ajustes a los datos. El sobreajuste, por eso se usa SAFE para mejorar el rendimiento de los modelos de aprendizaje automático.

En esta práctica, se usan dos conjuntos de datos de dos hoteles diferentes para mostrar cómo esta técnica puede ayudar a mejorar la precisión de los modelos de regresión lineal y de Random Forest.

## Desarrollo

Para empezar la práctica, se verá el problema del **sobreajuste**, esto ocurre cuando un modelo se ajusta demasiado bien a los datos de **entrenamiento** y, como resultado, tiene un mal desempeño en los datos de **prueba**. Para demostrar este problema, utilizare un conjunto de datos de un hotel y pondré a entrenar un modelo de regresión lineal y un modelo de Random Forest.

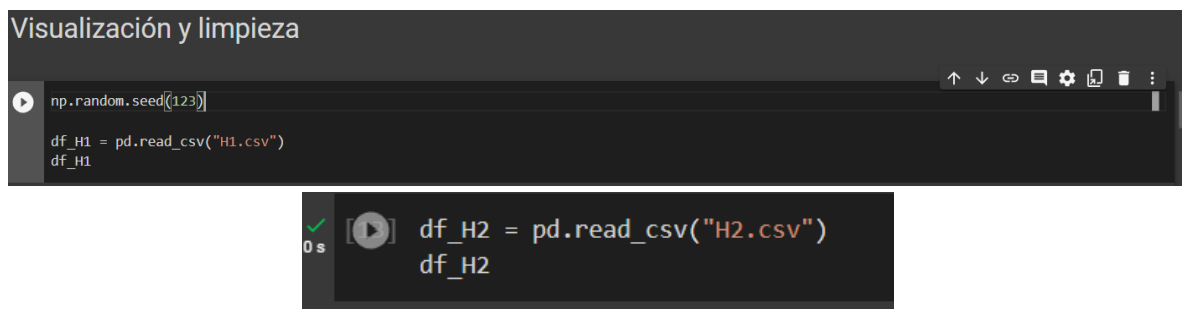
Primero antes que nada se obtienen las librerías necesarias.



```
import pandas as pd
import numpy as np
from dalex import model_explanations
from SafeTransformer import SafeTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt
```

**Nota:** La mayoría de ellas se deben instalar mediante el comando `!pip install` y luego importar las librerías en collab.

Siguiente se hace la carga de los datos de los Hoteles.



Visualización y limpieza

```
np.random.seed(123)

df_H1 = pd.read_csv("H1.csv")
df_H1
```

```
df_H2 = pd.read_csv("H2.csv")
df_H2
```

Visualización general de los datos:

	IsCanceled	LeadTime	ArrivalDateYear	ArrivalDateMonth	ArrivalDateWeekNumber	ArrivalDateDayOfMonth	StaysInWeekendNights	StaysInWeekNights	Adults	Children	...	DepositType	Agent	Company	Day
0	0	342	2015	July	27	1	0	0	2	0	...	No Deposit	NULL	NULL	
1	0	737	2015	July	27	1	0	0	2	0	...	No Deposit	NULL	NULL	
2	0	7	2015	July	27	1	0	1	1	0	...	No Deposit	NULL	NULL	
3	0	13	2015	July	27	1	0	1	1	0	...	No Deposit	304	NULL	
4	0	14	2015	July	27	1	0	2	2	0	...	No Deposit	240	NULL	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
40055	0	212	2017	August	35	31	2	8	2	1	...	No Deposit	143	NULL	
40056	0	169	2017	August	35	30	2	9	2	0	...	No Deposit	250	NULL	
40057	0	204	2017	August	35	29	4	10	2	0	...	No Deposit	250	NULL	
40058	0	211	2017	August	35	31	4	10	2	0	...	No Deposit	40	NULL	
40059	0	161	2017	August	35	31	4	10	2	0	...	No Deposit	69	NULL	

40060 rows x 31 columns

La dimensión para H1 son de 40,060 filas y 31 columnas. Y tiene datos nulos.

df_H1.dtypes	
IsCanceled	int64
LeadTime	int64
ArrivalDateYear	int64
ArrivalDateMonth	object
ArrivalDateWeekNumber	int64
ArrivalDateDayOfMonth	int64
StaysInWeekendNights	int64
StaysInWeekNights	int64
Adults	int64
Children	int64
Babies	int64
Meal	object
Country	object
MarketSegment	object
DistributionChannel	object
IsRepeatedGuest	int64
PreviousCancellations	int64
PreviousBookingsNotCanceled	int64
ReservedRoomType	object
AssignedRoomType	object
BookingChanges	int64
DepositType	object
Agent	object
Company	object
DaysInWaitingList	int64
CustomerType	object
ADR	float64
RequiredCarParkingSpaces	int64
TotalOfSpecialRequests	int64
ReservationStatus	object
ReservationStatusDate	object
dtype: object	

Estas son todas las columnas y su tipo de dato que tiene cada una se puede ver que hay algunas no numericas pero la mayoría son numericas. Por lo que se guardara en otra variable con las variables importantes. Para la cantidad de datos nulos,

df_H1.isna().sum()	
IsCanceled	0
LeadTime	0
ArrivalDateYear	0
ArrivalDateMonth	0
ArrivalDateWeekNumber	0
ArrivalDateDayOfMonth	0
StaysInWeekendNights	0
StaysInWeekNights	0
Adults	0
Children	0
Babies	0
Meal	0
Country	464
MarketSegment	0
DistributionChannel	0
IsRepeatedGuest	0
PreviousCancellations	0
PreviousBookingsNotCanceled	0
ReservedRoomType	0
AssignedRoomType	0
BookingChanges	0
DepositType	0
Agent	0
Company	0
DaysInWaitingList	0
CustomerType	0
ADR	0
RequiredCarParkingSpaces	0
TotalOfSpecialRequests	0
ReservationStatus	0
ReservationStatusDate	0
dtype: int64	

Para el analisis me quedo solo con aquellas variables numericas, y le digo que tome un sample de 20,000 datos aleatorios y que se guarden en una random seed para ser reproducibles.

Estas columnas son irrelevantes para el analisis:

- "ReservationStatus",
- "ReservationStatusDate",
- "Company",
- "Agent",
- "ArrivalDateMonth",
- "Meal",
- "MarketSegment",
- "DistributionChannel",
- "ReservedRoomType",
- "AssignedRoomType",
- "DepositType"
- "CustomerType"

Creo mi variables de entrenamiento y prueba con los 20 mil datos escogidos aleatorios.

```

n = hotel_bookings.shape[0]
train_idx = np.random.choice(n, int(0.8 * n), replace=False)
train = hotel_bookings.iloc[train_idx]
test = hotel_bookings.iloc[np.setdiff1d(np.arange(n), train_idx)]

[46] x_train = train.drop(columns=["IsCanceled"])
     y_train = train["IsCanceled"]

     x_test = test.drop(columns=["IsCanceled"])
     y_test = test["IsCanceled"]

```

X serán las variables predictoras.

Y la variable de respuesta "IsCanceled".

Siguiente realizo un modelo con Random Forest para obtener el AUC-ROC. Y obtengo los siguientes resultados.

```
### Random Forest ###

# Crear un modelo de Random Forest
model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=123)

# Entrenar el modelo con los datos de entrenamiento
model.fit(x_train, y_train)

# Predecir la probabilidad de que cada registro del conjunto de entrenamiento sea de la clase positiva (y = 1)
pred_train = model.predict_proba(x_train)[:, 1]

# Calcular el AUC-ROC del modelo con los datos de entrenamiento
roc_auc_train = roc_auc_score(y_train, pred_train)
print('AUC-ROC en entrenamiento: {:.2f}'.format(roc_auc_train))

# Se predice las veces que un cliente pueda cancelar su reserva
pred_test = model.predict_proba(x_test)[:, 1]
print("Predicciones del modelo:" + str(pred_test))
# Se usan ahora los datos de prueba para el AUC-ROC
roc_auc_test = roc_auc_score(y_test, pred_test)
print('AUC-ROC en prueba: {:.2f}'.format(roc_auc_test))
```

AUC-ROC en entrenamiento: 0.88  
Predicciones del modelo:[0.06262771 0.00537389 0.07354657 ... 0.04396817 0.29629015 0.35238848]  
AUC-ROC en prueba: 0.84

Como se puede ver, este modelo en los datos de entrenamiento funciona decentemente, un .8 para arriba es un funcionamiento decente sin embargo hay que destacar el problema que ocurre al utilizar los de prueba. Esta baja en un .04.

El AUC-ROC es nos dice la calidad de un modelo de **clasificación binaria** que mide el rendimiento del modelo en términos de la tasa de verdaderos positivos y la tasa de falsos positivos.

Esto **sugiere** que el modelo puede estar **sobreajustando** los datos de entrenamiento y no generalizando bien a nuevos datos. El modelo aún tiene un buen rendimiento en los datos de prueba.

Lo siguiente para hacer una demostración de los problemas que pueden surgir del sobreajustado de los datos, ahora se usará un modelo lineal.

## Modelo lineal

```
[54] ### Linear model ###  
      from sklearn.preprocessing import StandardScaler  
  
      # Escalando los datos  
      scaler = StandardScaler()  
      x_train_scaled = scaler.fit_transform(x_train)  
      x_test_scaled = scaler.transform(x_test)  
  
      # Creando el modelo  
      model_lm1 = LogisticRegression(random_state=123, max_iter=500)  
      model_lm1.fit(x_train_scaled, y_train)  
  
      # Haciendo predicciones y evaluando  
      pred_train = model_lm1.predict_proba(x_train_scaled)[:,-1]  
      1 - roc_auc_score(y_train, pred_train)  
  
      pred_test = model_lm1.predict_proba(x_test_scaled)[:,-1]  
      1 - roc_auc_score(y_test, pred_test)  
  
0.22150711689286628
```

El valor que obtenido fue un 0.22 corresponde este valor corresponde a la resta de 1 - AUC-ROC en el conjunto de prueba del modelo de regresión logística. Hay que recordar que un AUC-ROC cercano a 1 indica un buen rendimiento del modelo, mientras que un valor cercano a 0.5 indica un rendimiento aleatorio, y en este caso se obtuvo algo incluso menor a ello.

Hay múltiples razones por las cuales puede ocurrir estas situaciones, pero lo principal que esto indica es que de nuevo el modelo tiene un rendimiento pobre en el conjunto de prueba. Es posible que el modelo tenga problemas de sobreajuste o subajuste, o que no se hayan seleccionado las mejores características para el modelo.

Para la etapa de la implementación de SAFE tuve bastante inconvenientes y Alertas del procesamiento que hasta ahora no he sido capaz de resolver. Pero esto es lo que obtuve al final:

```
1 min  ### SAFE ###
safe = SafeTransformer(model=LogisticRegression())
safe.fit(X=x_train, y=y_train)

train_trans_new = safe.transform(x_train)
test_trans_new = safe.transform(x_test)

model_glm = LogisticRegression(random_state=123)
model_glm.fit(X=train_trans_new, y=y_train)

pred_train = model_glm.predict_proba(train_trans_new)[:,-1]
1 - roc_auc_score(y_train, pred_train)

pred_test = model_glm.predict_proba(test_trans_new)[:,-1]
1 - roc_auc_score(y_test, pred_test)
```

✓ 1 min 47 s completado a las 21:51

El procesamiento de este demoró un poco más de minuto y medio.

```
Proceso SAFE

1 min  ### SAFE ###
safe = SafeTransformer(model=LogisticRegression())
safe.fit(X=x_train, y=y_train)

train_trans_new = safe.transform(x_train)
test_trans_new = safe.transform(x_test)

model_glm = LogisticRegression(random_state=123)
model_glm.fit(X=train_trans_new, y=y_train)

pred_train = model_glm.predict_proba(train_trans_new)[:,-1]
1 - roc_auc_score(y_train, pred_train)

pred_test = model_glm.predict_proba(test_trans_new)[:,-1]
1 - roc_auc_score(y_test, pred_test)
```

✖ /usr/local/lib/python3.9/dist-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>  
Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)  
n\_iter\_i = \_check\_optimize\_result(  
/usr/local/lib/python3.9/dist-packages/sklearn/linear\_model/\_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):  
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max\_iter) or scale the data as shown in:  
<https://scikit-learn.org/stable/modules/preprocessing.html>  
Please also refer to the documentation for alternative solver options:  
[https://scikit-learn.org/stable/modules/linear\\_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)  
n\_iter\_i = \_check\_optimize\_result(  
0.3876476409549774

Y este fue el resultado.

Este es el valor restado del área bajo la curva, si es restado con 1, significa que roc\_auc ha de ser de .61 que por lo menos es mayor a .5 pero sigue siendo malo para evaluar en los de prueba.

Si se hace la misma prueba directamente con el segundo data set de hoteles se obtiene un resultado similar:

```
[62] #Se quitan las columnas con elementos NULL
hotel_bookings = df_H1.loc[:, df_H2.notna().all()]
hotel_bookings = hotel_bookings.drop(columns=["ReservationStatus", "ReservationStatusDate", "Company", "
hotel_bookings = hotel_bookings.sample(n=20000, random_state=123)

[63] n = hotel_bookings.shape[0]
train_idx = np.random.choice(n, int(0.8 * n), replace=False)
train = hotel_bookings.iloc[train_idx]
test = hotel_bookings.iloc[np.setdiff1d(np.arange(n), train_idx)]

[64] x_train = train.drop(columns=["IsCanceled"])
y_train = train["IsCanceled"]

x_test = test.drop(columns=["IsCanceled"])
y_test = test["IsCanceled"]

[65] ### SAFE ###
safe = SafeTransformer(model=LogisticRegression())
safe.fit(X=x_train, y=y_train)

train_trans_new = safe.transform(x_train)
test_trans_new = safe.transform(x_test)

model_glm = LogisticRegression(random_state=123)
model_glm.fit(X=train_trans_new, y=y_train)

pred_train = model_glm.predict_proba(train_trans_new)[:,-1]
1 - roc_auc_score(y_train, pred_train)

pred_test = model_glm.predict_proba(test_trans_new)[:,-1]
1 - roc_auc_score(y_test, pred_test)
```

Aquí se repite el proceso de creación de los datos de entrenamiento y prueba para el dataset de H2, y luego directo a ser evaluado con SAFE.

Y el resultado obtenido fue:

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
n_iter_i = _check_optimize_result(
0.30181606429332697
```



La visualización de los datos esta pendiente por hacer, por lo que no se pueden tomar decisiones seguras con esta calidad de modelos aún.

## Conclusiones

Para finalizar hay varios puntos por destacar, primero el sobreajuste es un problema común en los modelos de aprendizaje automático y puede tener un impacto negativo en la precisión del modelo. Y SAFE cuando es correctamente aplicada es eficiente para evitar esto y mejorar la precisión del modelo al seleccionar solo las características más importantes del conjunto de datos. Se encontraron los casos de sobre ajuste a diferentes niveles. En el primer conjunto de datos, se encontró que el modelo de regresión lineal tenía un menor sobreajuste que el modelo de Random Forest. Sin embargo, ambos modelos empeoraron su rendimiento con el uso de SAFE ML pues el problema está en la programación no en la técnica.