Modelo de evaluación crediticia

Análisis de Giancarlo Marchesi

Este workbook contiene el análisis de un modelo de evaluación crediticia para una pasarela de pagos de pago directo de cuenta. Las transacciones malas corresponden a aquellos clientes que no contaban con fondos en sus cuentas en el momento. La pasarela no tiene conexión en línea con los bancos, por lo que debe basarse en en sus propios modelos de scoring para determinar a qué clientes aprobar. Los modelos se basan en dos scores de proveedores independientes. Este análisis presenta algunas alternativas de modelos de riesgo crediticio utilizando machine learning supervisado y no supervisado.

Explorando la data

In [135…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
```

In [136…
```
df=pd.read_csv('data_final_2.csv', sep=";")
```

In [137…
```
df.head()
```

Out[137…

	score_1	riesgo_score1	score_2	riesgo_score2	interaccion_1	interaccion_2	magnitud_riesgo	r
0	excelente	0	medio	1	0	1	3	
1	excelente	0	excelente	0	0	0	1	
2	excelente	0	medio	1	0	1	3	
3	excelente	0	excelente	0	0	0	1	
4	bueno	0	excelente	0	0	0	2	

In [138…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   score_1           824 non-null    object
 1   riesgo_score1     824 non-null    int64 
 2   score_2           824 non-null    object
 3   riesgo_score2     824 non-null    int64 
 4   interaccion_1     824 non-null    int64 
 5   interaccion_2     824 non-null    int64 
 6   magnitud_riesgo   824 non-null    int64 
 7   monto             824 non-null    int64 
 8   mala_transaccion  824 non-null    int64 
 9   state             823 non-null    object
```

```
 10  poblacion         824 non-null    int64
 11  sc_control        824 non-null    int64
dtypes: int64(9), object(3)
memory usage: 77.4+ KB
```

In [139…
```
df.shape
```

Out[139… (824, 12)

Cada uno de los 824 registros corresponde a una transacción de un cliente individual. Un cliente puede haber realizado varias transacciones.

Descripción de las variables

- `score_1` : calificación del riesgo del cliente según el primer proveedor de score. Tiene 5 categorías: excelente, bueno, medio, malo y no existe data.
- `riesgo_score1` : asignación de riesgo según el score. excelente y bueno tiene un score de 0. Los demás un score de 1.
- `score_2` : calificación del riesgo del cliente según el segundo proveedor de score. Tiene 5 categorías: excelente, bueno, medio, malo y no existe data.
- `riesgo_score2` : asignación de riesgo según el score. excelente y bueno tiene un score de 0. Los demás un score de 1.
- `interacción_1` : un término de interacción entre ambos scores. Si ambos tienen un score de riesgo 1, le corresponde un uno. Si no, le corresponde un 0. Es la versión más laxa de riesgo. Una transacción es solo riesgosa si ambos scores la consideran así.
- `interacción_2` : es una visión más exigente del riesgo. Solo las combinaciones de scores (excelente, excelente), (excelente, bueno), (bueno, excelente) son considerados 0. El resto de combinaciones son considerados 1, inclusive la combinación (bueno, bueno)
- `magnitud_riesgo` : es una variable ordinal construida en base al análisis previo. Se le asigna el valor de 1 al par (excelente, excelente), el valor de 2 al par (bueno, excelente), el valor de 4 al par (malo/nd, todos) y el valor de 3 a las demás combinaciones.
- `monto` : es la magnitud de la transaccion en dólares
- `mala_transaccion` : se le asigna el valor de 1 si es un cheque rebotado y el valor de 0 si la transacción pagó existosamente.
- `poblacion` : es una variable binaria donde 1 corresponde a los cinco estados más poblados, California, Texas, Florida, New York y Pennsylvania
- `sc_control` : es una variable binaria donde 1 corresponde al estado South Carolina, por tener un número desproporcionalmente alto de transacciones, considerando su población total.

In [140…
```
## ¿Cuántas transacciones fueron marcadas como riesgosas por el primer score?
df.riesgo_score1.sum()
```

Out[140… 126

In [141…
```
## ¿Cuántas transacciones fueron marcadas como riesgosas por el segundo score?
df.riesgo_score2.sum()
```

Out[141… 179

In [142...
```
## ¿Cuántas transacciones fueron marcadas como riesgosas por la primera variable de
df.interaccion_1.sum()
```

Out[142... 42

In [143...
```
## ¿Cuántas transacciones fueron marcadas como riesgosas por la segunda variable de
df.interaccion_2.sum()
```

Out[143... 323

In [144...
```
## ¿Cuántas transacciones fueron malas?
df.mala_transaccion.sum()
```

Out[144... 153

Podemos observar que hubo 153 malas transacciones, y que los scores individuales identifican riesgo en un número similar de transacciones (126,179). El primer término de interacción es muy laxo (42) y el segundo es muy restrictivo (323).

Análisis de precisión de las variables

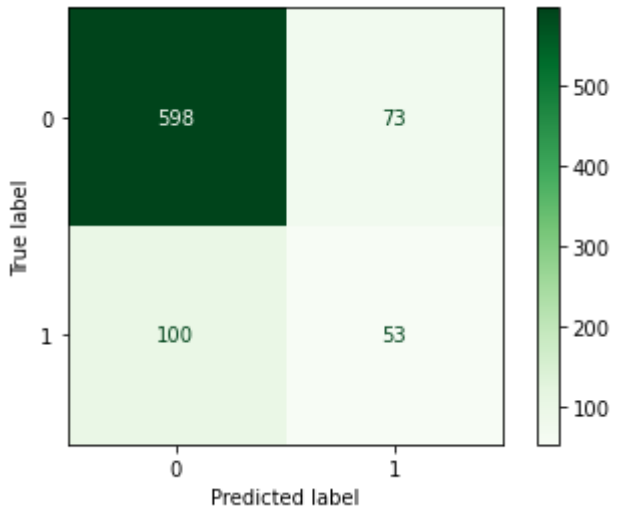
Se puede analizar la precisón de cada uno de los anteriores criterios con una matriz de confusion. Esta determina cuales predicciones fueron correctas (positivos verdaderso y negativos verdaderos), así como los falsos positivos (transacciones que el score predijo que pasarían pero el cliente no tenía fondos) y falsos negativos (transacciones que el score predijo serían negativas pero sí tenían fondos)

Matrix de confusion del score 1 solo

In [145...
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion , df.riesgo_score1)

# Presentando la matrix
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Greens')
plt.show()
```



La matriz se lee de la siguiente manera: el caso positivo 1 es riesgo que no tenga fondos y el negativo 0, indicativo de una transaccion exitosa

- True, Predicted (0,0). Negativos verdaderos. La transaccion pasó y el modelo predijo que pasaría exitosamente
- True, Predicted (1,0). Falsos negativos. La transaccion no pasó, pero el modelo predijo que pasaría exitosamente
- True, Predicted (1,1). Positivos verdadres. La transaccion no pasó y el modelo predijo que no pasaría
- True, Predicted (0,1). Falsos positivos. La transaccion pasó y el modelo predijo que no pasaría

El score 1 presenta 100 falsos negativos y 73 falsos positivos. Si se le hubiera hecho caso 100% al score, hubieran pasado 100 transacciones malas y se hubiese dejado de cobrar por 73 transacciones buenas. La precisión se calcula como la suma de negativos verdaderos y positivos verdaderos dividida por el total de observaciones.

In [146…
```
print("La precision del score 1 es:", ((598+53)/824)*100, "%")
```

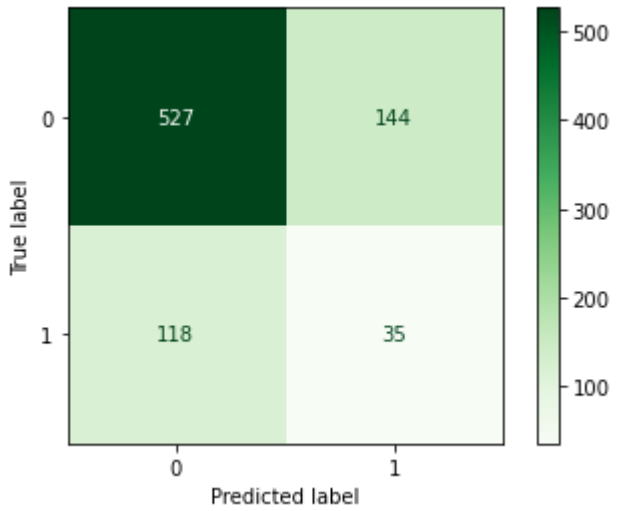
```
La precision del score 1 es: 79.00485436893204 %
```

Matrix de confusion del score 2 solo

In [147…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion , df.riesgo_score2)

# Presentando la matrix
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Greens')
plt.show()
```



In [148…
```
print("La precision del score 2 es:", ((527+35)/824)*100, "%")
```

```
La precision del score 2 es: 68.20388349514563 %
```

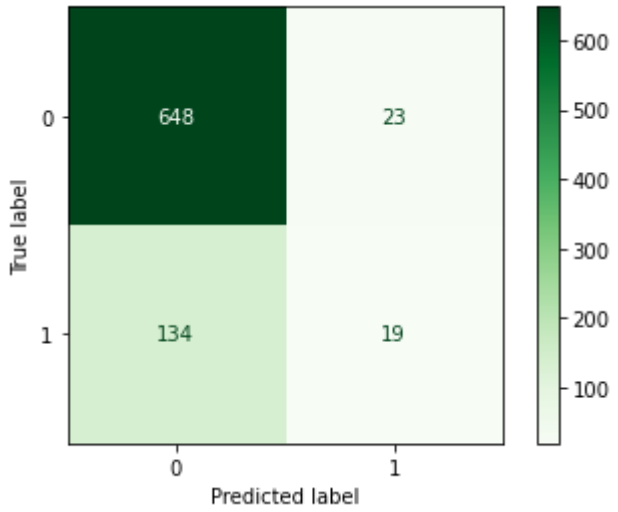
El score 2 presenta 118 falsos negativos y 144 falsos positivos . Si se le hubiera hecho caso 100% al score, hubieran pasado 118 transacciones malas y se hubiese dejado de cobrar por 144 transacciones positivas. La performance del score 2, por si solo, no es buena y no debería usarse como guía de evaluación crediticia.

Matrix de confusion del primer término de interacción (laxo)

In [149…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion , df.interaccion_1)

# Presentando la matrix
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Greens')
plt.show()
```



In [150…
```
print("La precision del termino de interaccion 1 es:", ((649+19)/824)*100, "%")
```

```
La precision del termino de interaccion 1 es: 81.06796116504854 %
```

Una vez que se combinan ambos scores, la precision aumenta 2 puntos porcentuales respecto al score 1 solo. Aun se tiene un número alto de falsos negativos.

In [151…
```
## Centremonos solo sobre los falsos negativos
df_zoom1=df[(df.mala_transaccion==1) & (df.interaccion_1==0)]
```

In [152…
```
pd.crosstab(df_zoom1.score_1, df_zoom1.score_2)
```

Out[152…

score_1 \ score_2	bueno	excelente	malo	medio
bueno	36	12	7	3
excelente	26	10	2	4
malo	6	2	0	0
medio	15	2	0	0
nd	8	1	0	0

Vemos que existen muchos casos de falsos negativos donde el score bueno,bueno corresponde a una mala transaccion. Esto se ha querido corregir con la variable interaccion_2.

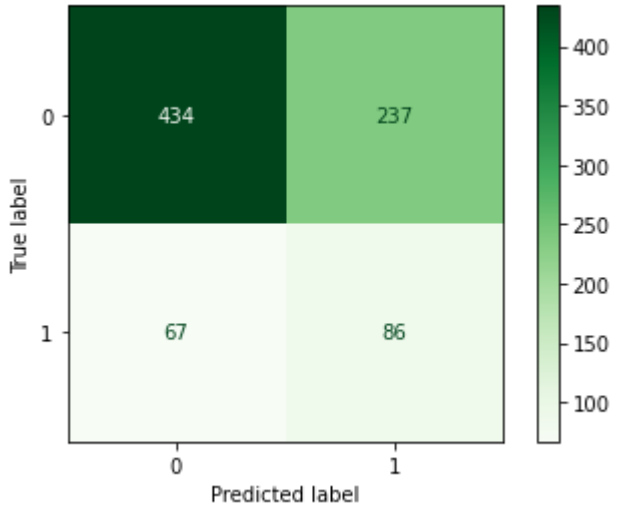
Matrix de confusion del segundo término de interacción (restrictivo)

In [153…
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

# generando la matriz de confusion
```

```
cf_matrix = confusion_matrix(df.mala_transaccion , df.interaccion_2)

# Presentando la matrix
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Greens')
plt.show()
```



In [154…

```
print("La precision del termino de interaccion 2 es:", ((434+86)/824)*100, "%")
```

```
La precision del termino de interaccion 2 es: 63.10679611650486 %
```

El segundo criterio muestra ser muy restrictivo. Termina teniendo muchos falsos positivos, transacciones determinadas como riesgosas pero que sí fueron exitosas. Como conclusión, sin haber utilizado ningún modelo, la línea base de precisión es 81%. Se logra con el término de interaccion 1. Veremos si es posible mejorarlo con modelos de machine learning.

A priori, un modelo de regresión logística nos indicaría la probabilidad que la transacción sea mala pasado cierto umbral. Esa decisión se instalaría en el motor de evaluación para alejar de manera automática las malas transacciones.

Modelos de machine learning

Modelo de regresion logistica

Con este modelo se busca que la interacción de las variables explicativas genere una probabilidad de que la trasacción sea mala.Pasado el umbral del 0.5, se determina que la transacción sería mala. Este modelo podría se alimentado al motor de evaluación en línea para aprobar o declinar las transacciones.

In [155…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   score_1           824 non-null    object
 1   riesgo_score1     824 non-null    int64 
 2   score_2           824 non-null    object
 3   riesgo_score2     824 non-null    int64 
 4   interaccion_1     824 non-null    int64 
 5   interaccion_2     824 non-null    int64 
 6   magnitud_riesgo   824 non-null    int64 
```

```
 7   monto             824 non-null    int64
 8   mala_transaccion  824 non-null    int64
 9   state             823 non-null    object
 10  poblacion         824 non-null    int64
 11  sc_control        824 non-null    int64
dtypes: int64(9), object(3)
memory usage: 77.4+ KB
```

Primera especificación

In [156…

```
# Separamos a las variables dependientes de las independientes
X = df[['riesgo_score1', 'riesgo_score2', 'interaccion_1', 'interaccion_2', 'magnitu
y = df['mala_transaccion']

# Añadiendo el intercepto
X = sm.add_constant(X)
```

In [208…

```
# Instanciando el modelo
logreg1= sm.Logit(y, X)

# Haciendo fit
logreg1_results = logreg1.fit()

# Publicando los resultados
logreg1_results.summary()
```

```
Optimization terminated successfully.
         Current function value: 0.430799
         Iterations 6
```

Out[208…

Logit Regression Results

Dep. Variable:	mala_transaccion	**No. Observations:**	824
Model:	Logit	**Df Residuals:**	815
Method:	MLE	**Df Model:**	8
Date:	Mon, 20 Sep 2021	**Pseudo R-squ.:**	0.1023
Time:	21:07:16	**Log-Likelihood:**	-354.98
converged:	True	**LL-Null:**	-395.44
Covariance Type:	nonrobust	**LLR p-value:**	3.201e-14

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.1019	0.598	-5.188	0.000	-4.274	-1.930
riesgo_score1	1.0127	0.297	3.412	0.001	0.431	1.594
riesgo_score2	-1.0918	0.328	-3.325	0.001	-1.735	-0.448
interaccion_1	0.7925	0.495	1.602	0.109	-0.177	1.762
interaccion_2	0.9833	0.258	3.811	0.000	0.478	1.489
magnitud_riesgo	0.1717	0.163	1.053	0.292	-0.148	0.491
monto	0.0110	0.005	2.033	0.042	0.000	0.022
poblacion	-0.2707	0.198	-1.369	0.171	-0.658	0.117
sc_control	-0.5435	0.416	-1.307	0.191	-1.359	0.272

La variables `magntinud_riesgo` , `interaccion_1` , `población` o `sc_cointrol` no son

estadísticamente significativas, pero si se les elimina se pierde poder explicativo y baja ligeramente la precisión. Por ese motivo, trabajaremos con esta especificación.

In [158…
```
# Obteniendo las probabilidades de prediccion y transformandolas en una variable bin
model_predictions_prob = logreg1_results.predict(X)
model_predictions_binary = np.where(model_predictions_prob>0.5,1,0)
```

In [159…
```
## ¿A cuántas observaciones se les asignó la probabilidad de 1, dado un umbral de 0.
model_predictions_binary.sum()
```

Out[159… 46

In [160…
```
## ¿Cuantas predcciones fueron correctas?
(model_predictions_binary == df['mala_transaccion']).sum()
```

Out[160… 671

In [161…
```
print("La precisión del modelo de regresión logística es:", (671/824)*100, "%")
```

```
La precisión del modelo de regresión logística es: 81.43203883495146 %
```

Es ligeramente superior en décimas al término de interacción 1. Pasaremos a calcular la matriz de confusión.

In [162…
```
## creando una variable de las predicciones binarias del modelo
pred_bin=model_predictions_binary
```

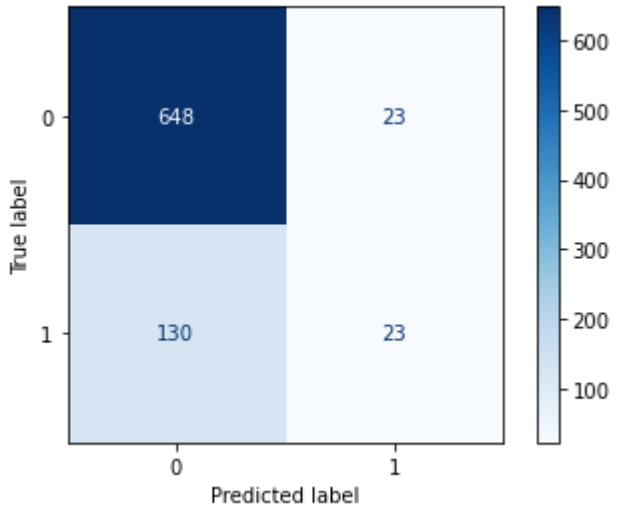
In [163…
```
## insertando esa variable en el dataframe con el nombre pred_bin
df.loc[:,'pred_log']=pred_bin
```

In [164…
```
## comprobando que están ambas variables
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   score_1           824 non-null    object
 1   riesgo_score1     824 non-null    int64 
 2   score_2           824 non-null    object
 3   riesgo_score2     824 non-null    int64 
 4   interaccion_1     824 non-null    int64 
 5   interaccion_2     824 non-null    int64 
 6   magnitud_riesgo   824 non-null    int64 
 7   monto             824 non-null    int64 
 8   mala_transaccion  824 non-null    int64 
 9   state             823 non-null    object
 10  poblacion         824 non-null    int64 
 11  sc_control        824 non-null    int64 
 12  pred_log          824 non-null    int32 
dtypes: int32(1), int64(9), object(3)
memory usage: 80.6+ KB
```

In [165…
```
# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion, df.pred_log)
```

```
# presentando la matriz
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Blues')
plt.show()
```



In [166…
```
## Centremonos solo sobre los falsos negativos
df_zoom2=df[(df.mala_transaccion==1) & (df.pred_log==0)]
```

In [167…
```
df_zoom2.shape
```

Out[167… (130, 13)

In [168…
```
pd.crosstab(df_zoom2.score_1, df_zoom2.score_2)
```

Out[168…

score_1 \ score_2	bueno	excelente	malo	medio
bueno	36	12	7	3
excelente	26	10	2	4
malo	2	0	0	2
medio	15	2	3	3
nd	1	1	1	0

Continua el problema de clientes con buenos scores dando transacciones malas.

- score 1, score 2: (bueno, bueno) 36 transacciones malas
- score 1, score 2: (excelente, bueno) 26 transacciones malas
- score 1, score 2: (bueno, excelente) 12 transacciones malas
- score 1, score 2: (excelente, excelente) 10 transacciones malas

Buscaremos de optimizar el nivel de precisión utilizando otros modelos de machine learning no supervisado como el arbol de decisión y el KNN.

Modelo de arbol de decisión

La principal limitación que tenemos para el uso de modelos de machine learning no supervisado

es el numero de observaciones. No va a ser útil separar la separacion de la data en conjuntos de entrenamiento, validación y testeo. Tampoco va a ser posible hacer optimización de hiperparámetros. Aun con esas limitaciones, aplicaremos los modelos al conjunto completo para darnos una idea aproximada a las mejoras que podrían generar.

In [169…
```
## Importando la librería
from sklearn.tree import DecisionTreeClassifier
```

In [170…
```
## Haciendo fit al modelo con una máxima profundidad de 8 nodos
## Se puede optimizar el número de nodos de manera sistémica. En este caso, por el n
dt_model = DecisionTreeClassifier(max_depth=8)
dt_model.fit(X,y)

# Score on train and test
print(f"La precision del modelo de arbol de decision es {dt_model.score(X, y)}")
```

```
La precision del modelo de arbol de decision es 0.8543689320388349
```

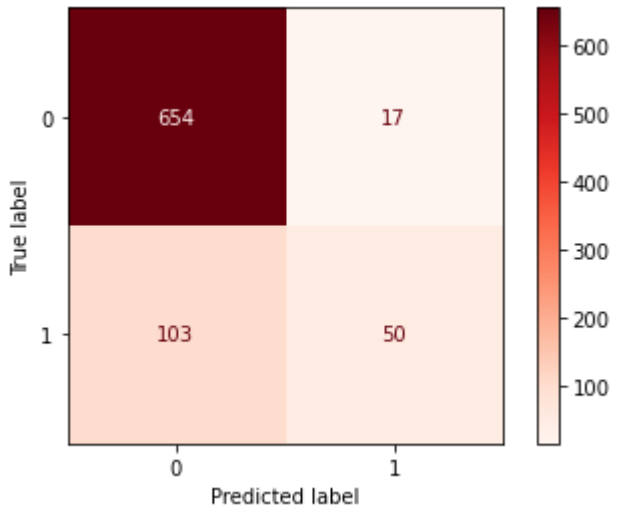
Mejora ligeramente el modelo, hasta 85.43% de precisión.

In [171…
```
# Obteniendo las predicciones del modelo
y_pred_dt = dt_model.predict(X)
```

In [172…
```
df.loc[:,'pred_prob_dt']=y_pred_dt
```

In [173…
```
# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion, df.pred_prob_dt)

# presentando la matriz
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Reds')
plt.show()
```



In [174…
```
## Centremonos solo sobre los falsos negativos
df_zoom3=df[(df.mala_transaccion==1) & (df.pred_prob_dt==0)]
```

In [175…
```
df_zoom3.shape
```

Out[175… (103, 14)

In [209…
```
pd.crosstab(df_zoom3.score_1, df_zoom3.score_2)
```

Out[209…

score_1 \ score_2	bueno	excelente	malo	medio
bueno	23	12	7	2
excelente	25	10	2	4
malo	3	0	0	2
medio	8	0	0	1
nd	0	1	1	2

En comparación al modelo logístico, el modelo de arbol de decisón ha disminuido las malas transacciones que traía la categoría "bueno" del score 2. En el primer caso, eran 80. En este nuevo modelo son 59. En el caso de las malas transacciones que traía la categoría "excelente" del score 2, bajan a 23. Esta mejoras marginales se traducen finalmente en una mayor precisión.

Modelo de KNN - vecinos cercanos

In [177…
```
## Importando la librería
from sklearn.neighbors import KNeighborsClassifier
```

In [178…
```
## Haciendo fit al modelo con un número de 10 vecinos
## Se puede optimizar el número de vecinos de manera sistémica. En este caso, por el
best_knn= KNeighborsClassifier(n_neighbors=10)
best_knn.fit(X, y)
```

Out[178… `KNeighborsClassifier(n_neighbors=10)`

In [179…
```
# Obteniendo las predicciones del modelo
y_pred_knn = best_knn.predict(X)
```

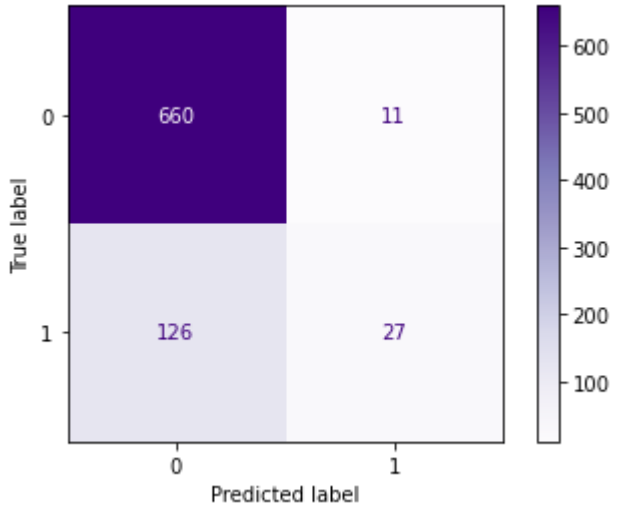
In [180…
```
## incluyendo esta prediccion en el df
df.loc[:,'pred_prob_knn']=y_pred_knn
```

In [181…
```
print (f"La precision del modelo KNN es {best_knn.score(X,y)}")
```

```
La precision del modelo KNN es 0.8337378640776699
```

In [182…
```
# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion, df.pred_prob_knn)

# presentando la matriz
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Purples')
plt.show()
```



La precisión del modelo KNN es superior al modelo logístico pero inferior al modelo de arbol de decisión.

In [183…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   score_1           824 non-null    object
 1   riesgo_score1     824 non-null    int64 
 2   score_2           824 non-null    object
 3   riesgo_score2     824 non-null    int64 
 4   interaccion_1     824 non-null    int64 
 5   interaccion_2     824 non-null    int64 
 6   magnitud_riesgo   824 non-null    int64 
 7   monto             824 non-null    int64 
 8   mala_transaccion  824 non-null    int64 
 9   state             823 non-null    object
 10  poblacion         824 non-null    int64 
 11  sc_control        824 non-null    int64 
 12  pred_log          824 non-null    int32 
 13  pred_prob_dt      824 non-null    int64 
 14  pred_prob_knn     824 non-null    int64 
dtypes: int32(1), int64(11), object(3)
memory usage: 93.5+ KB
```

In [184…
```
## Exportando la data trabajada hasta el momento
df.to_csv('data_final2_prob.csv', index = False)
```

Análisis

El uso de modelos de machine learning mejora en 4.4 puntos porcentuales la precisión de la evaluación del riesgo crediticio cuando se les compara con una regla que combina los scores 1 y 2 (85.43% vs 81.06%). Sin embargo, a pesar de esto, la precisión sigue estando por debajo del óptimo. Veamos esto en terminos de dinero, solo utilizando el mejor modelo, el arbol de decisión.

In [185…
```
## Quedamos solo con los aciertos del modelo
df_zoom4=df[(df.mala_transaccion==0) & (df.pred_prob_dt==0) | (df.mala_transaccion==
```

In [186…

```
## comprobando la forma de la matriz
df_zoom4.shape
```

Out[186… (704, 15)

In [187…
```
df_zoom4.monto.sum()
```

Out[187… 52436

Las transacciones correctamente identificadas suman 52436 dolares.

In [188…
```
## Identificando el valor de las falsos negativos en la matriz calculada en la secci
df_zoom3.shape
```

Out[188… (103, 14)

In [189…
```
df_zoom3.monto.sum()
```

Out[189… 7964

Los falsos negativos cuestan a la empresa 7964 dolares, a un promedio de 77.32 dólares por transacción.

In [190…
```
## Identificando el valor de las falsos positivos
df_zoom5=df[(df.mala_transaccion==0) & (df.pred_prob_dt==1)]
```

In [191…
```
df_zoom5.shape
```

Out[191… (17, 15)

In [192…
```
df_zoom5.monto.sum()
```

Out[192… 1302

Los falsos positivos cuestan a la empresa 1302 dolares, a un promedio de 76.47 dólares por transacción.

In [193…
```
pd.crosstab(df_zoom5.score_1, df_zoom5.score_2)
```

Out[193…

score_1 \ score_2	bueno	excelente	malo	medio
bueno	7	0	0	0
malo	2	0	0	0
medio	2	1	0	1
nd	2	0	2	0

Al igual que en el caso de los falsos negativos, el principal problema del clasificador viene por las transacciones de la categoría "bueno" del score 2. ¿Cuáles podrían ser algunas alternativas

para mejorar este clasificador?

Alternativas para mejorar la precisión del clasificador

Combinar las predicciones de arboles de decision y KNN

El modelo KNN identifica mejor a los negativos (0) y el modelo de arbol de decision identifica mejor a los positivos. La variable `pred_optima` combina los resultados de ambos para dichas clasificaciones. La labor de etiquetado se hizo en Excel. El archivo resultante es data_final_optima.csv.

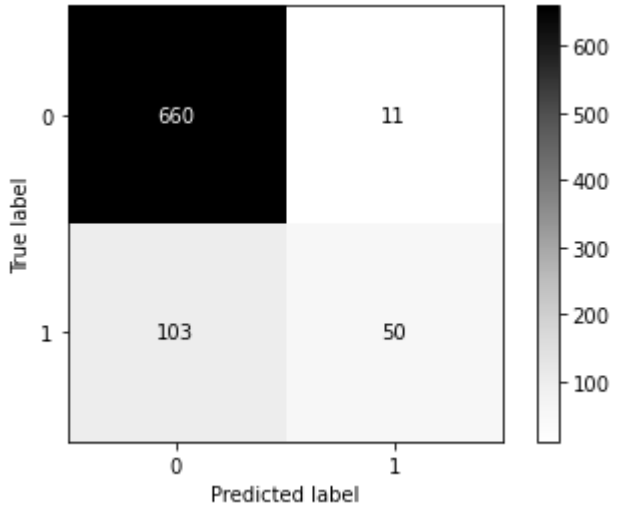
In [194…
```
df=pd.read_csv('data_final_optim.csv', sep=";")
```

In [195…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 824 entries, 0 to 823
Data columns (total 16 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   score_1           824 non-null    object
 1   riesgo_score1     824 non-null    int64 
 2   score_2           824 non-null    object
 3   riesgo_score2     824 non-null    int64 
 4   interaccion_1     824 non-null    int64 
 5   interaccion_2     824 non-null    int64 
 6   magnitud_riesgo   824 non-null    int64 
 7   monto             824 non-null    int64 
 8   mala_transaccion  824 non-null    int64 
 9   state             823 non-null    object
 10  poblacion         824 non-null    int64 
 11  sc_control        824 non-null    int64 
 12  pred_log          824 non-null    int64 
 13  pred_prob_dt      824 non-null    int64 
 14  pred_prob_knn     824 non-null    int64 
 15  pred_optima       824 non-null    int64 
dtypes: int64(13), object(3)
memory usage: 103.1+ KB
```

In [196…
```
# generando la matriz de confusion
cf_matrix = confusion_matrix(df.mala_transaccion, df.pred_optima)

# presentando la matriz
ConfusionMatrixDisplay(cf_matrix).plot(cmap='Greys')
plt.show()
```



In [197…
```
print("La precisión del modelo combinado es:", (710/824)*100, "%")
```

```
La precisión del modelo combinado es: 86.16504854368931 %
```

La precisión ha mejorado 6 décimas, pero subsiten 103 falsos negativos y 11 falsos positivos.

Conseguir una variabe explicativa adicional

Una versión previa del modelo fue trabajada sin las variables `población` y `sc_control`, lográndose una precisión de 84.59% en el modelo de arbol de decisión versus 85.43% en esta versión. Si se incluye alguna otra variable no correlacionada con las anteriores, se podría seguir aumentando la precisión. Por ejemplo, score 2 tiene poca precisión. Se podría pensar en reemplazar esta por otro score, de preferencia uno que tenga una frecuenca de actualización mayor al score 1.

Hacer un chequeo telefónico focalizado sobre las principales áreas problemáticas

In [198…
```
## Identificando el número falsos negativos en el modelo óptimo hasta el momento
df_zoom6=df[(df.mala_transaccion==1) & (df.pred_optima==0)]
```

In [199…
```
df_zoom6.shape
```

Out[199… `(103, 16)`

In [200…
```
pd.crosstab(df_zoom6.score_1, df_zoom6.score_2)
```

Out[200…

score_1 \ score_2	bueno	excelente	malo	medio
bueno	23	12	7	2
excelente	25	10	2	4
malo	3	0	0	2
medio	8	0	0	1
nd	0	1	1	2

Con cada iteración del modelo, se ha seguido mejorando el proceso de filtrado de las malas

transacciones. Sin embargo, las combinaciones de bueno y excelente de ambos scores siguen dando un número alto de transacciones fallidas. ¿Cuánto podría mejorar con un filtrado de llamadas? ¿Caunto es el conjunto universal sobre el que se debería hacer el filtrado?

In [201…
```
## El conjunto de malas transacciones con calificación de buena o excelente por algu
23+25+12+10
```

Out[201… 70

In [202…
```
print("Esto es el ", (70/103)*100, "% del total de malas transacciones")
```

```
Esto es el  67.96116504854369 % del total de malas transacciones
```

In [210…
```
## cuantas transacciones del total corresponde a estas 4 combinaciones de score?
pd.crosstab(df.score_1, df.score_2)
```

Out[210…

score_1 \ score_2	bueno	excelente	malo	medio	nodata
bueno	106	77	19	20	1
excelente	227	147	24	74	3
malo	17	2	1	4	0
medio	31	17	7	6	0
nd	14	2	9	15	1

In [204…
```
## El conjunto universal de transacciones por filtar es
106+227+77+147
```

Out[204… 557

In [205…
```
print("Esto es el ", (557/824)*100, "% del total")
```

```
Esto es el  67.59708737864078 % del total
```

Las malas transacciones son proporcionales al número de transacciones totales correspondientes a los categorias 'bueno' o 'excelente' de ambos scores.

Si se hace un muestreo de llamadas:

- 25% de 557 transacciones: 139 llamadas **Con el 100% de respuesta: 25% de 103. 26 transacciones nuevas negadas (negativo, negativo).**

In [206…
```
print("La precisión del modelo total sería:", ((710+26)/824)*100, "%")
```

```
La precisión del modelo total sería: 89.32038834951457 %
```

- 33% de 557 transacciones: 184 llamadas **Con el 100% de respuesta: 33% de 103. 34 transacciones nuevas negadas.

In [207…
```
print("La precisión del modelo total sería:", ((710+36)/824)*100, "%")
```

```
La precisión del modelo total sería: 90.53398058252428 %
```

La efectividad final del modelo dependería de si se logra el 100% de respuesta de los bancos. Si esto no es posible, bajará respecto al estimado pues por defecto el motor debería aprobar la operación.

Conclusiones

Este análisis muestra cómo mejorar la precisón de un sistema de análisis de riesgo tomando como base un score de un proveedor independiente, el score 1. La máxima precisión se logra con una combinacion de modelos de árboles de decisión de 8 nodos y un modelo de 10 vecinos cercanos. Entre las alternativas para mejorar este modelo de clasificación están:

- Obtener más data de transacciones para entrenar y validar los modelos.
- Obtener otras variables explicativas que no estén correlacionadas con los scores. Por ejemplo, las variables poblacionales añadieron 1 punto porcentual de precisión.
- Complementar con un proceso manual solo especificamente sobre los pares bueno y excelente de ambos scores. El inconveniente de esta alternativa es su escalabilidad. Suponiendo que la validacion tiene un ratio de respuesta del 100% (supuesto optimista), efectivamente un muestreo de 33% de transacciones en este subgrupo puede mejorar la precision del sistema en 4 puntos, posicionándolo en 90.5%, en el rango de muy bueno para los estándares de modelos de clasificación. En caso dicho supuesto no se cumpla, se seguirán filtrando falsos negativos, pues la respuesta por defecto para este subconjunto es dejar pasar la transacción.

In [ ]: