

Trabajo práctico Inteligencia Artificial FIUBA

Pedro Grin 108937
Lucas Nahuel Vilas 108650
Lisandro Bachur 108572

Profesor Hernán Daniel Merlino

Roadmap de la presentación:

- Objetivo del trabajo
- Presentación de los datos y del problema
- Preprocesamiento de datos
- Entrenamiento de modelos
- Conclusión del trabajo

— — —

Objetivo del trabajo

Objetivo del trabajo

Queríamos aplicar lo visto en clase a un problema de nuestro interés



Problema de interés social

Problema sobre un área ajena a la informática

Ver como la informática, y particularmente la Inteligencia Artificial, puede ser fácilmente aplicable a diversas áreas para resolver problemas

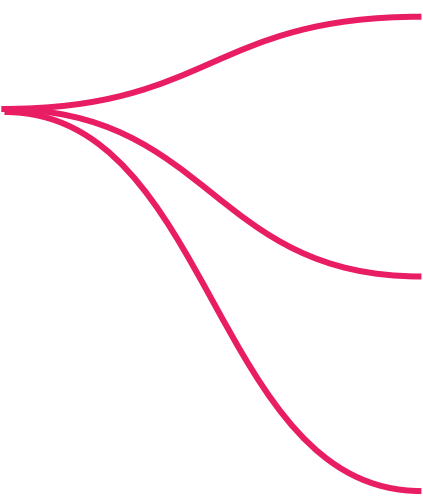
Presentación de los datos y del problema

Presentación de los datos y del problema

— — —

Problema:

En base a datos
“fácilmente”
medibles por una
persona poder
decir si tiene o
no posibilidades
de sufrir un
problema cardíaco



Mandar al
paciente a
hacerse un
estudio
específico

Recomendar
cambiar algún
hábito en
específico

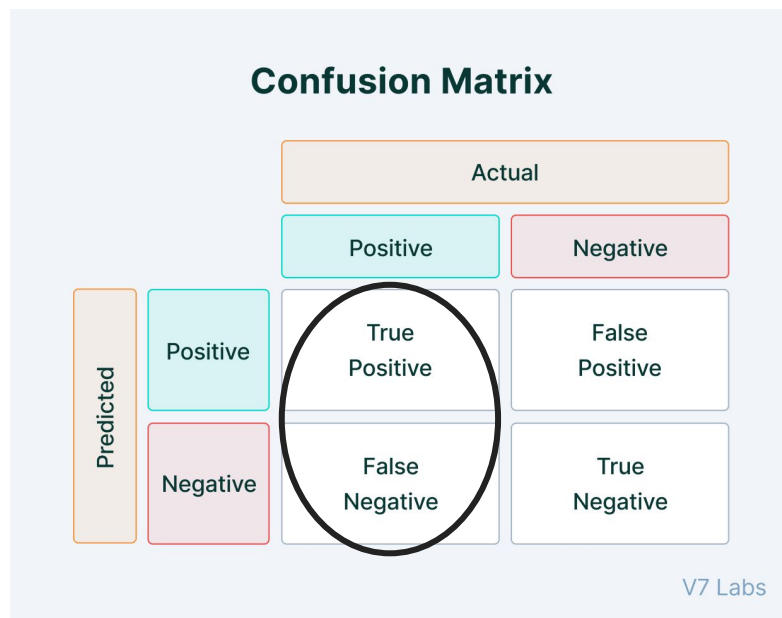
Poder centrarse o
priorizar un
hábito sobre otro

Presentación de los datos y del problema

La métrica a optimizar, y en la que más hincapié haremos será el recall

$$\frac{TP}{TP + FN}$$

TP: True positives
FN: False negatives



Presentación de los datos y del problema

Elegimos esta métrica porque es mucho más importante reducir los falsos negativos ya que un paciente podría tener un ataque al corazón y no llegar a prevenirlo

En cambio, si a un paciente lo envíamos a hacerse los estudios y finalmente no era nada (falso positivo) lo único que pierde es tiempo en hacerse los estudios, algo que tampoco es malo ya que deberíamos chequearnos continuamente

Presentación de los datos y del problema

— — —

Smoking (binario)
AlcoholDrinking (binario)
Stroke (binario)
DiffWalking (binario)
Sex (binario)
AgeCategory (13 categorías)
Race (6 categorías)
Diabetic (binario)
PhysicalActivity (binario)
GenHealth (5 categorías)
Asthma (binario)
KidneyDisease (binario)
SkinCancer (binario)

Features categóricos

BMI (valores de 12.020 a 94.850)
PhysicalHealth (valores de 0 a 30)
MentalHealth (valores de 0 a 30)
SleepTime (valores de 1 a 24)

Features numéricos

Presentación de los datos y del problema

17 columnas entre categóricas y numéricas que buscan predecir **HeartDisease**, una variable categórica y binaria

1: HeartDisease

0: Not HeartDisease

Problema de clasificación binario

Preprocesamiento de datos

Preprocesamiento de datos

Pantallazo general:

- Gráficos
- Encodings
- Ahorro en memoria
- Undersampling
- Reducción de dimensiones

Preprocesamiento de datos

Gráficos:

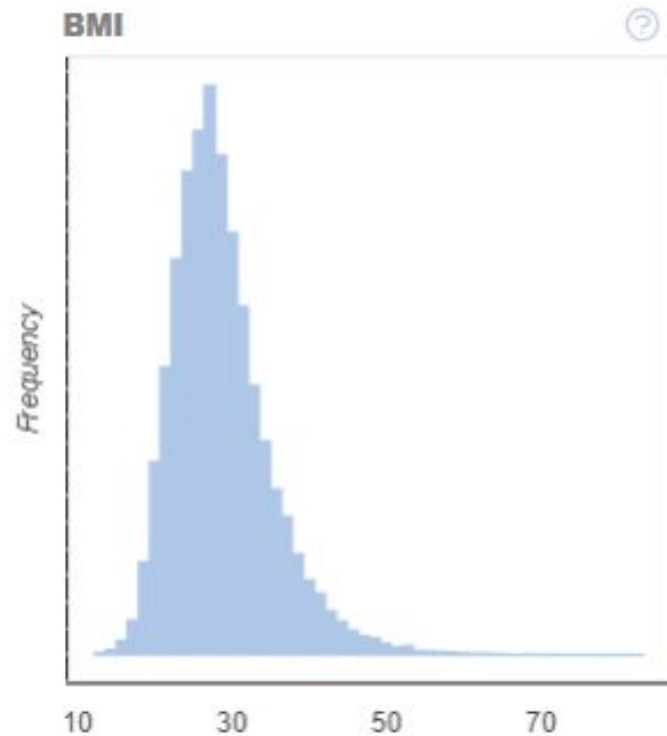
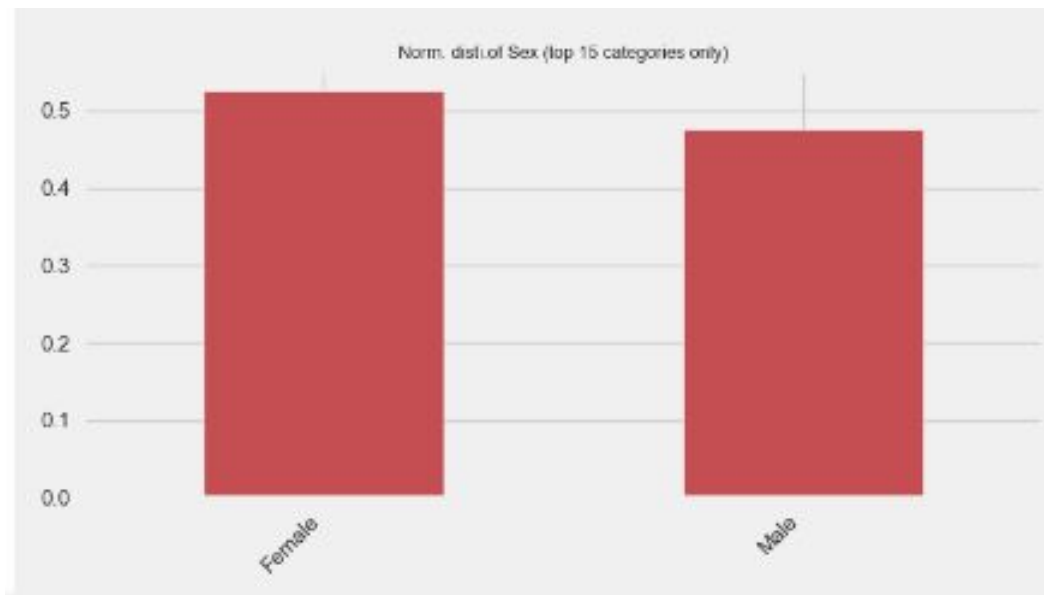
Nos sirven para visualizar los datos y así poder entenderlos mejor, ya sean sus distribuciones o el aporte que estos realizan.

Las librerías que utilizamos son Dataprep, AutoViz, Sweetviz, Dtale.

Todas estas librerías son para realizar gráficos automáticos, lo que nos ahorra tiempo de trabajo ya sea de pensar que graficar como también de la forma en realizarlos

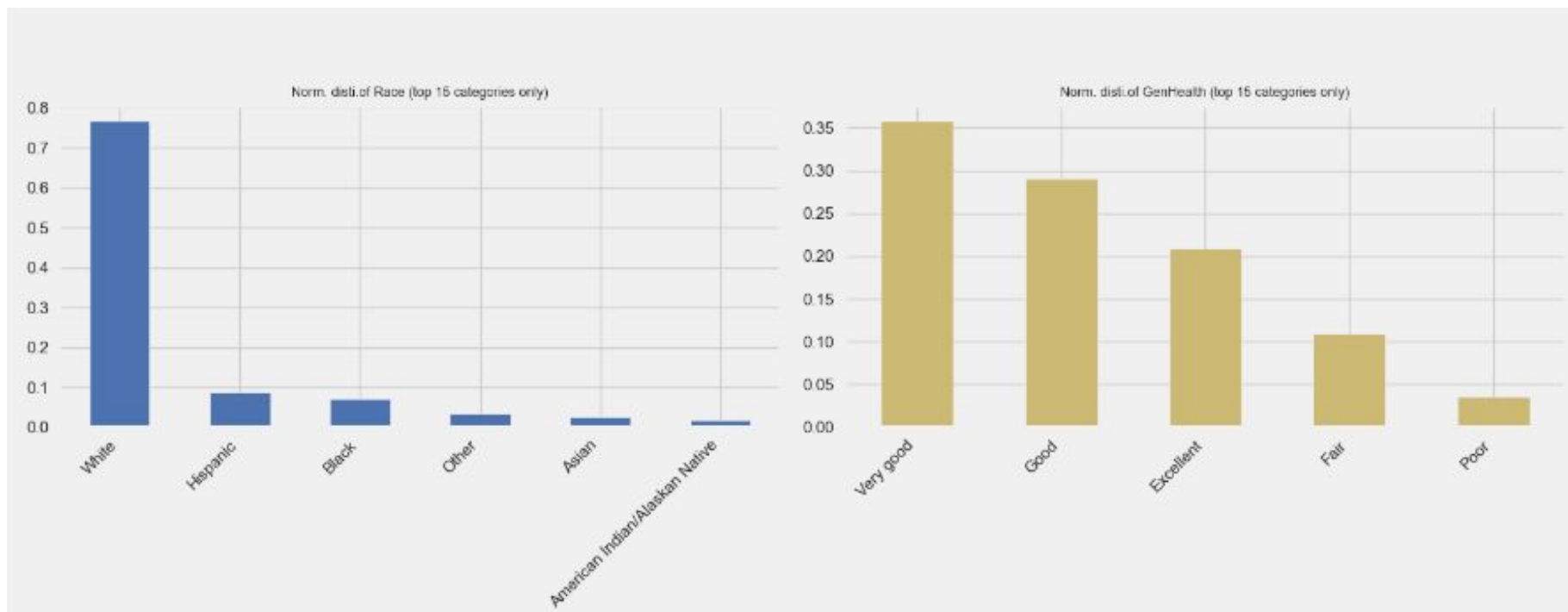
Preprocesamiento de datos

Algunos gráficos interesantes



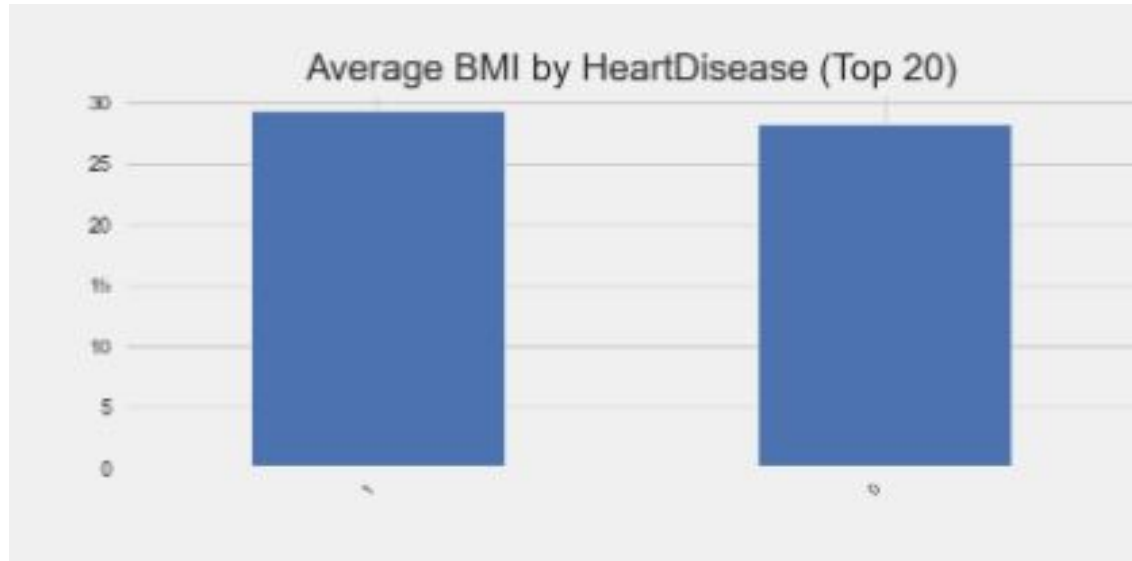
Preprocesamiento de datos

— — —
Algunos gráficos interesantes



Preprocesamiento de datos

Algunos gráficos interesantes



Preprocesamiento de datos

Encodings:

- **Preparación de los datos:** En primer lugar, se recopila y se prepara el conjunto de datos de enfermedades del corazón. Esto puede incluir la limpieza de datos, el manejo de valores faltantes
- **Codificación de variables categóricas:** Si el conjunto de datos contiene variables categóricas, como el género o el tipo de enfermedad, es necesario codificarlas en una representación numérica adecuada para su procesamiento.

Preprocesamiento de datos

Encodings:

- **Normalización de variables numéricas:** Si hay variables numéricas, es común aplicar técnicas de normalización para escalar los valores y asegurar que todas las variables tengan un rango similar.
- **División de los datos:** Se divide el conjunto de datos en conjuntos de entrenamiento y prueba. El conjunto de entrenamiento se utiliza para entrenar el modelo, mientras que el conjunto de prueba se utiliza para evaluar su desempeño posteriormente.
- **Aplicación de modelos de aprendizaje automático:** Una vez que los datos se han codificado y dividido, se pueden aplicar diferentes algoritmos de aprendizaje automático, como árboles de decisión, regresión logística o redes neuronales, para construir un modelo que pueda predecir enfermedades del corazón o realizar otro tipo de análisis.

Preprocesamiento de datos

— — —

Ahorro en memoria:

Como ahora conocemos los datos que contiene cada columna, podemos transformar las columnas en distintos tipos de datos acorde a la información que almacena la columna

INICIAL VS FINAL

0	HeartDisease	319795	non-null	object
1	BMI	319795	non-null	float64
2	Smoking	319795	non-null	object
3	AlcoholDrinking	319795	non-null	object
4	Stroke	319795	non-null	object
5	PhysicalHealth	319795	non-null	float64
6	MentalHealth	319795	non-null	float64
7	DiffWalking	319795	non-null	object
8	Sex	319795	non-null	object
9	AgeCategory	319795	non-null	object
10	Race	319795	non-null	object
11	Diabetic	319795	non-null	object
12	PhysicalActivity	319795	non-null	object
13	GenHealth	319795	non-null	object
14	SleepTime	319795	non-null	float64
15	Asthma	319795	non-null	object
16	KidneyDisease	319795	non-null	object
17	SkinCancer	319795	non-null	object

dtypes: float64(4), object(14)

memory usage: 43.9+ MB

0	HeartDisease	319764	non-null	int8
1	BMI	319764	non-null	float64
2	Smoking	319764	non-null	int8
3	AlcoholDrinking	319764	non-null	int8
4	Stroke	319764	non-null	int8
5	PhysicalHealth	319764	non-null	int8
6	MentalHealth	319764	non-null	int8
7	DiffWalking	319764	non-null	int8
8	Sex	319764	non-null	category
9	AgeCategory	319764	non-null	category
10	Race	319764	non-null	category
11	Diabetic	319764	non-null	int8
12	PhysicalActivity	319764	non-null	int8
13	GenHealth	319764	non-null	category
14	SleepTime	319764	non-null	int8
15	Asthma	319764	non-null	int8
16	KidneyDisease	319764	non-null	int8
17	SkinCancer	319764	non-null	int8

dtypes: category(4), float64(1), int8(13)

memory usage: 7.6 MB

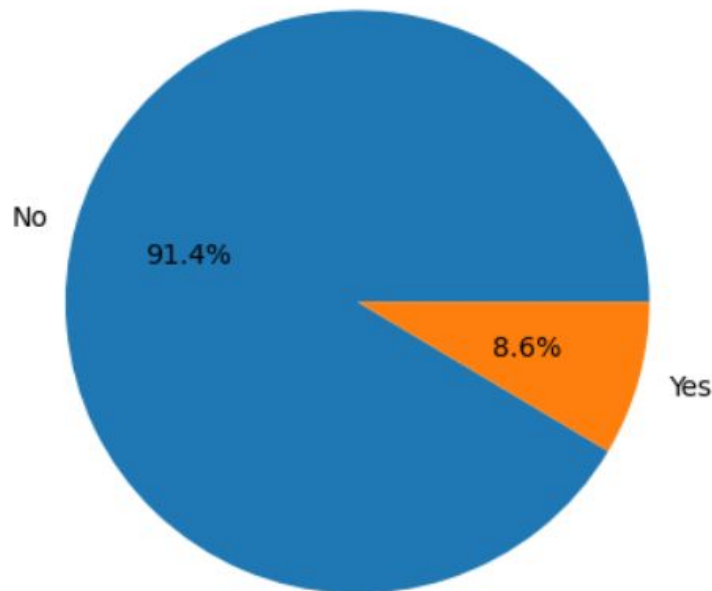
Preprocesamiento de datos

Undersampling:

Técnica utilizada para equilibrar las clases, ya que nuestro modelo tenía muchos más datos sobre casos donde no había un problema de corazón frente a los casos en donde había un problema de corazón

Preprocesamiento de datos

Undersampling:



Muy desequilibrados

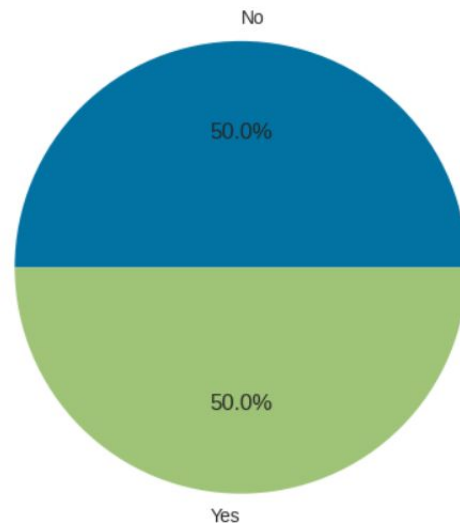
Preprocesamiento de datos

Undersampling:

```
len(df_clase_0), len(df_clase_1)
```

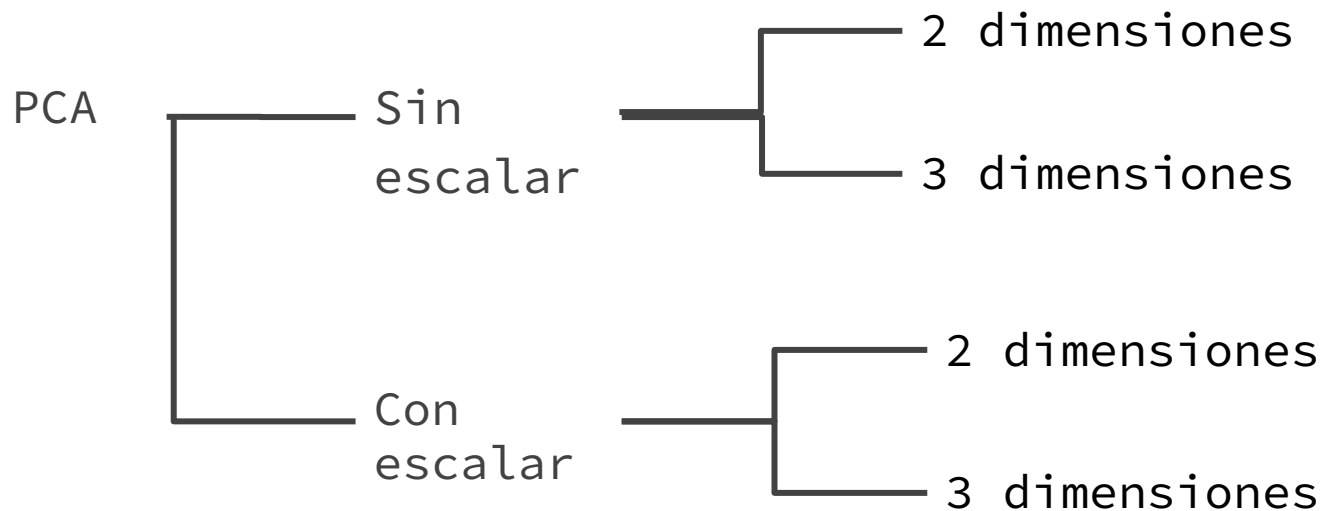
(292397, 27367)

```
df_clase_0 = df_clase_0[0:27367]
```

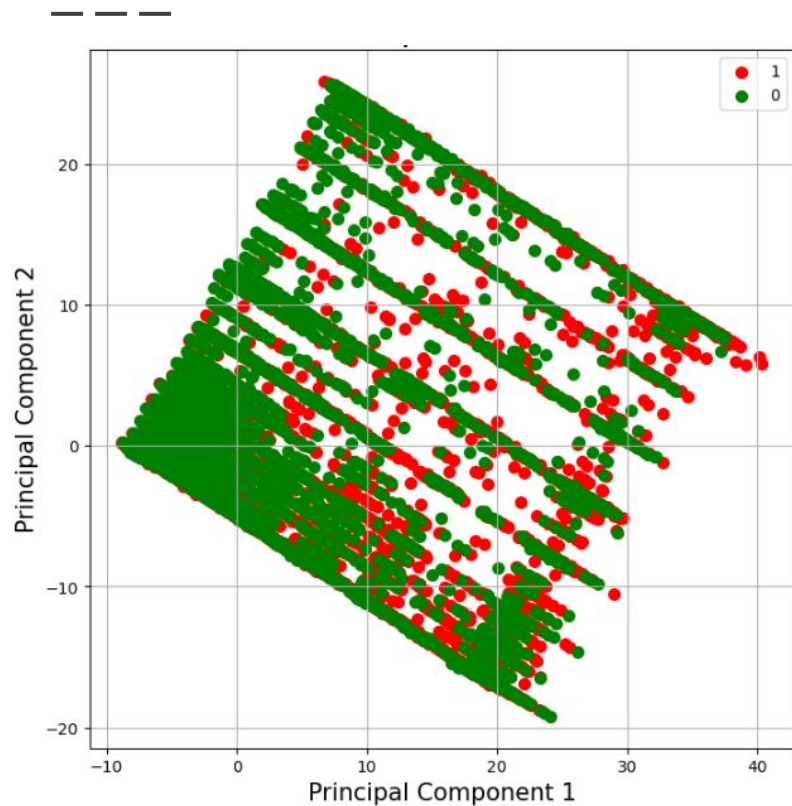


Preprocesamiento de datos

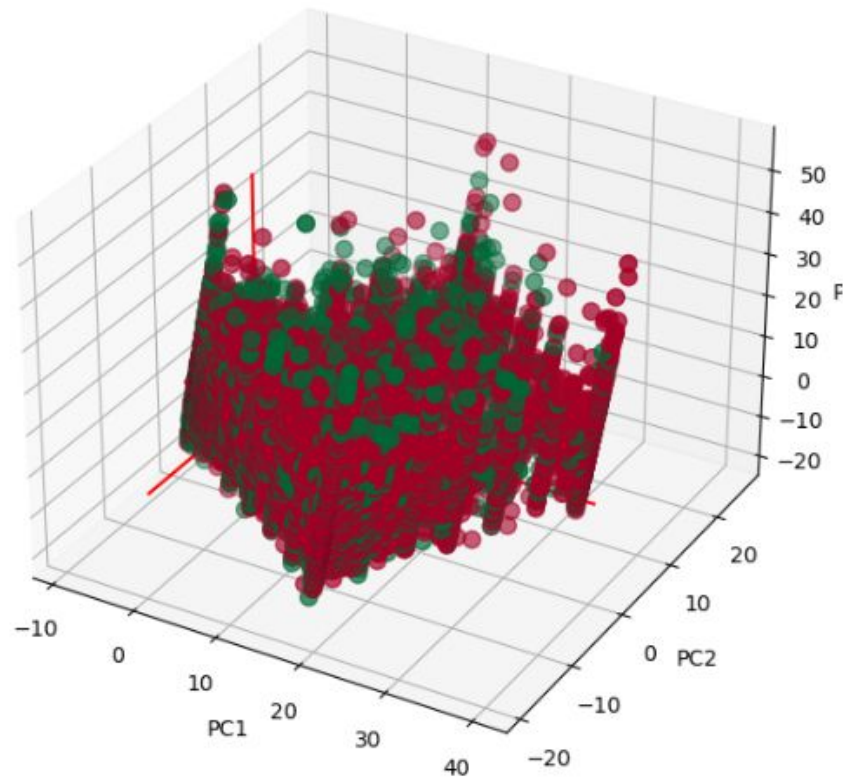
Reducción de dimensiones:



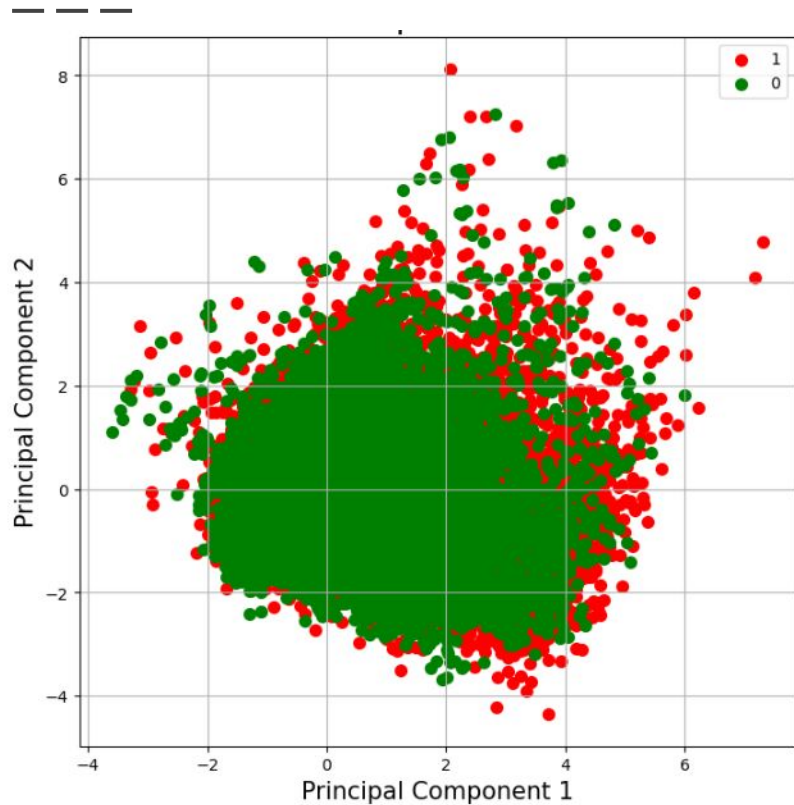
Preprocesamiento de datos



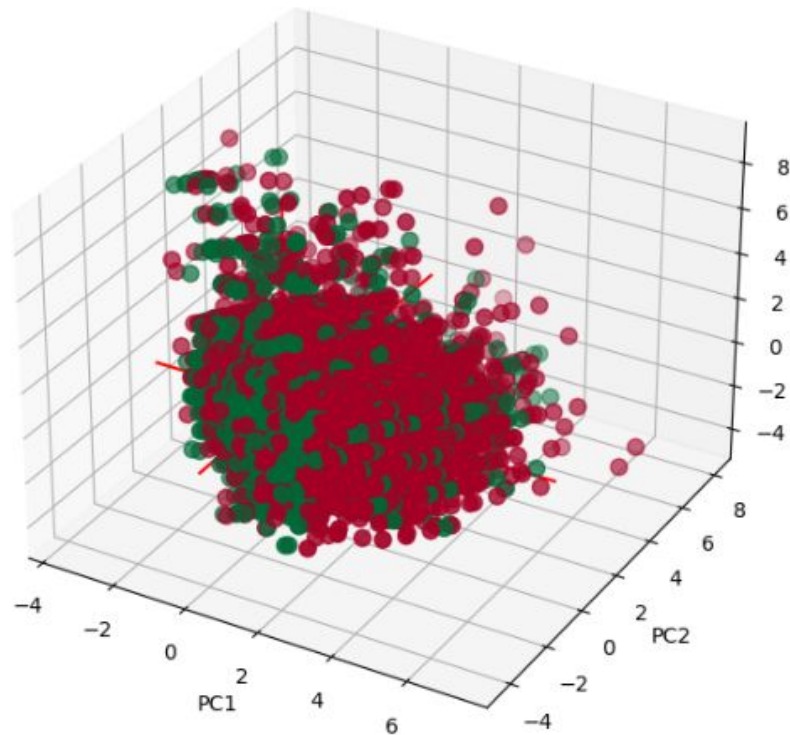
PCA sin escalar



Preprocesamiento de datos



PCA con escalar



Preprocesamiento de datos

— — —

Gráficos interactivos de PCA:

<https://colab.research.google.com/drive/1rSfxCYhfLoH4zMpxIq-Megpz0LF9toKe?usp=sharing>

Preprocesamiento de datos

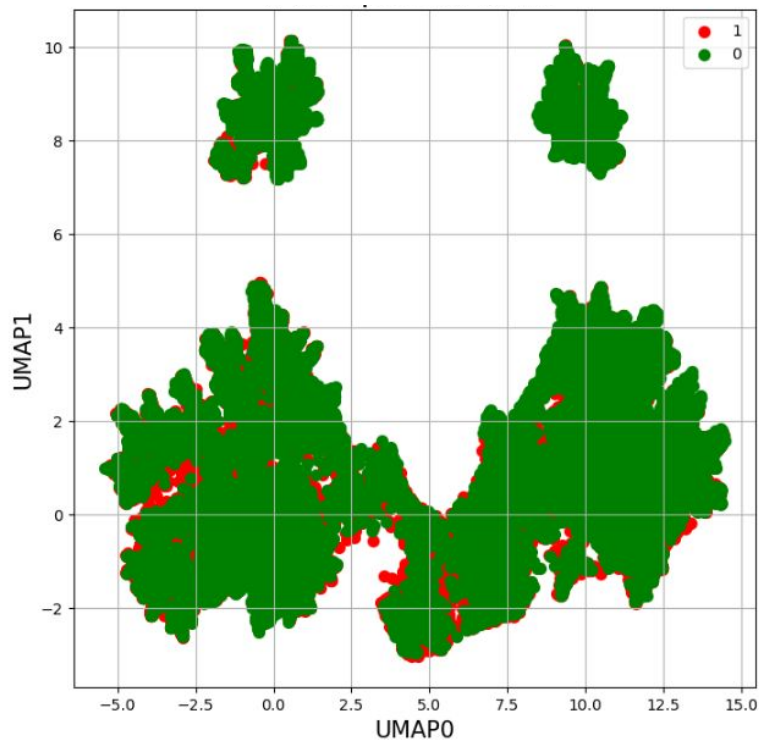
Reducción de dimensiones:

UMAP: Uniform Manifold Approximation and Projection for
Dimension Reduction

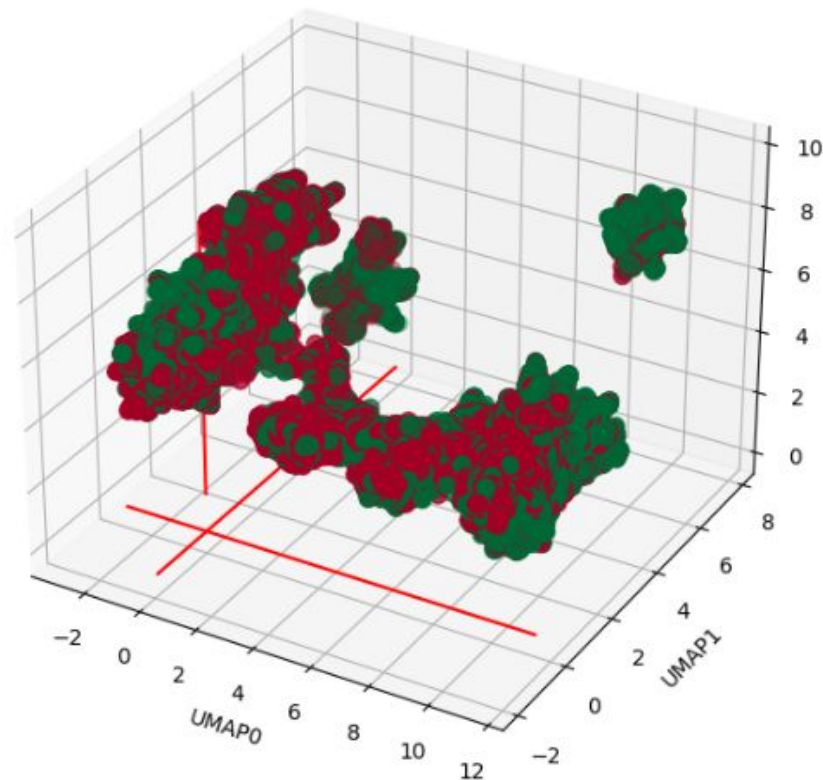
Sirve tanto para reducción de dimensiones de datos lineales
como no lineales

Documentación: <https://umap-learn.readthedocs.io/en/latest/>

Preprocesamiento de datos



UMAP con escalar



Preprocesamiento de datos

Como nuestros datos no parecen ser lineales ya que no tienen mucha correlación entre ellos (ver matriz correlación en la próxima filmína)

Decidimos quedarnos con la reducción hecha por UMAP con 3 componentes, ya que sirve para datos no lineales

Agregamos esos 3 componentes como features al dataset para ayudar al entrenamiento de nuestros modelos

Preprocesamiento de datos

— — —

Matriz de correlación:

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Diabetic	PhysicalActivity	SleepTime	Asthma	KidneyDisease	SkinCancer
HeartDisease	1.000	0.052	0.108	-0.032	0.197	0.171	0.029	0.201	0.175	-0.100	0.008	4.141e-02	0.145	9.336e-02
BMI	0.052	1.000	0.023	-0.039	0.020	0.110	0.064	0.182	0.200	-0.151	-0.053	9.233e-02	0.051	-3.361e-02
Smoking	0.108	0.023	1.000	0.112	0.061	0.115	0.085	0.120	0.056	-0.097	-0.030	2.415e-02	0.035	3.401e-02
AlcoholDrinking	-0.032	-0.039	0.112	1.000	-0.020	-0.017	0.051	-0.035	-0.058	0.018	-0.005	-2.186e-03	-0.028	-5.688e-03
Stroke	0.197	0.020	0.061	-0.020	1.000	0.137	0.046	0.174	0.105	-0.079	0.012	3.889e-02	0.091	4.808e-02
PhysicalHealth	0.171	0.110	0.115	-0.017	0.137	1.000	0.288	0.428	0.154	-0.232	-0.063	1.180e-01	0.142	4.172e-02
MentalHealth	0.029	0.064	0.085	0.051	0.046	0.288	1.000	0.152	0.030	-0.096	-0.122	1.140e-01	0.037	-3.339e-02
DiffWalking	0.201	0.182	0.120	-0.035	0.174	0.428	0.152	1.000	0.209	-0.278	-0.023	1.032e-01	0.153	6.484e-02
Diabetic	0.175	0.200	0.056	-0.058	0.105	0.154	0.030	0.209	1.000	-0.137	0.003	4.692e-02	0.149	3.408e-02
PhysicalActivity	-0.100	-0.151	-0.097	0.018	-0.079	-0.232	-0.096	-0.278	-0.137	1.000	0.005	-4.154e-02	-0.082	-1.319e-03
SleepTime	0.008	-0.053	-0.030	-0.005	0.012	-0.063	-0.122	-0.023	0.003	0.005	1.000	-4.867e-02	0.006	4.139e-02
Asthma	0.041	0.092	0.024	-0.002	0.039	0.118	0.114	0.103	0.047	-0.042	-0.049	1.000e+00	0.040	-3.771e-04
KidneyDisease	0.145	0.051	0.035	-0.028	0.091	0.142	0.037	0.153	0.149	-0.082	0.006	3.972e-02	1.000	6.177e-02
SkinCancer	0.093	-0.034	0.034	-0.006	0.048	0.042	-0.033	0.065	0.034	-0.001	0.041	-3.771e-04	0.062	1.000e+00

Entrenamiento de modelos

Entrenamiento de modelos

Pantallazo general:

- Baseline
- Modelo LGBM
- Modelo XGBoost
- Modelo RandomForest
- Ensemble de modelos
- Red Neuronal

Entrenamiento de modelos

Baseline:

“A simple model that acts as a reference in a machine learning project. Its main function is to contextualize the results of trained models. Baseline models usually lack complexity and may have little predictive power”

Entrenamiento de modelos

— — —

Utilizamos la librería de Pycaret ya que nos permite comparar distintos modelos rápidamente

Baseline:

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.7581	0.8322	0.7954	0.7402	0.7668	0.5161	0.5176	0.8840
gbc	Gradient Boosting Classifier	0.7505	0.8248	0.7614	0.7451	0.7532	0.5009	0.5011	1.9020
xgboost	Extreme Gradient Boosting	0.7501	0.8254	0.7804	0.7358	0.7574	0.5002	0.5012	1.3850
rf	Random Forest Classifier	0.7413	0.8112	0.7725	0.7272	0.7492	0.4827	0.4836	1.4070
ada	Ada Boost Classifier	0.7385	0.8089	0.7192	0.7481	0.7333	0.4770	0.4774	0.5620
lr	Logistic Regression	0.7309	0.8020	0.7122	0.7399	0.7258	0.4618	0.4621	1.8240
ridge	Ridge Classifier	0.7291	0.0000	0.7105	0.7380	0.7240	0.4582	0.4585	0.2730
lda	Linear Discriminant Analysis	0.7290	0.8005	0.7103	0.7379	0.7238	0.4580	0.4583	0.2840
et	Extra Trees Classifier	0.7288	0.7961	0.7511	0.7190	0.7347	0.4575	0.4580	1.2780
svm	SVM - Linear Kernel	0.7172	0.0000	0.6721	0.7418	0.7026	0.4345	0.4389	0.2690
knn	K Neighbors Classifier	0.7162	0.7758	0.7171	0.7158	0.7164	0.4323	0.4323	0.3940
nb	Naive Bayes	0.7043	0.7799	0.6183	0.7467	0.6764	0.4085	0.4148	0.2130
dt	Decision Tree Classifier	0.6642	0.6642	0.6608	0.6653	0.6631	0.3284	0.3285	0.2690
qda	Quadratic Discriminant Analysis	0.5446	0.5607	0.6315	0.5977	0.4755	0.0891	0.0875	0.2840
dummy	Dummy Classifier	0.5000	0.5000	0.1000	0.0500	0.0667	0.0000	0.0000	0.2600

Entrenamiento de modelos

Baseline:

Elegimos el modelo de Naive Bayes porque, si bien no es el que mejor métrica nos da, es el más rápido y en esta etapa buscamos eso: simplicidad y velocidad

Entrenamiento de modelos

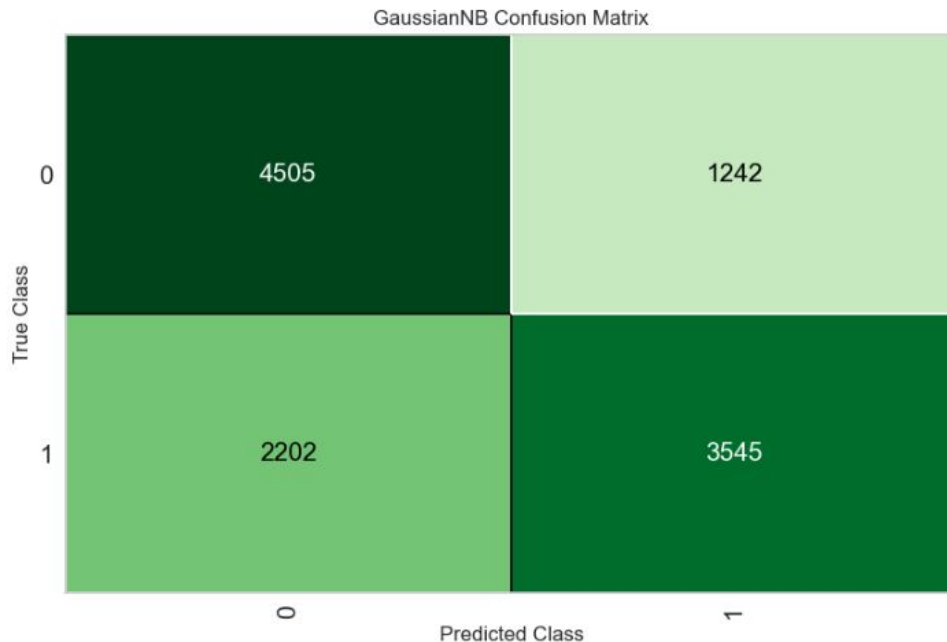
— — —

Vemos que las métricas no son muy buenas

Cuando es 1 predice más veces 0 que cuando es 0 que predice menos veces 1

Y nosotros preferimos equivocarnos al revés, mandando a alguien a hacer estudios pero que se nos pasen los menos posibles que puedan tener un problema del corazón

Baseline:



Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
Naive Bayes	0.7004	0.7825	0.6168	0.7405	0.6731	0.4007	0.4064

Entrenamiento de modelos

Selección de
modelos:

Basados en este
cuadro elegimos
los modelos con
mejor Recall

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lightgbm	Light Gradient Boosting Machine	0.7581	0.8322	0.7954	0.7402	0.7668	0.5161	0.5176	0.8840
gbc	Gradient Boosting Classifier	0.7505	0.8248	0.7614	0.7451	0.7532	0.5009	0.5011	1.9020
xgboost	Extreme Gradient Boosting	0.7501	0.8254	0.7804	0.7358	0.7574	0.5002	0.5012	1.3850
rf	Random Forest Classifier	0.7413	0.8112	0.7725	0.7272	0.7492	0.4827	0.4836	1.4070
ada	Ada Boost Classifier	0.7385	0.8089	0.7192	0.7481	0.7333	0.4770	0.4774	0.5620
lr	Logistic Regression	0.7309	0.8020	0.7122	0.7399	0.7258	0.4618	0.4621	1.8240
ridge	Ridge Classifier	0.7291	0.0000	0.7105	0.7380	0.7240	0.4582	0.4585	0.2730
lda	Linear Discriminant Analysis	0.7290	0.8005	0.7103	0.7379	0.7238	0.4580	0.4583	0.2840
et	Extra Trees Classifier	0.7288	0.7961	0.7511	0.7190	0.7347	0.4575	0.4580	1.2780
svm	SVM - Linear Kernel	0.7172	0.0000	0.6721	0.7418	0.7026	0.4345	0.4389	0.2690
knn	K Neighbors Classifier	0.7162	0.7758	0.7171	0.7158	0.7164	0.4323	0.4323	0.3940
nb	Naive Bayes	0.7043	0.7799	0.6183	0.7467	0.6764	0.4085	0.4148	0.2130
dt	Decision Tree Classifier	0.6642	0.6642	0.6608	0.6653	0.6631	0.3284	0.3285	0.2690
qda	Quadratic Discriminant Analysis	0.5446	0.5607	0.6315	0.5977	0.4755	0.0891	0.0875	0.2840
dummy	Dummy Classifier	0.5000	0.5000	0.1000	0.0500	0.0667	0.0000	0.0000	0.2600

Entrenamiento de modelos

Modelo de **LightGBM**:

“LightGBM is a gradient boosting framework that uses tree based learning algorithms”

Boosting consiste básicamente en:

1. Entrenar algoritmo simple y analizar sus resultados
2. Entrenar otro algoritmo simple en donde se le de mayor peso a los resultados para los cuales el anterior tuvo peor performance
3. Resultado final en base a un promedio ponderado

Entrenamiento de modelos

— — —

Modelo de LightGBM:

Búsqueda de hiperparametros:

```
lgbm = LGBMClassifier(random_state=10)
```

```
params = [  
    {'n_estimators': [80, 100, 120]},  
    {'max_depth': [5, 10, 15]},  
    {"random_state": [10]},  
    {"learning_rate": [0.01, 0.1, 0.3]}  
]
```

```
gs = GridSearchCV(lgbm, param_grid = params, cv=5, scoring='recall')
```

Hiperparametros más importantes

Usamos cross-validation

Optimizamos el recall

Entrenamiento de modelos

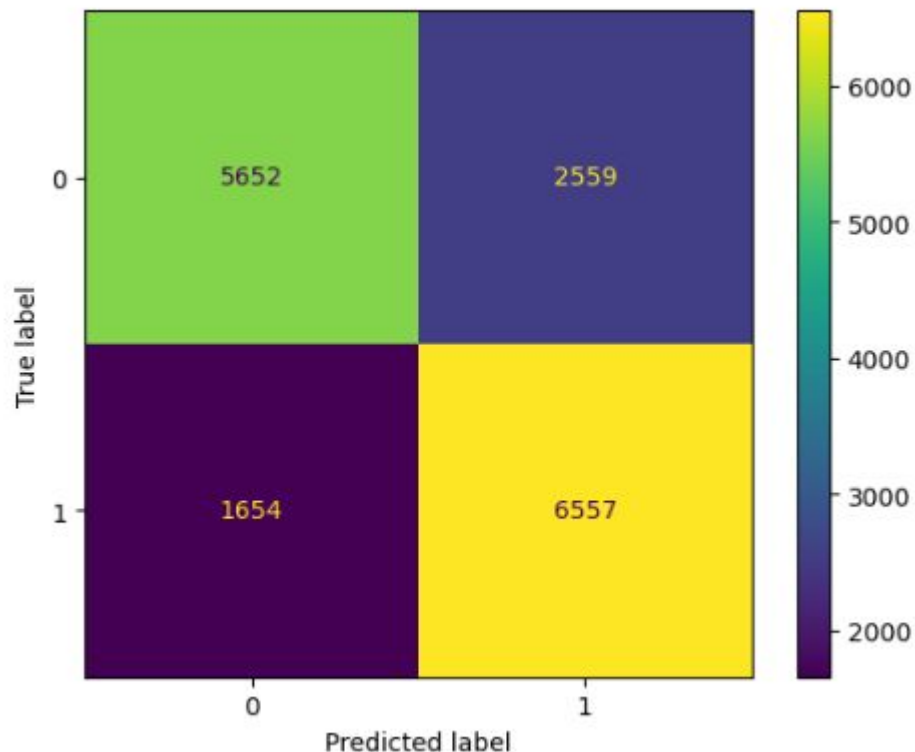
Modelo de LightGBM:

```
recall_score(y_test, preds)
```

0.7985629034222385

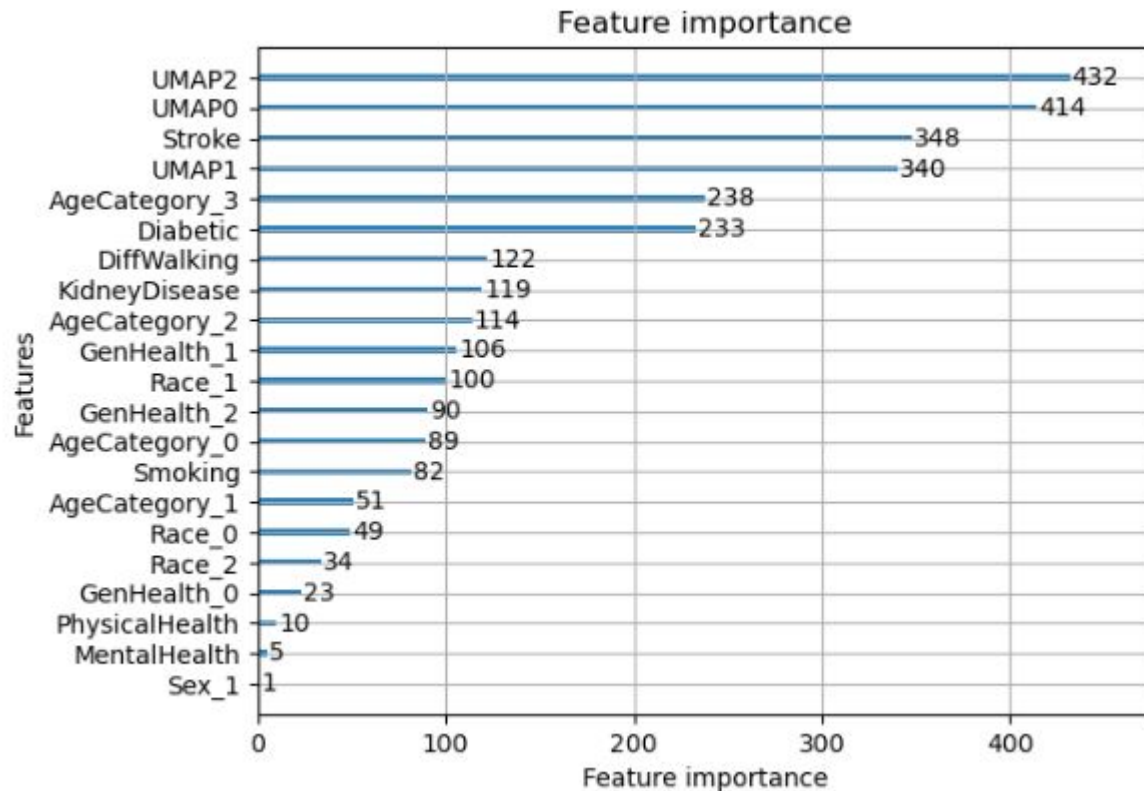
Documentación de
LightGBM:

<https://lightgbm.readthedocs.io/en/latest/index.html>



Entrenamiento de modelos

Modelo de
LightGBM:



Entrenamiento de modelos

Modelo de **Random Forest**:

“Un Random Forest es un conjunto (ensemble) de árboles de decisión combinados con bagging”

Bagging consiste básicamente en:

1. Aplicar el mismo clasificador n veces usando bootstrapping
2. Esto sería tomar muestras del set de train (con reemplazo) de igual tamaño
3. Promediamos sus resultados

Entrenamiento de modelos

Modelo de Random Forest:

Búsqueda de hiperparametros:

```
rf = RandomForestClassifier(random_state=10)
```

```
params = [  
    {'n_estimators': [80, 100, 120]},  
    {'max_depth': [5, 10, 15]},  
    {'min_samples_split': [2, 5]},  
    {'min_samples_leaf': [1, 2]},  
    {"random_state": [10]}  
]
```

```
gs = GridSearchCV(rf, param_grid = params, cv=5, scoring='recall')
```

Hiperparametros más importantes

Usamos cross-validation

Optimizamos el recall

Entrenamiento de modelos

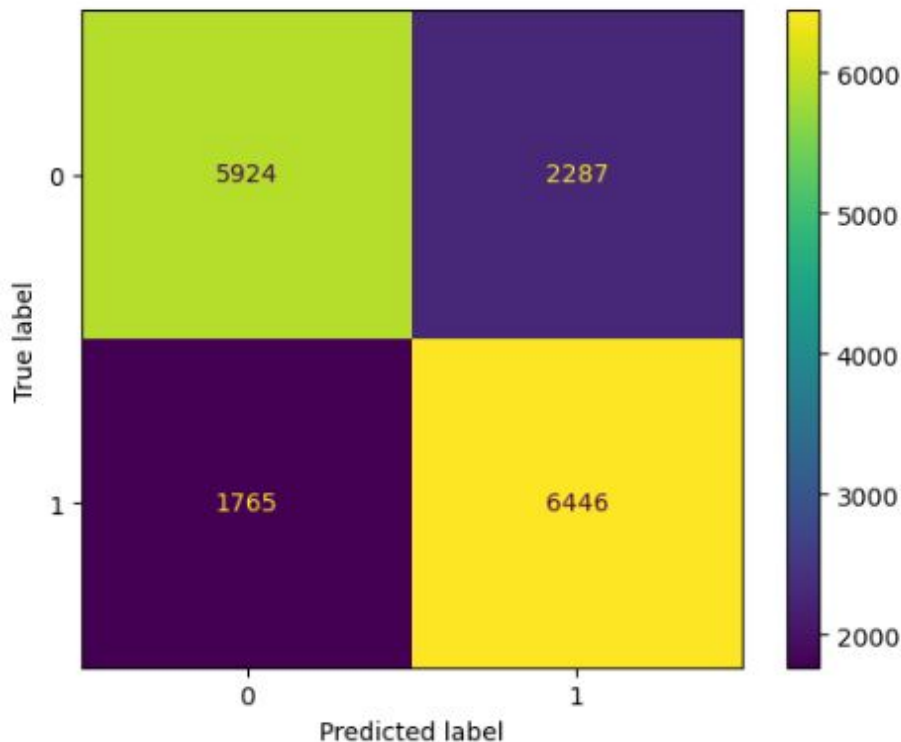
Modelo de Random Forest:

```
recall_score(y_test, preds)
```

0.7850444525636342

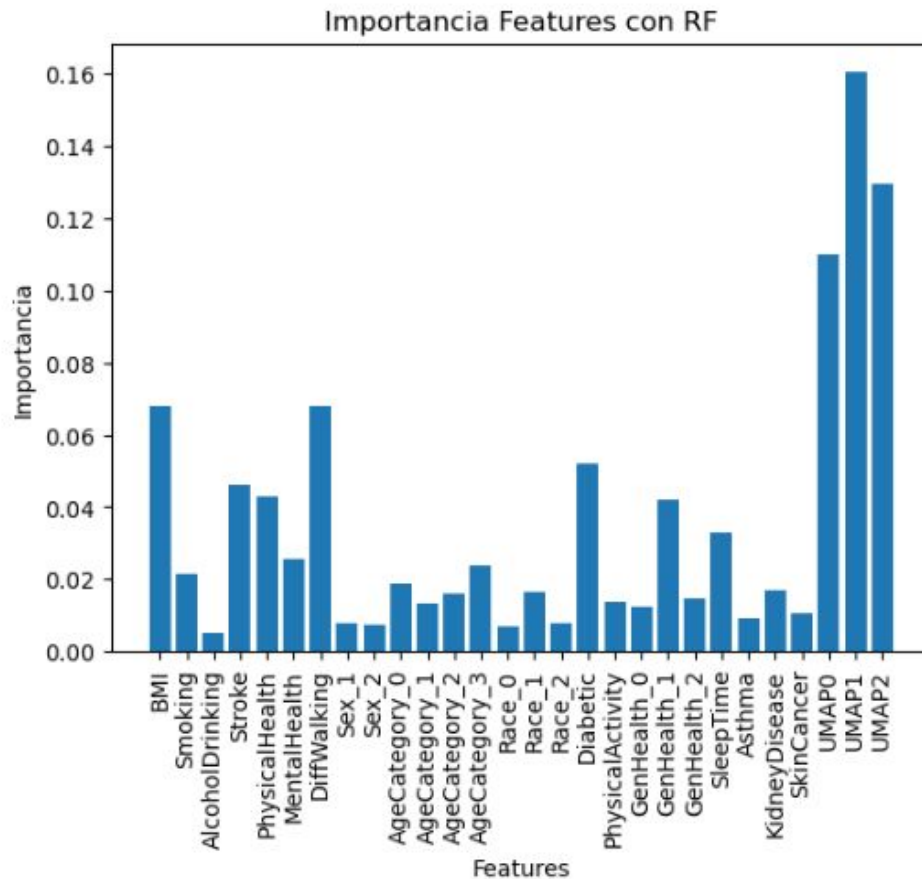
Documentación de
RandomForest
(classifier):

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>



Entrenamiento de modelos

Modelo de
Random
Forest:



Entrenamiento de modelos

Modelo de **XGBoost**:

“XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way”

Boosting consiste básicamente en:

1. Entrenar algoritmo simple y analizar sus resultados
2. Entrenar otro algoritmo simple en donde se le de mayor peso a los resultados para los cuales el anterior tuvo peor performance
3. Resultado final en base a un promedio ponderado

Entrenamiento de modelos

Modelo de XGBoost:

Búsqueda de hiperparametros:

```
xgb_clf = xgb.XGBClassifier(random_state=10)
```

```
params = [  
    {'objective': ["binary:logistic"]},  
    {'max_depth': [5, 10, 15]},  
    {"learning_rate": [0.05, 0.1, 0.3]},  
    {"alpha" : [5, 7, 10]},  
    {"n_estimators" : [60, 100, 150]},  
    {"random_state": [10]}  
]
```

```
gs = GridSearchCV(xgb_clf, param_grid = params, cv=5, scoring='recall')
```

Hiperparametros más importantes

Usamos cross-validation

Optimizamos el recall

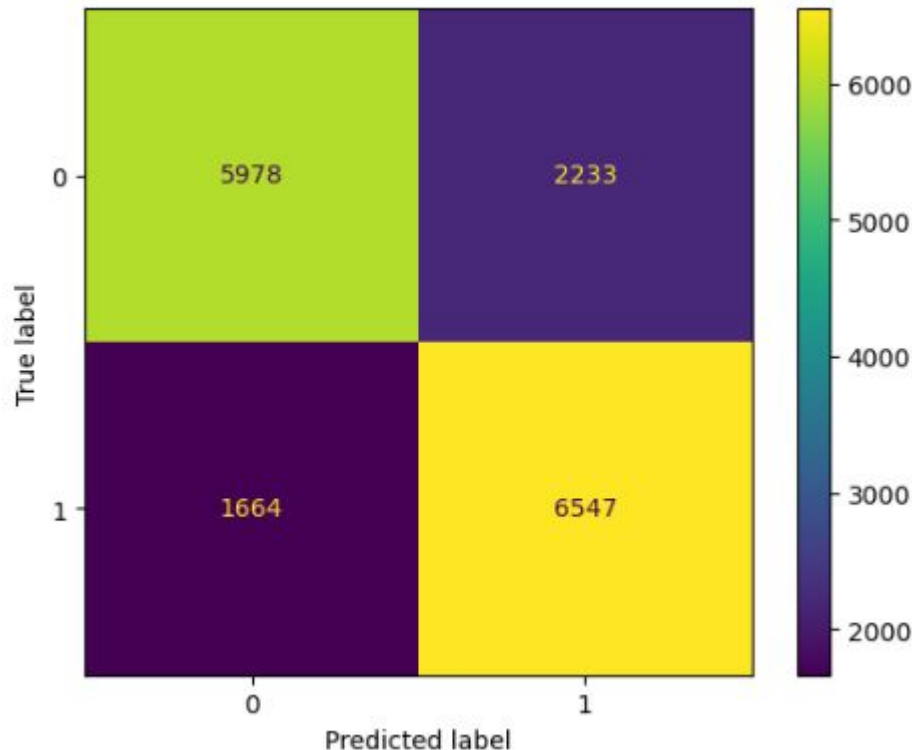
Entrenamiento de modelos

Modelo de XGBoost:

```
recall_score(y_test, preds)
```

```
0.7973450249665084
```

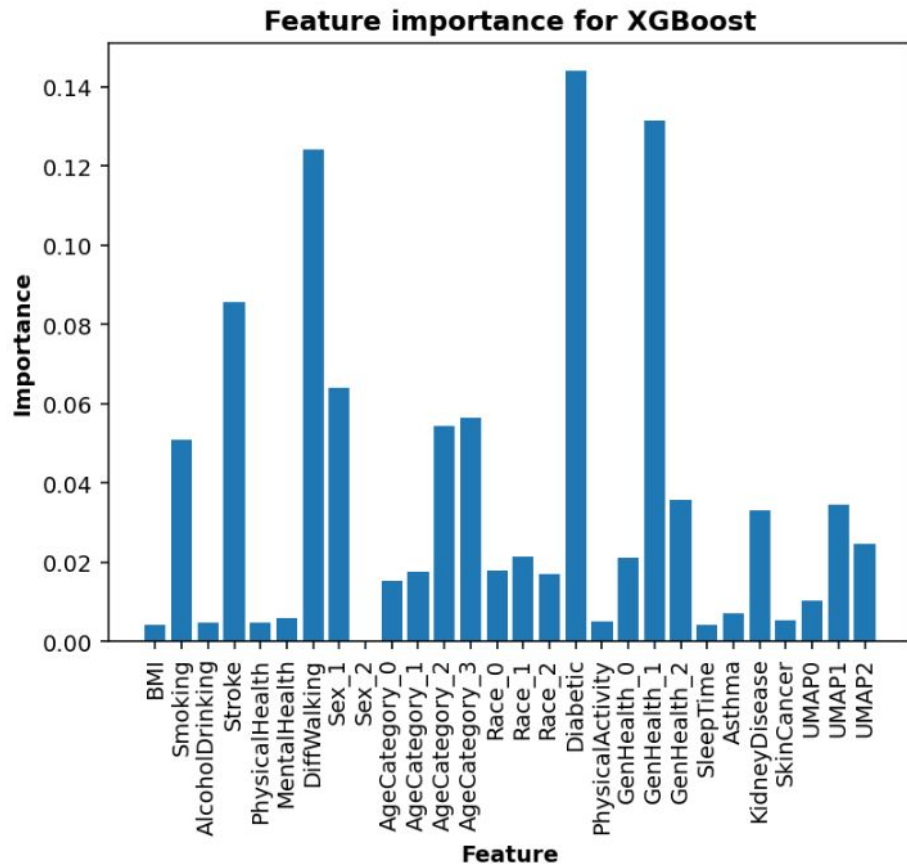
Documentación de XGBoost:
<https://xgboost.readthedocs.io/en/stable/>



Entrenamiento de modelos

Modelo de XGBoost:

Pese a que internamente usa boosting como LightGBM vemos que hay una diferencia en las features más importantes para cada algoritmo



Entrenamiento de modelos

Ensamble de los 3 modelos (LightGBM, Random Forest, XGBoost) usando **Voting Classifier**:

Esto consiste básicamente en:

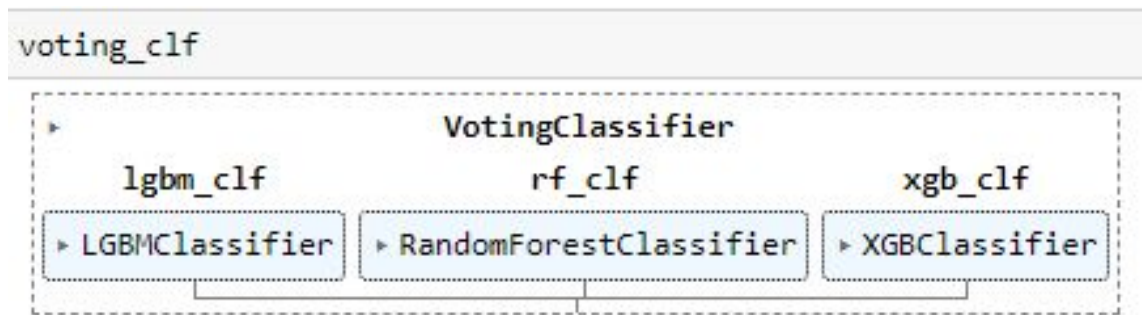
1. Cada clasificador realiza sus predicciones (1 o 0)
2. Cada clasificador aportará un voto

Ejemplos: Si nuestros 3 algoritmos predicen (1, 1, 0) será un 1 el resultado final.

En cambio si predicen (0,1,0) será un 0 el resultado final

Entrenamiento de modelos

Voting Classifier:

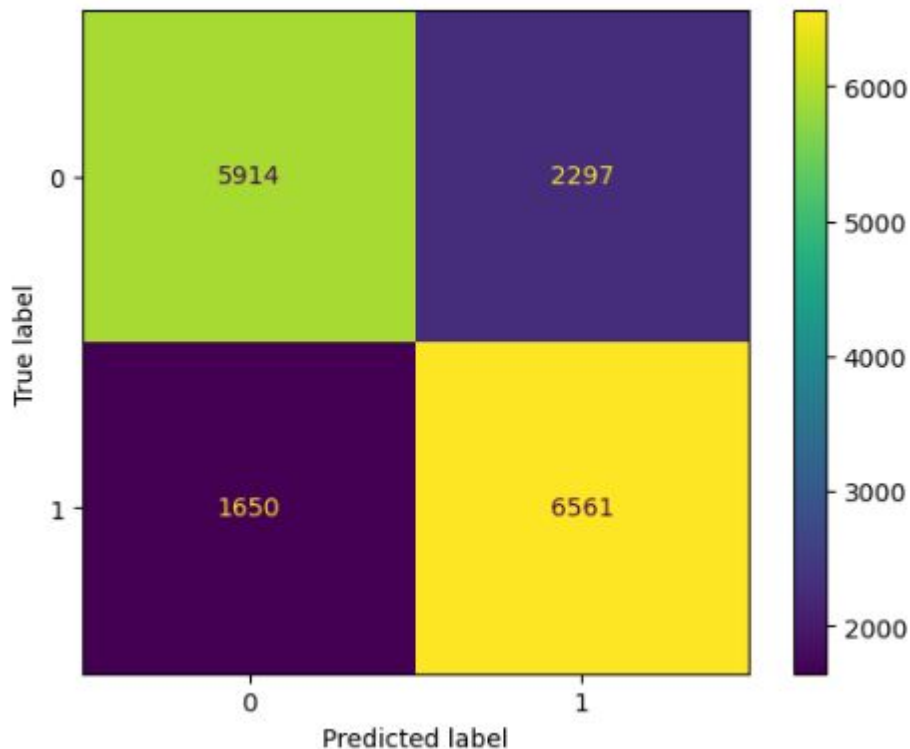


Entrenamiento de modelos

Voting
Classifier:

```
recall_score(y_test, preds)
```

```
0.7990500548045305
```



Entrenamiento de modelos

Voting Classifier:

No mejora mucho frente a cada clasificador por separado en este caso

Creemos que se debe a que los 3 algoritmos realizan predicciones muy similares (ver matriz de confusión en cada modelo) y además los 3 están basados en árboles

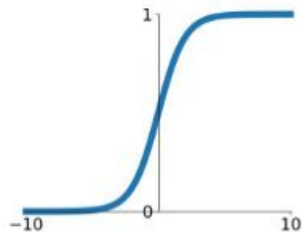
Entrenamiento de modelos

Red Neuronal:

Hicimos dos redes diferentes, las funciones matemáticas que utilizamos son:

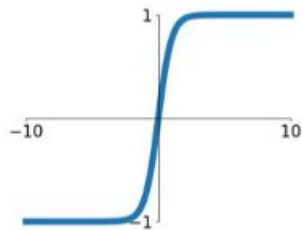
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



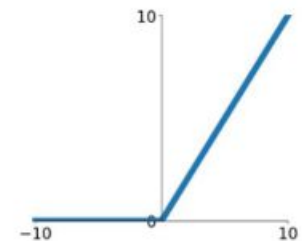
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



Entrenamiento de modelos

Red Neuronal:

En ambos modelos la función sigmoide va en la última capa, que contiene solo 1 neurona

Esto nos dirá la probabilidad de un dato de ser 1, es decir, de tener riesgo de un problema cardíaco

Debemos aplicar entonces un threshold (límite que decidimos arbitrariamente) para clasificar los datos en 1 o en 0

Entrenamiento de modelos

— — —

Red Neuronal:
Primer modelo

```
model = Sequential([
    Dense(1000, activation="tanh", input_shape=(28,)),
    Dense(100, activation="tanh"),
    Dense(10, activation="tanh"),
    Dense(1, activation="sigmoid")
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1000)	29000
dense_1 (Dense)	(None, 100)	100100
dense_2 (Dense)	(None, 10)	1010
dense_3 (Dense)	(None, 1)	11

=====
Total params: 130,121
Trainable params: 130,121
Non-trainable params: 0
=====

Entrenamiento de modelos

Red Neuronal: Primer modelo

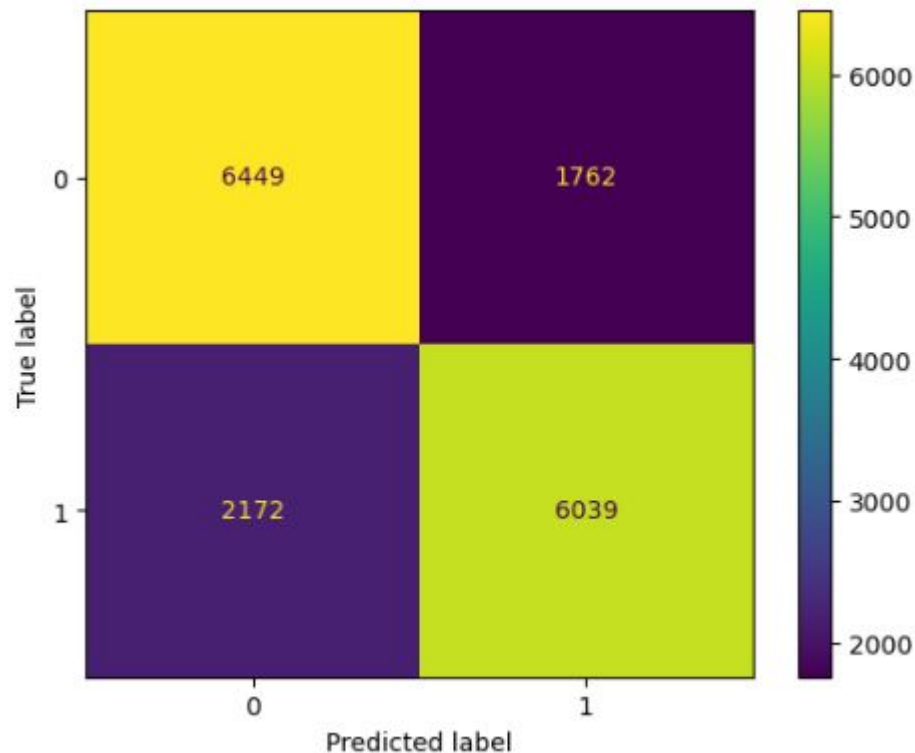
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_05)
```

```
0.7354767994154183
```

```
accuracy_score(y_test, preds_05)
```

```
0.7604433077578857
```



No esta tan bueno

Entrenamiento de modelos

Red Neuronal: Primer modelo

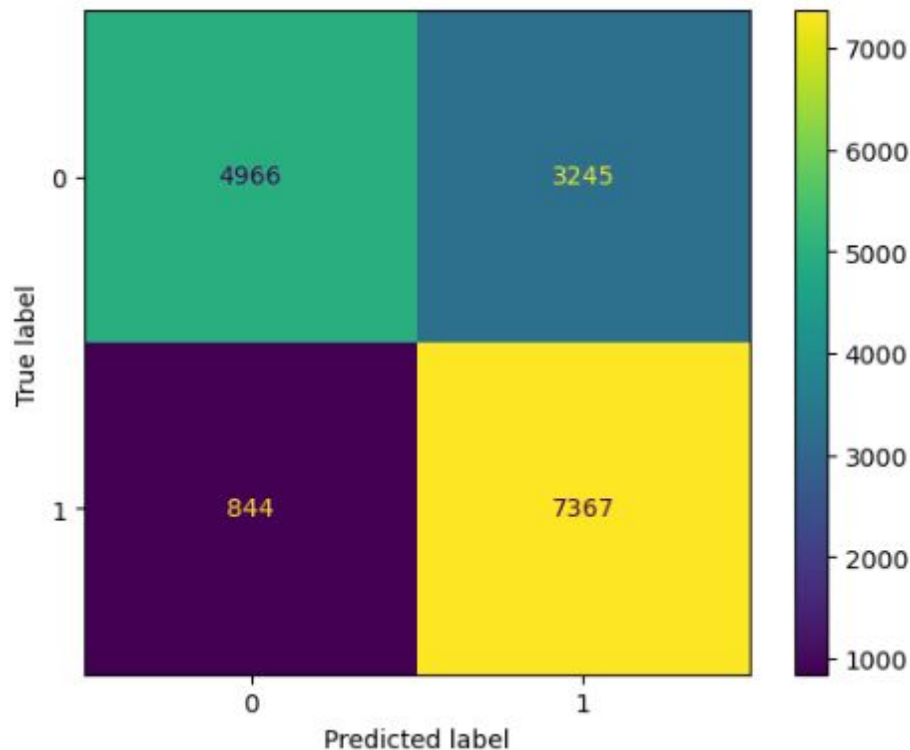
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_03)
```

```
0.897211058336378
```

```
accuracy_score(y_test, preds_03)
```

```
0.7510047497259773
```



Entrenamiento de modelos

Red Neuronal: Primer modelo

Vemos que llegamos a un resultado mucho mejor

No perdemos casi nada de accuracy pero reducimos en gran cantidad los casos de falsos negativos, lo cual es muy bueno para nuestro problema en particular

Entrenamiento de modelos

— — —

Red Neuronal:

Segundo modelo

```
model_1 = Sequential([
    Dense(1000, activation="relu", input_shape=(28,)),
    Dense(100, activation="relu"),
    Dense(10, activation="relu"),
    Dense(1, activation="sigmoid")
])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1000)	29000
dense_1 (Dense)	(None, 100)	100100
dense_2 (Dense)	(None, 10)	1010
dense_3 (Dense)	(None, 1)	11

Total params: 130,121

Trainable params: 130,121

Non-trainable params: 0

Entrenamiento de modelos

Red Neuronal:
Segundo modelo

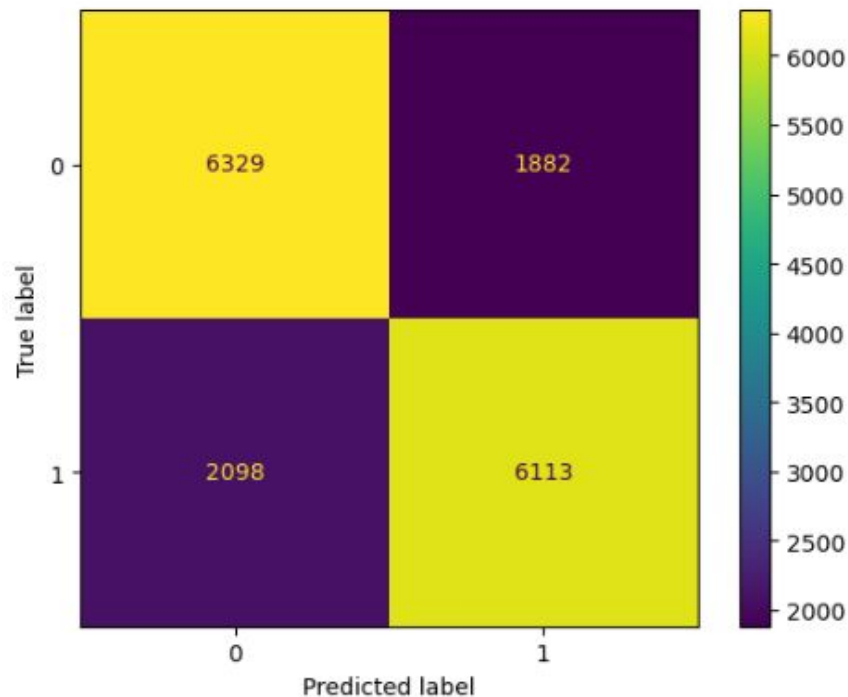
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_05)
```

```
0.7444890999878212
```

```
accuracy_score(y_test, preds_1_05)
```

```
0.7576421873097064
```



No esta tan bueno nuevamente

Entrenamiento de modelos

Red Neuronal:
Segundo modelo

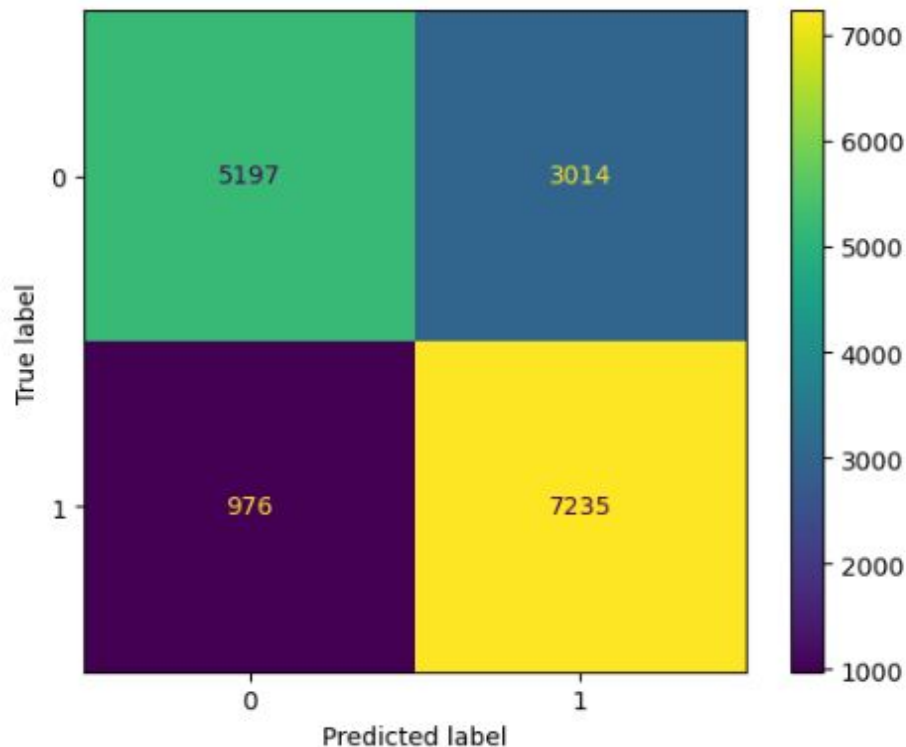
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_03)
```

0.8811350627207405

```
accuracy_score(y_test, preds_1_03)
```

0.7570332480818415



Entrenamiento de modelos

Red Neuronal: Segundo modelo

Vemos que llegamos a un resultado mucho mejor como sucedió antes

No perdemos casi nada de accuracy pero reducimos en gran cantidad los casos de falsos negativos, lo cual es muy bueno para nuestro problema en particular

Pese a esto, el primer modelo sigue siendo mejor (ya que queremos optimizar el recall)

Conclusiones

Conclusiones

En base a los modelos utilizados, decidimos quedarnos con la primer Red Neuronal que mostramos, con un threshold de 0,3 porque era en el que mejor métrica alcanzamos

Además, lo bueno de utilizar una red neuronal es que al predecir las probabilidades nosotros luego podemos decidir dónde aplicar el corte, como ya mostramos

Esto nos permite ser más o menos conservadores, obviamente se elegirá dependiendo el problema en particular que tratemos

Conclusiones

Importancia de la recolección de datos de calidad:

El éxito de los modelos de inteligencia artificial depende en gran medida de la calidad de los datos utilizados.

Es crucial asegurar que el dataset utilizado esté bien recopilado, balanceado, limpio y representativo de la población objetivo.

Conclusiones

En base a hábitos fácilmente medibles como si fumas, si haces actividad física, horas de sueño, la edad, si tomas alcohol...

Podemos hacer un modelo que sirve como filtro para determinar si tenemos más o menos chances de sufrir un ataque al corazón sin necesidad de nada muy complejo, simplemente completar unas preguntas

Esto ayuda a la medicina y a la sociedad en general con un problema de índole general que todos podemos sufrir como son los ataques al corazón

Conclusiones

Si bien nosotros nos centramos en los ataques al corazón, queremos dejar en claro que este modelo y forma de trabajo podría adaptarse a otras enfermedades como la detección de algún tipo de cáncer, alzheimer, asma y cualquier otra enfermedad que se nos ocurra

Obviamente deberíamos tener otros datos que sean relevantes para la enfermedad que se quiere diagnosticar

Hasta acá llegamos antes

Roadmap de los pasos a seguir que trabajamos:

1. No utilizar las reducciones (no UMAP)
2. Usar otros encodings
3. Utilizar solo los features más importantes
4. Encadenamiento de modelos

— — —

Entrenamiento sin reducciones

Entrenamiento sin reducciones:

Antes hicimos una reducción de datos con UMAP en 3 dimensiones y agregamos esos datos al DataFrame

Ahora eliminamos esas 3 columnas para ver cómo se comportarían nuestros modelos sin estas

Es interesante hacer esta comparación ya que algunos modelos tenían como las features más importantes a las columnas creadas por la reducción

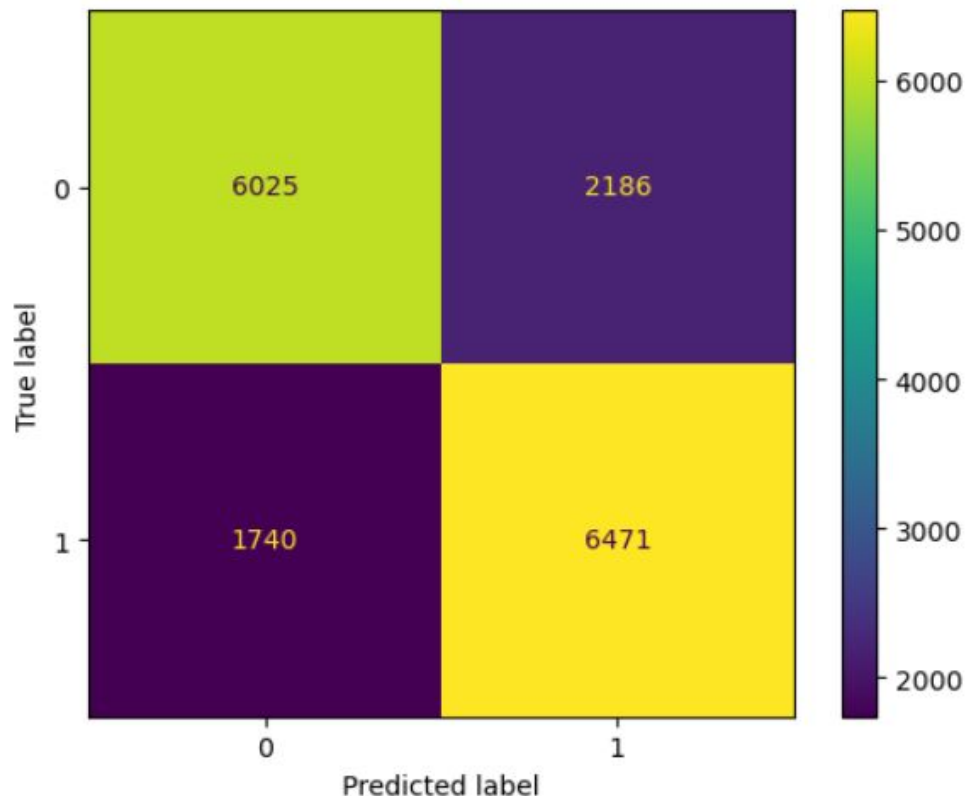
Entrenamiento sin reducciones

Modelo de LightGBM:

```
recall_score(y_test, preds)
```

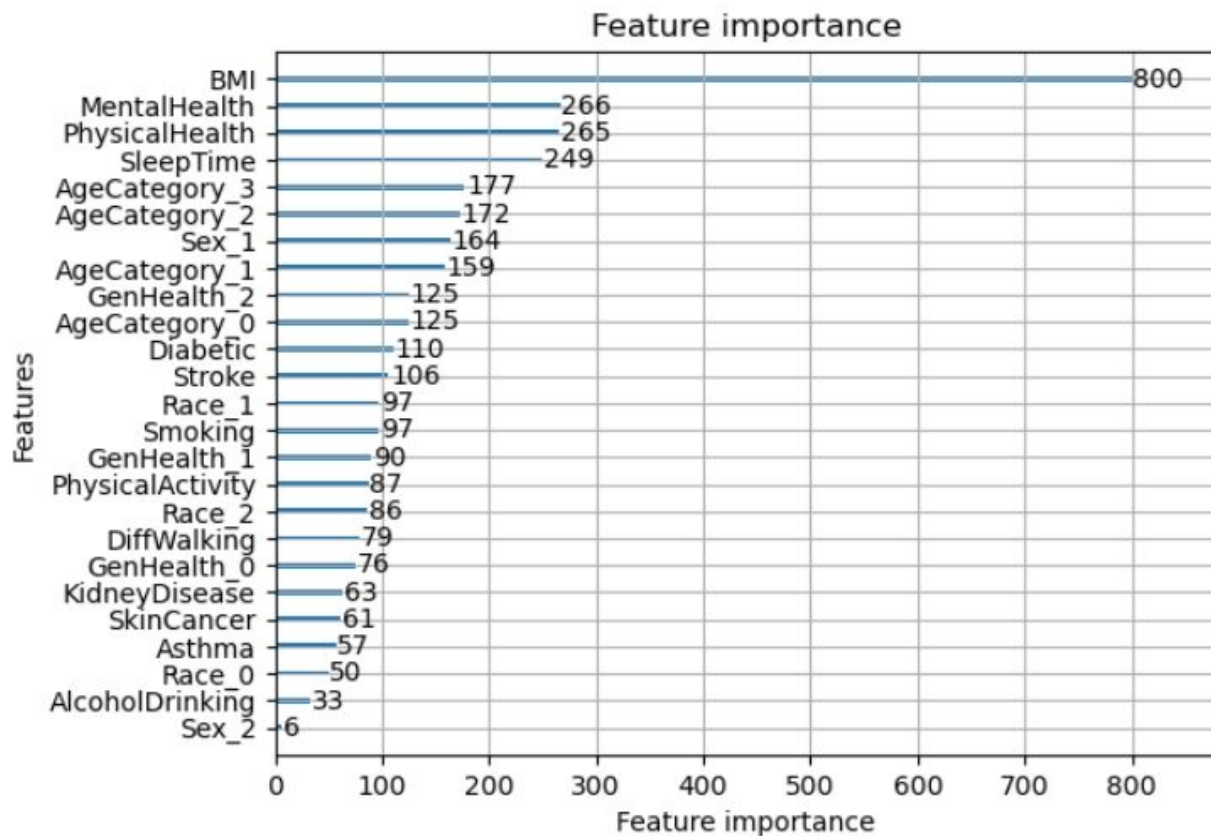
0.7880891487029594

Búsqueda de
hiperparametros igual que
antes
Usamos CV
Optimizamos Recall



Entrenamiento sin reducciones

Modelo de
LightGBM:



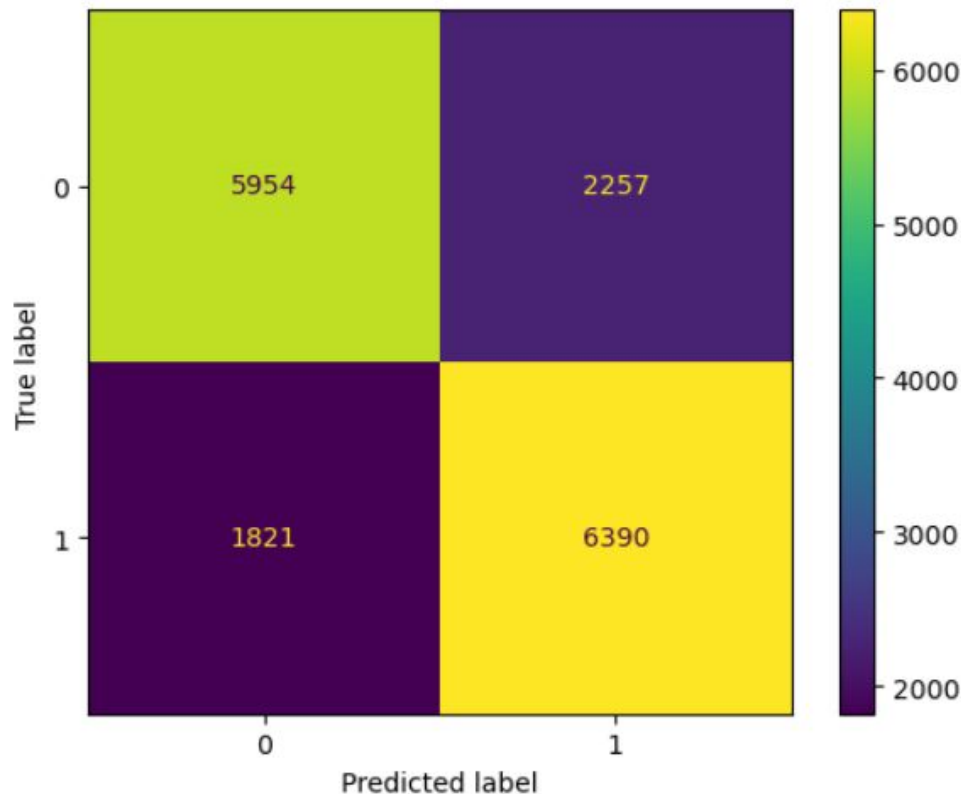
Entrenamiento sin reducciones

Modelo de RandomForest:

```
recall_score(y_test, preds)
```

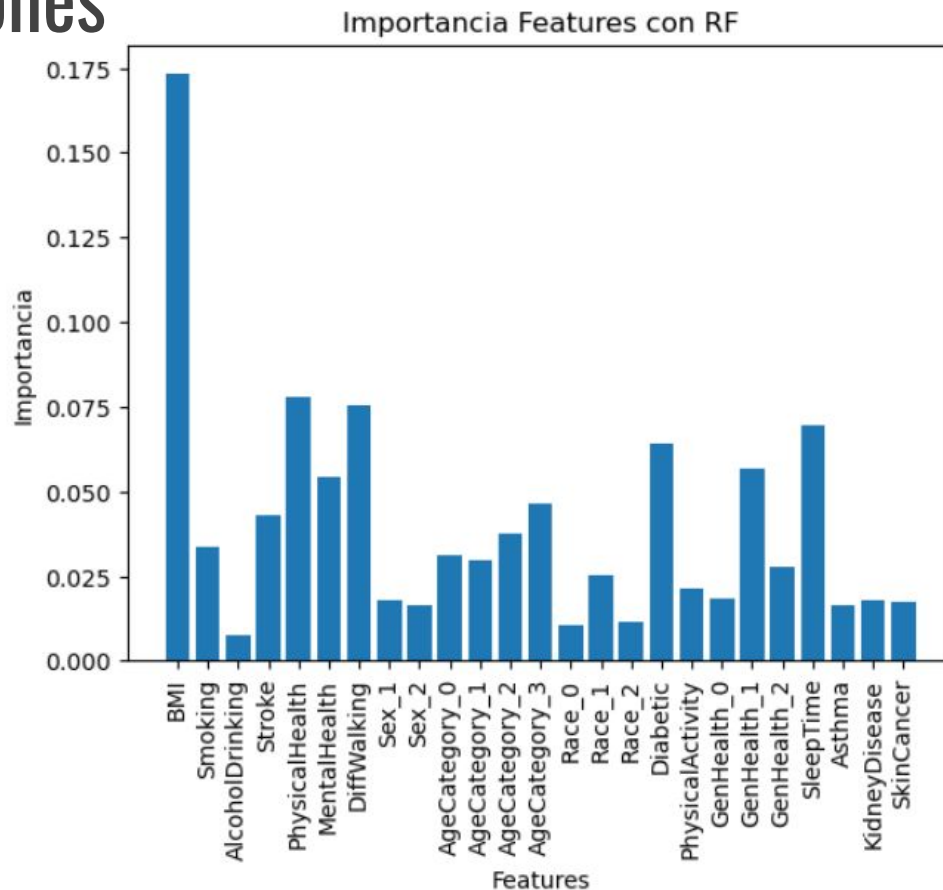
0.7782243332115455

Búsqueda de
hiperparametros igual que
antes
Usamos CV
Optimizamos Recall



Entrenamiento sin reducciones

Modelo de
RandomForest:



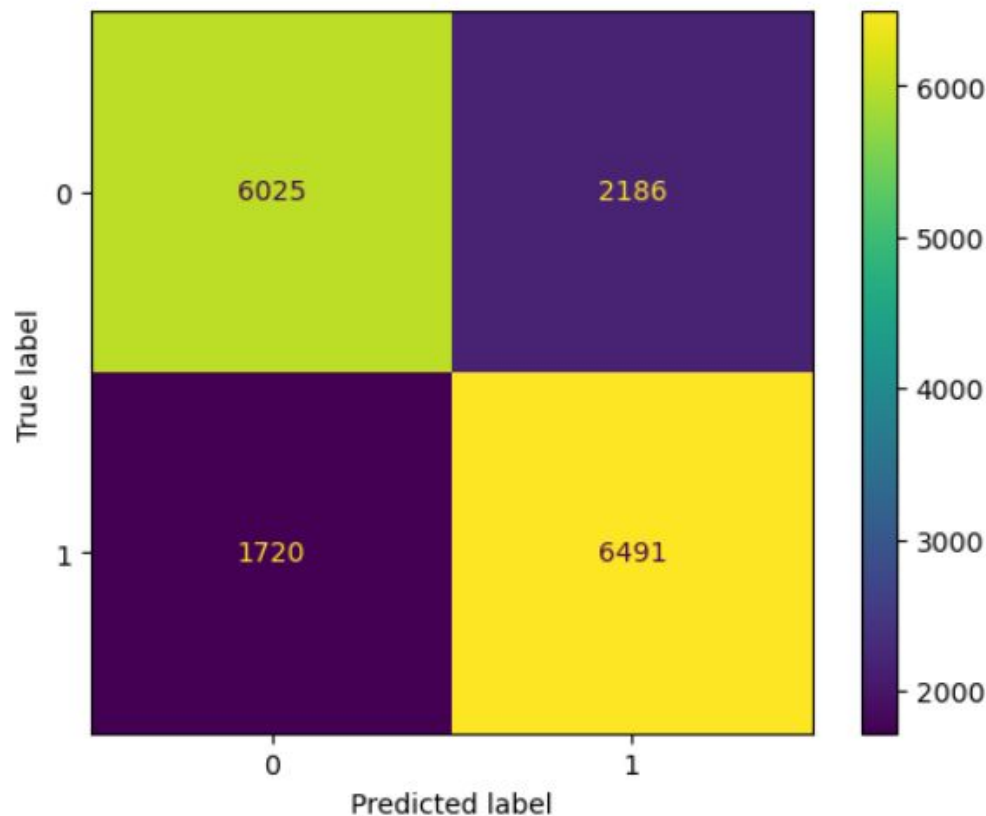
Entrenamiento sin reducciones

Modelo de XGBoost:

```
recall_score(y_test, preds)
```

0.7905249056144197

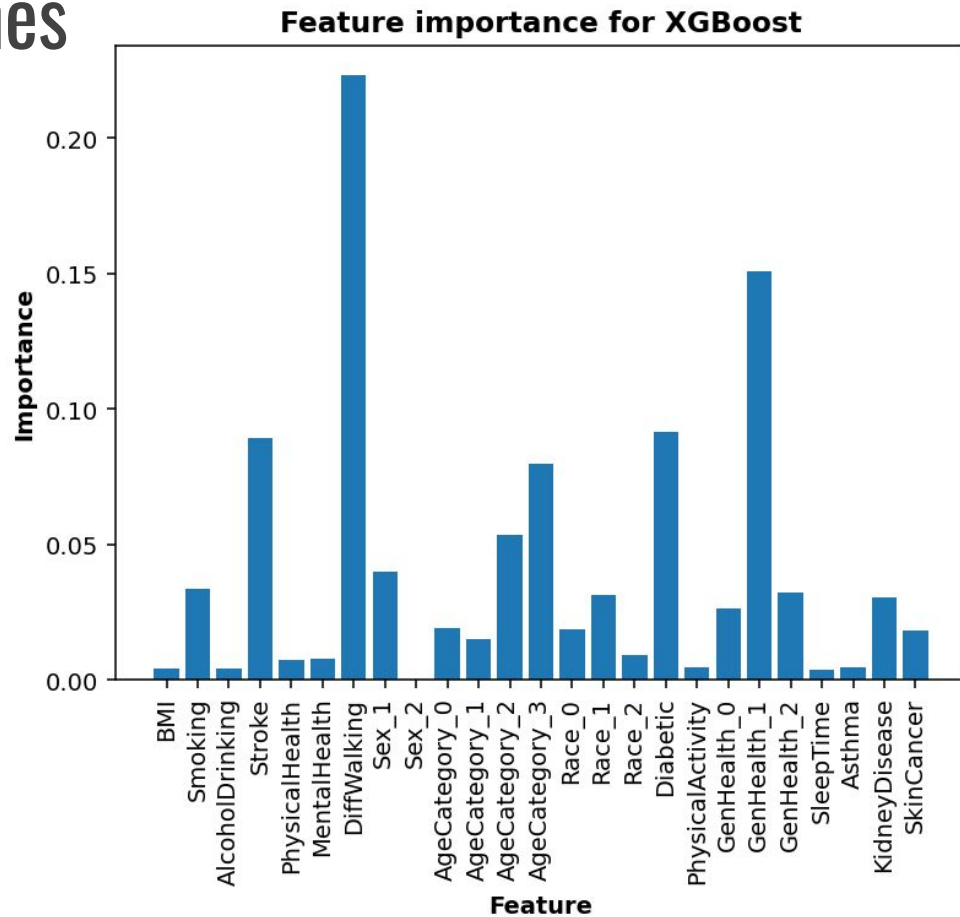
Búsqueda de
hiperparametros igual que
antes
Usamos CV
Optimizamos Recall



Entrenamiento sin reducciones

— — —

Modelo de
XGBoost:

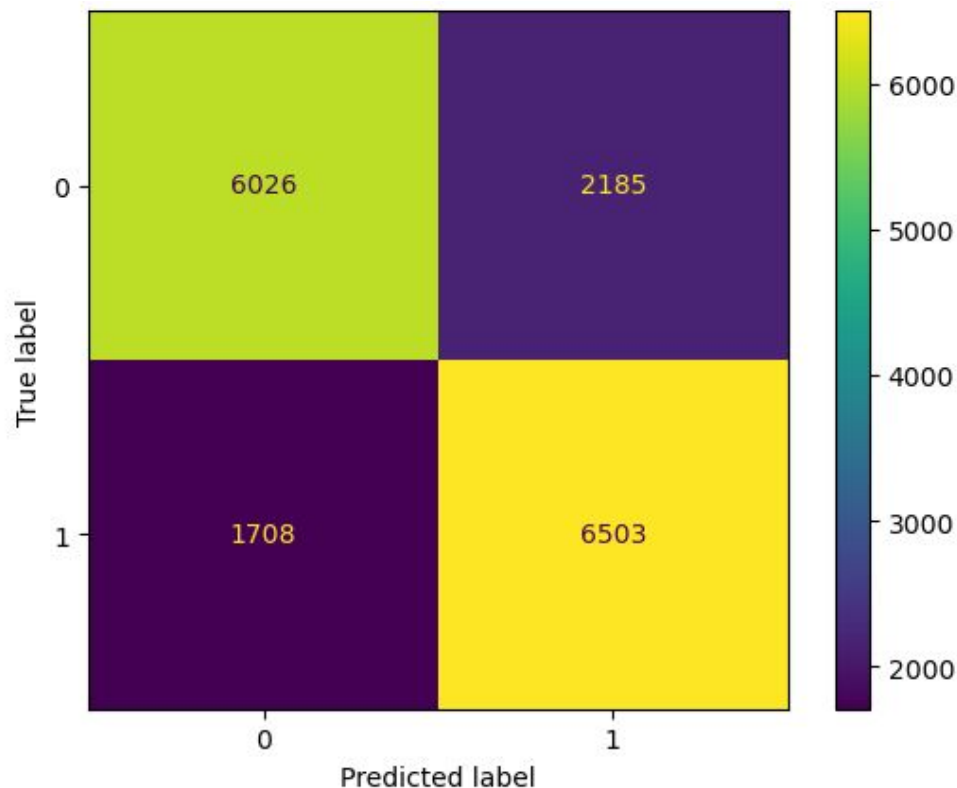


Entrenamiento sin reducciones

Modelo de Voting
Classifier:

```
recall_score(y_test, preds)
```

0.7919863597612958



Entrenamiento sin reducciones

Red Neuronal: Primer modelo

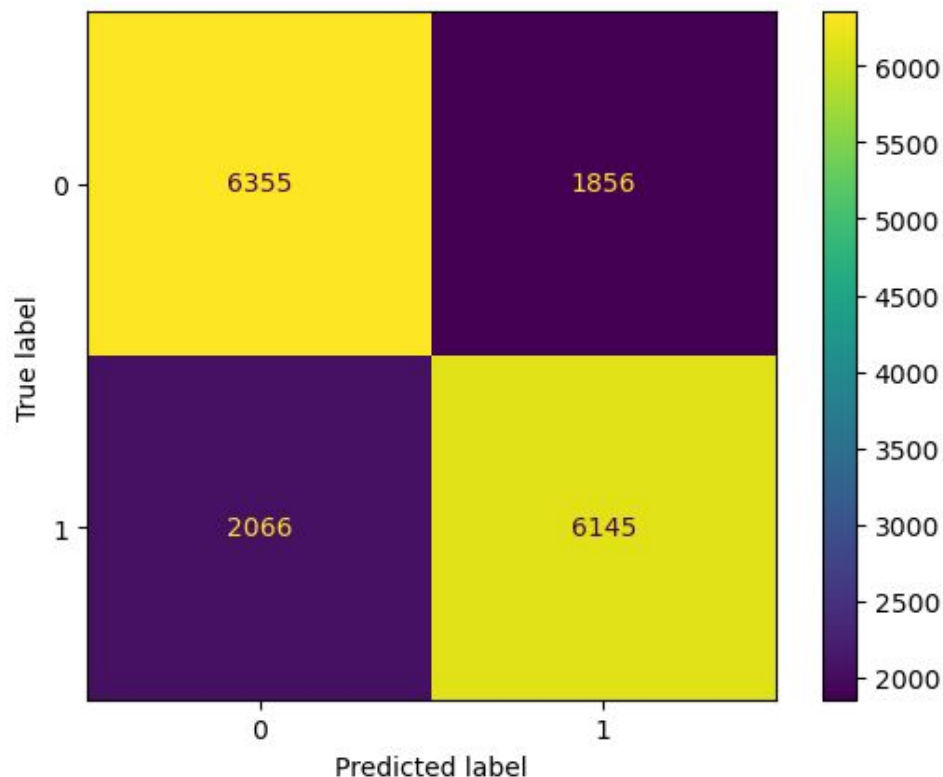
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_05)
```

```
0.7483863110461576
```

```
accuracy_score(y_test, preds_05)
```

```
0.7611740348313238
```



Entrenamiento sin reducciones

Red Neuronal: Primer modelo

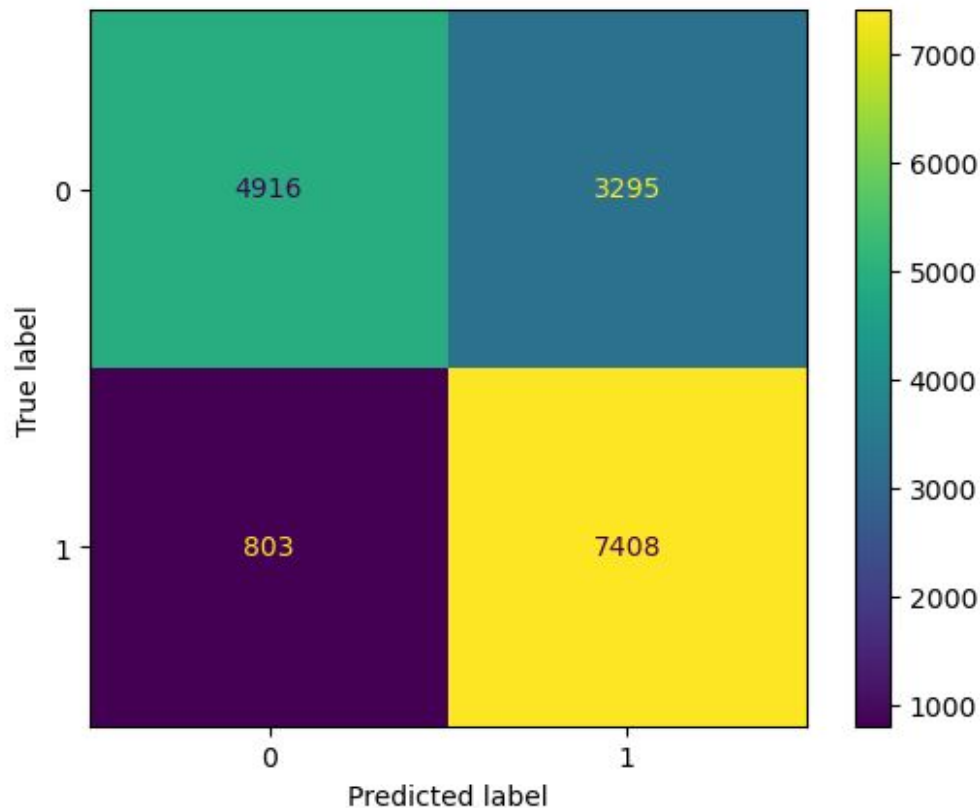
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_03)
```

```
0.9022043600048715
```

```
accuracy_score(y_test, preds_03)
```

```
0.7504567044208988
```



Entrenamiento sin reducciones

Red Neuronal:
Segundo modelo

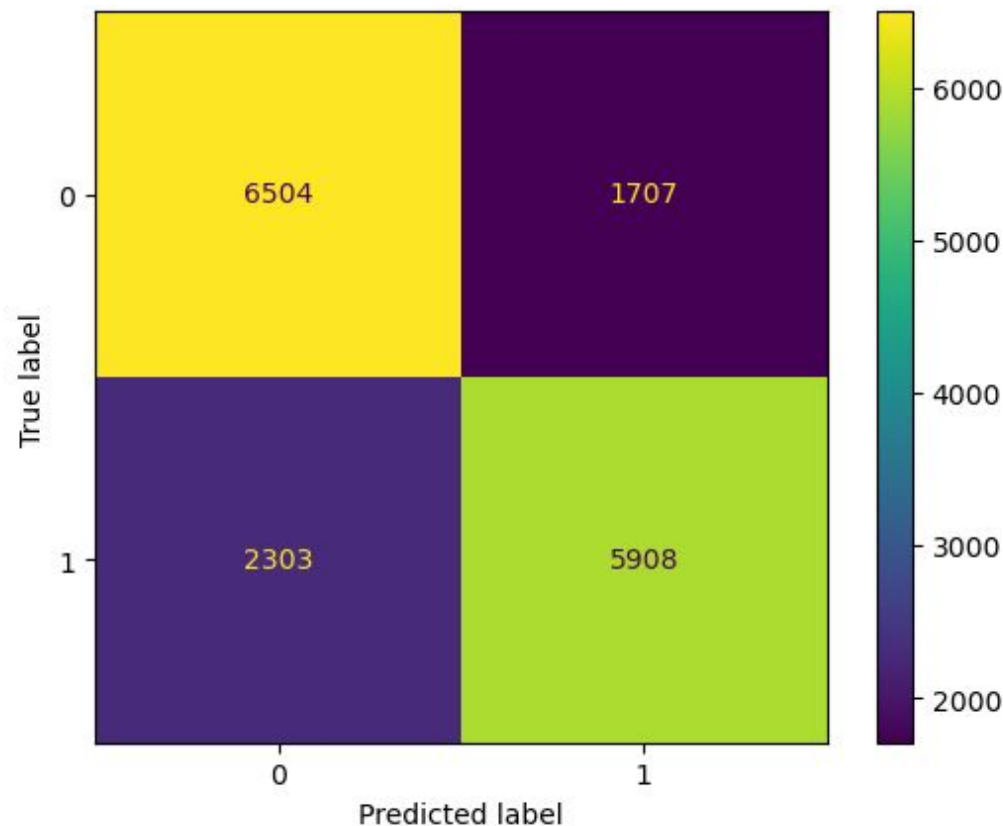
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_05)
```

```
0.7195225916453538
```

```
accuracy_score(y_test, preds_1_05)
```

```
0.7558153696261113
```



Entrenamiento sin reducciones

Red Neuronal:
Segundo modelo

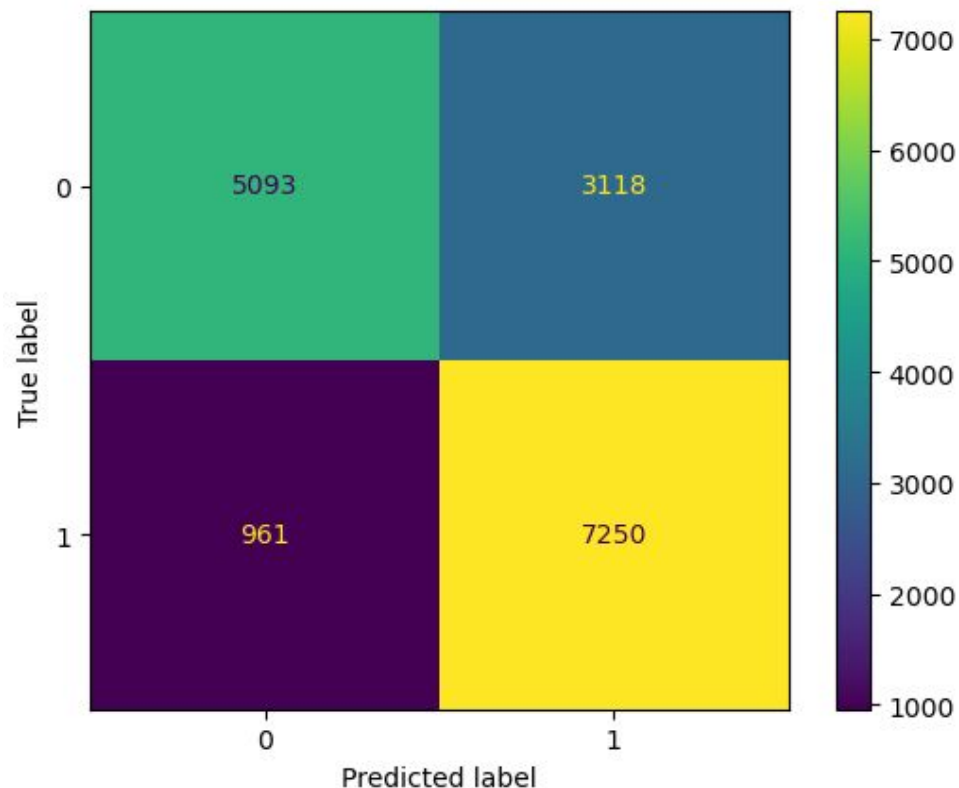
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_03)
```

```
0.8829618804043357
```

```
accuracy_score(y_test, preds_1_03)
```

```
0.7516136889538424
```



Usar otros encodings

Usar otros encodings

Vamos a cambiar los encodings para las columnas con datos numéricos

Estas son BMI, PhysicalHealth, MentalHealth, SleepTime

Usar otros encodings

Antes realizamos

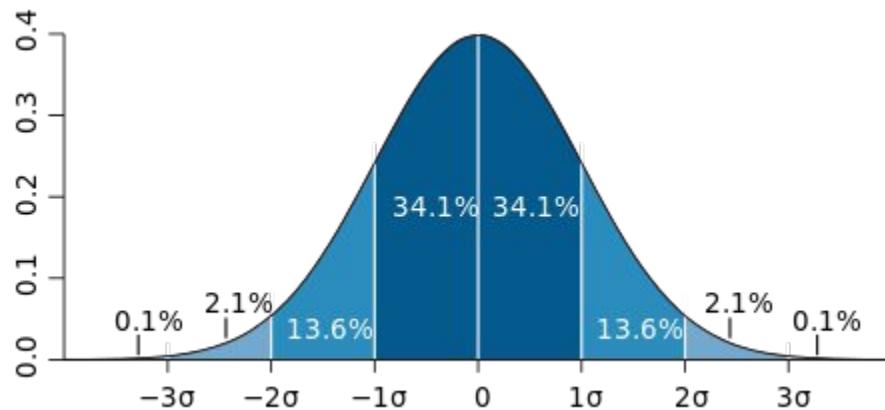
The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

u : promedio

s : desviación estándar(σ)

$$s = \sqrt{\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N - 1}}.$$



Distribución normal estándar

Usar otros encodings

Pero haciendo eso no quedaban los datos distribuidos de 0 a 1 (si bien se acercan al 0)

Por este motivo algunos algoritmos le daban más importancia a estos features ya que podían sacar más información de ellos

Entonces ahora aplicamos dos alternativas distintas para el encoding de estas features:

Usar otros encodings

MinMaxScaler:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

```
X_scaled = X_std * (max - min) + min
```

Con $\text{max} = 1$ y $\text{min} = 0$ para tener el mismo rango que el resto de features

Veamos los resultados

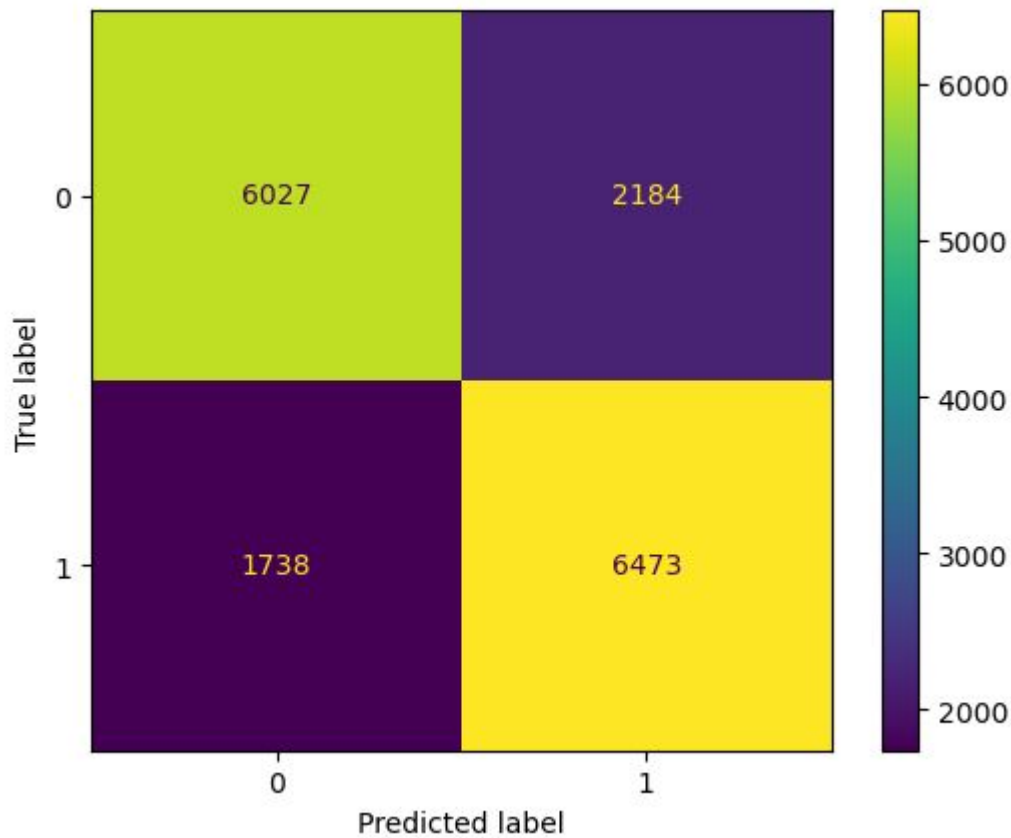
Entrenamiento con otros encodings

Modelo de LightGBM:

```
recall_score(y_test, preds)
```

0.7883327243941055

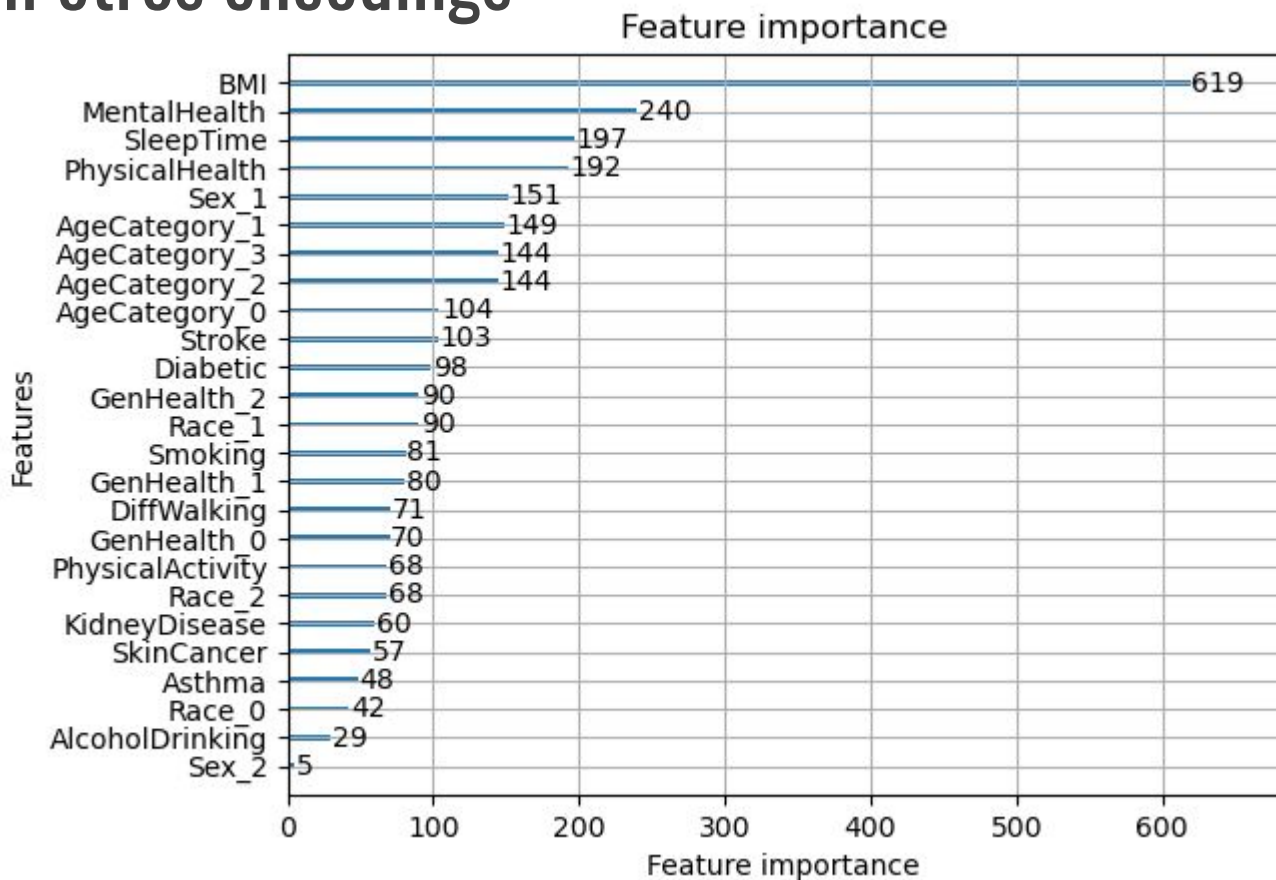
Encoding MinMax



Entrenamiento con otros encodings

Modelo de
LightGBM:

Encoding MinMax



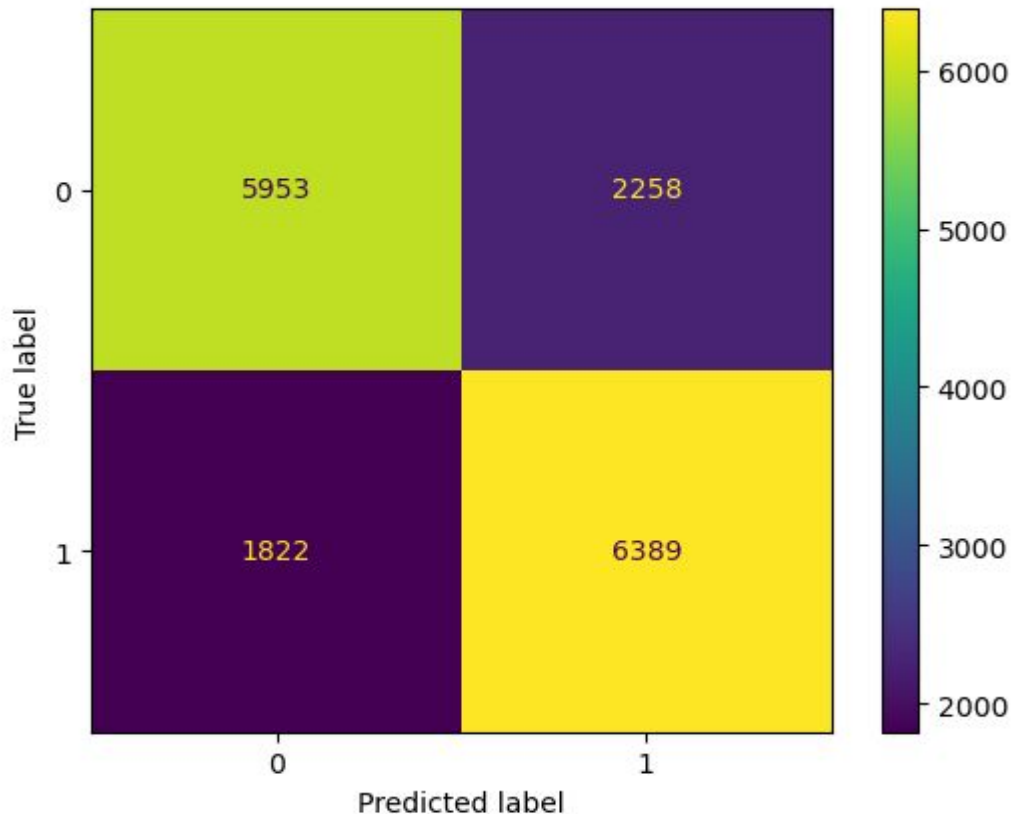
Entrenamiento con otros encodings

Modelo de
RandomForest:

```
recall_score(y_test, preds)
```

0.7781025453659725

Encoding MinMax



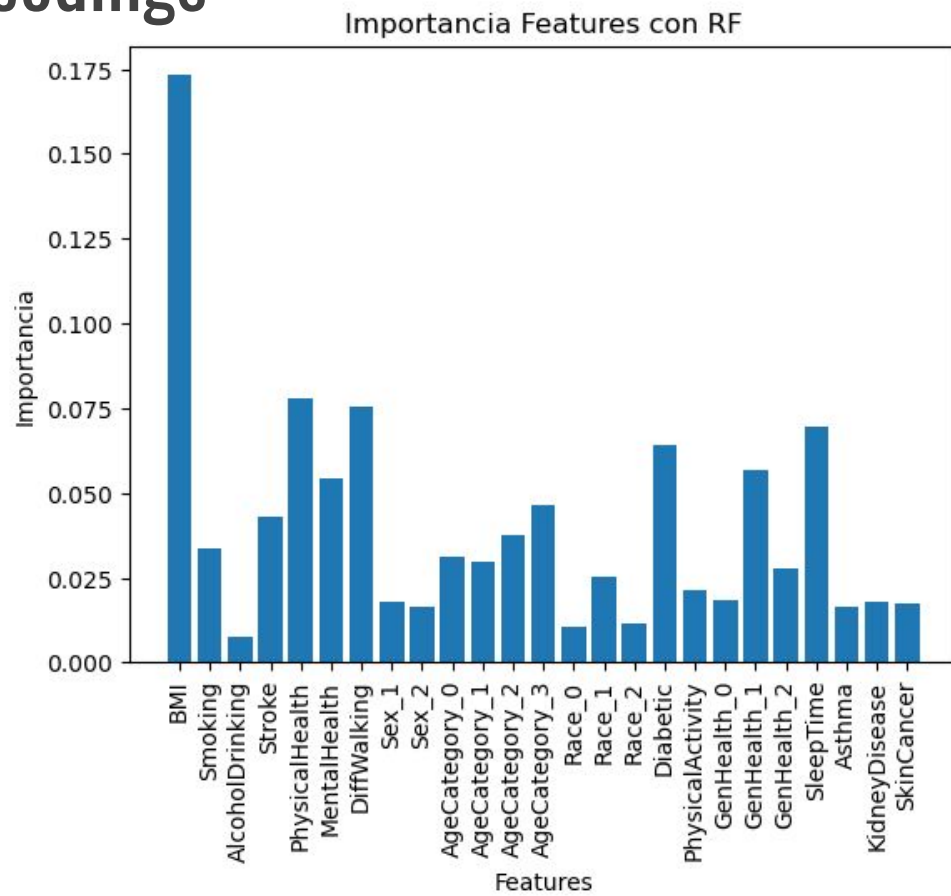
Entrenamiento con otros encodings

Modelo de

Random

Forest:

Encoding MinMax



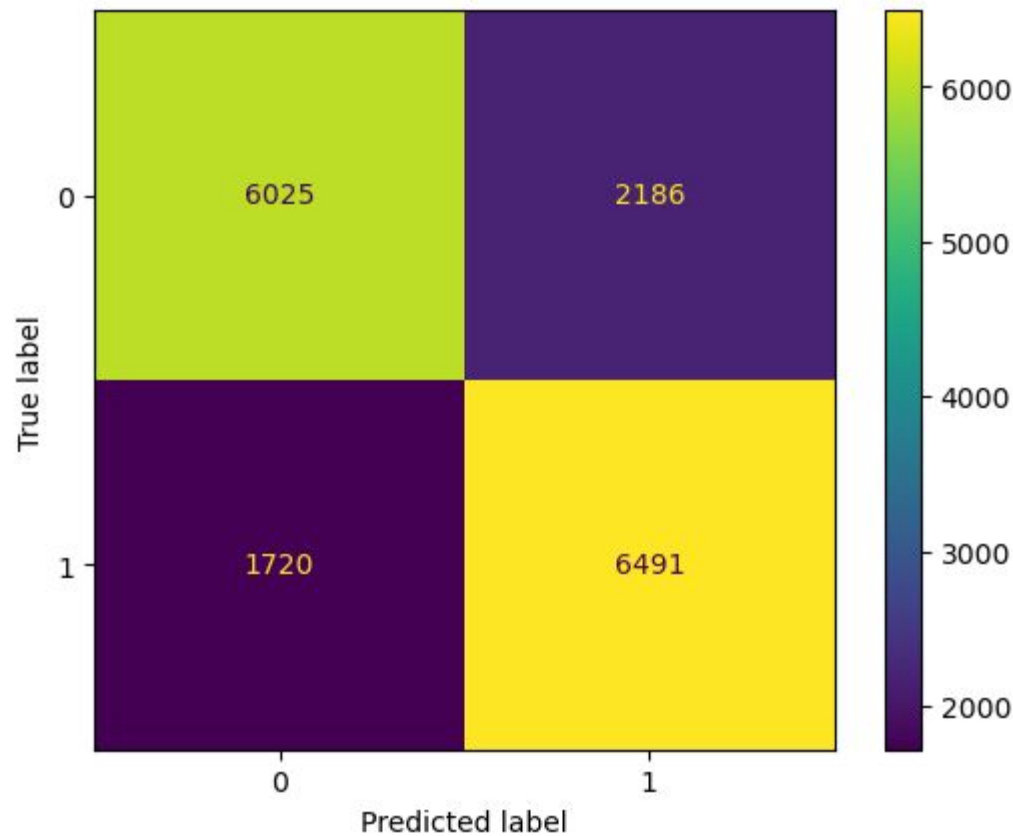
Entrenamiento con otros encodings

Modelo de XGBoost:

```
recall_score(y_test, preds)
```

```
0.7905249056144197
```

Encoding MinMax

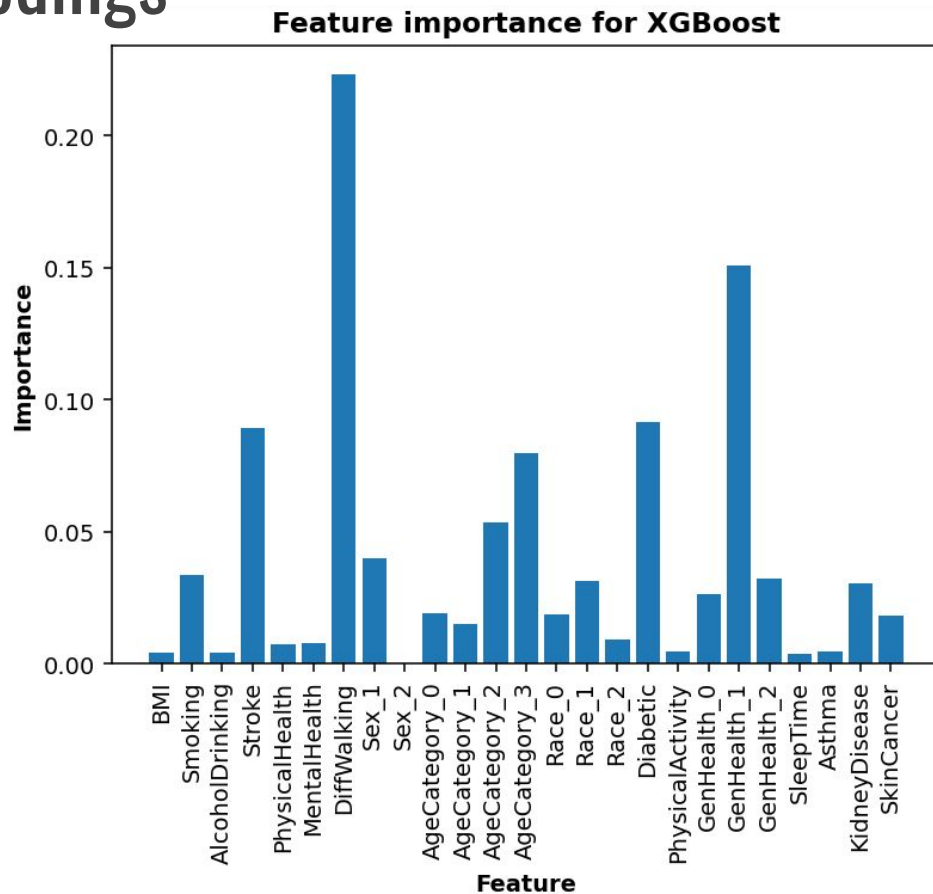


Entrenamiento con otros encodings

Modelo de

XGBoost:

Encoding MinMax



Entrenamiento con otros encodings

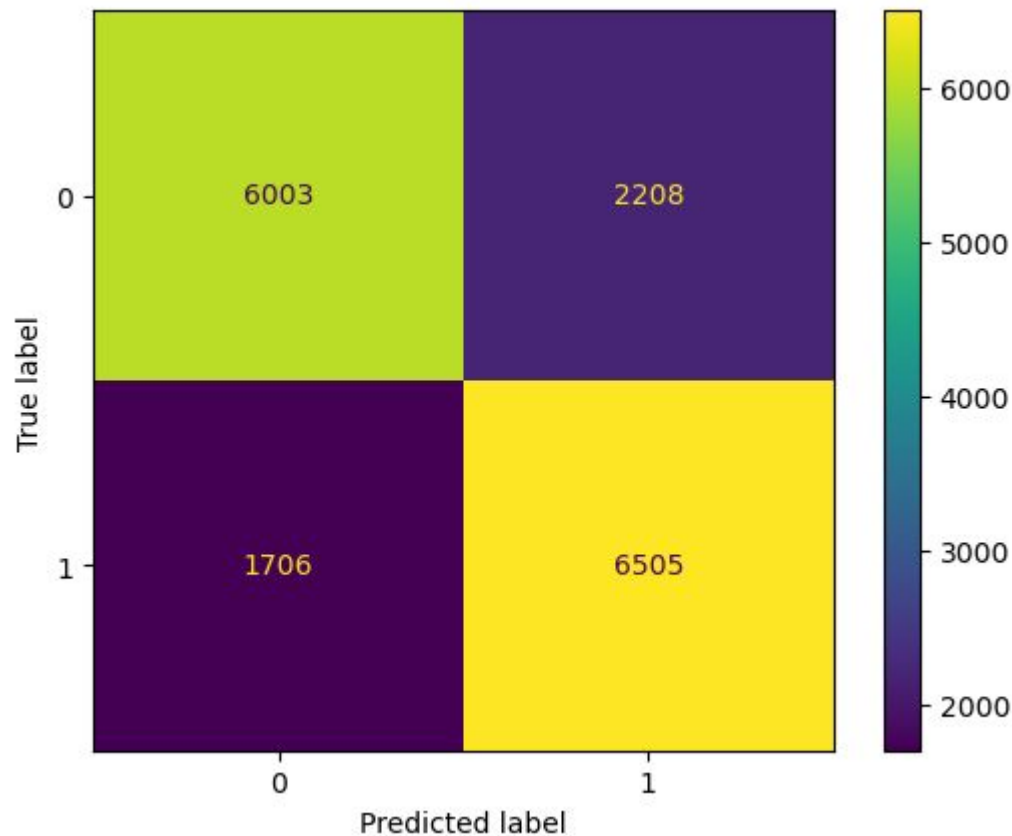
Modelo de

VotingClassifier:

```
recall_score(y_test, preds)
```

```
0.7922299354524418
```

Encoding MinMax



Entrenamiento con otros encodings

Red Neuronal: Primer modelo

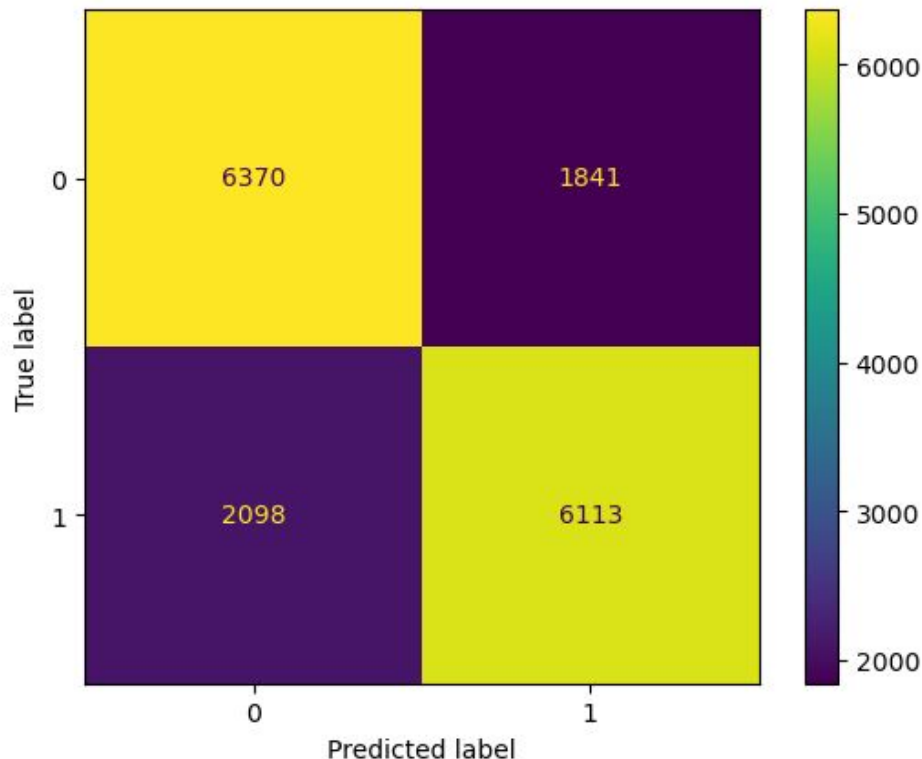
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_05)
```

```
0.7444890999878212
```

```
accuracy_score(y_test, preds_05)
```

```
0.7601388381439532
```



Entrenamiento con otros encodings

Red Neuronal: Primer modelo

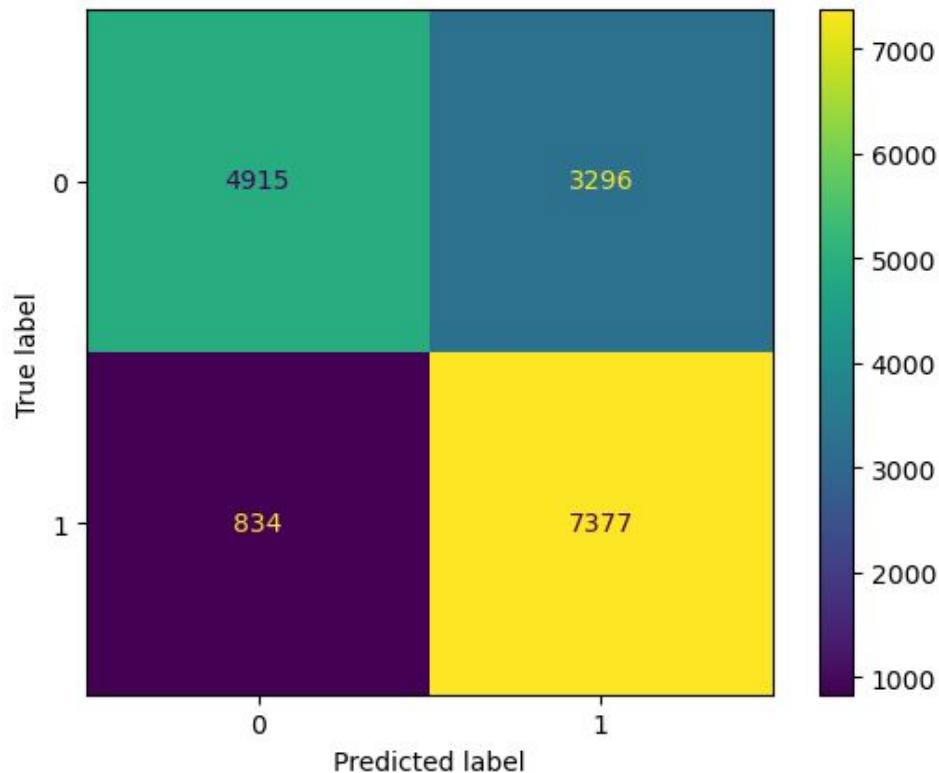
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_03)
```

```
0.8984289367921081
```

```
accuracy_score(y_test, preds_03)
```

```
0.7485080988917306
```



Entrenamiento con otros encodings

Red Neuronal:
Segundo modelo

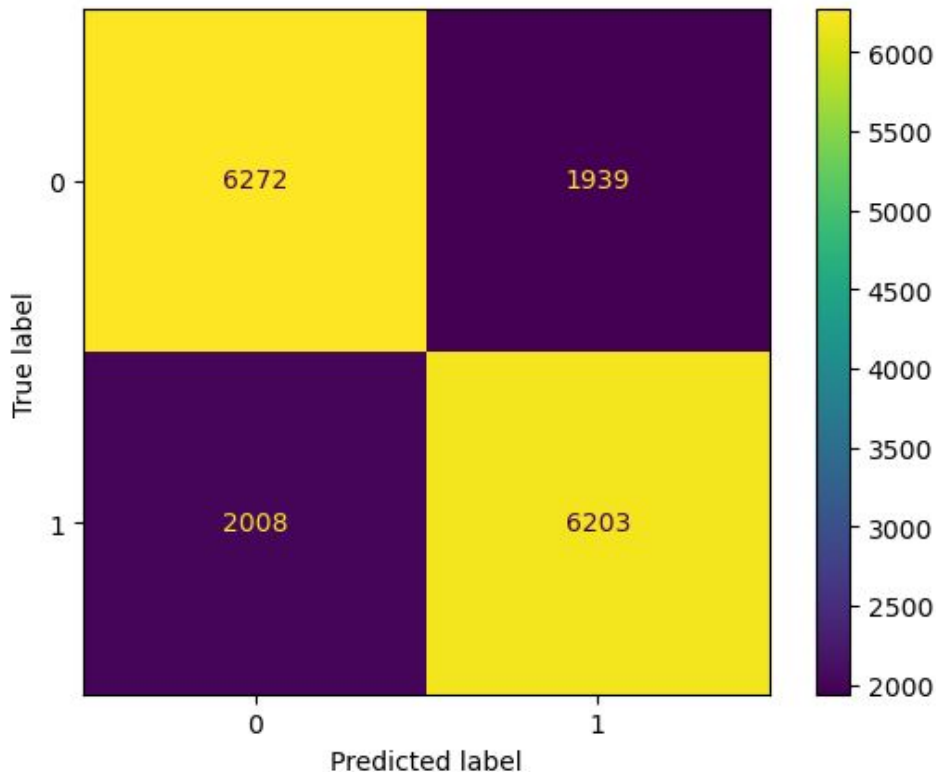
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_05)
```

```
0.7554500060893923
```

```
accuracy_score(y_test, preds_1_05)
```

```
0.7596516867616612
```



Entrenamiento con otros encodings

Red Neuronal:
Segundo modelo

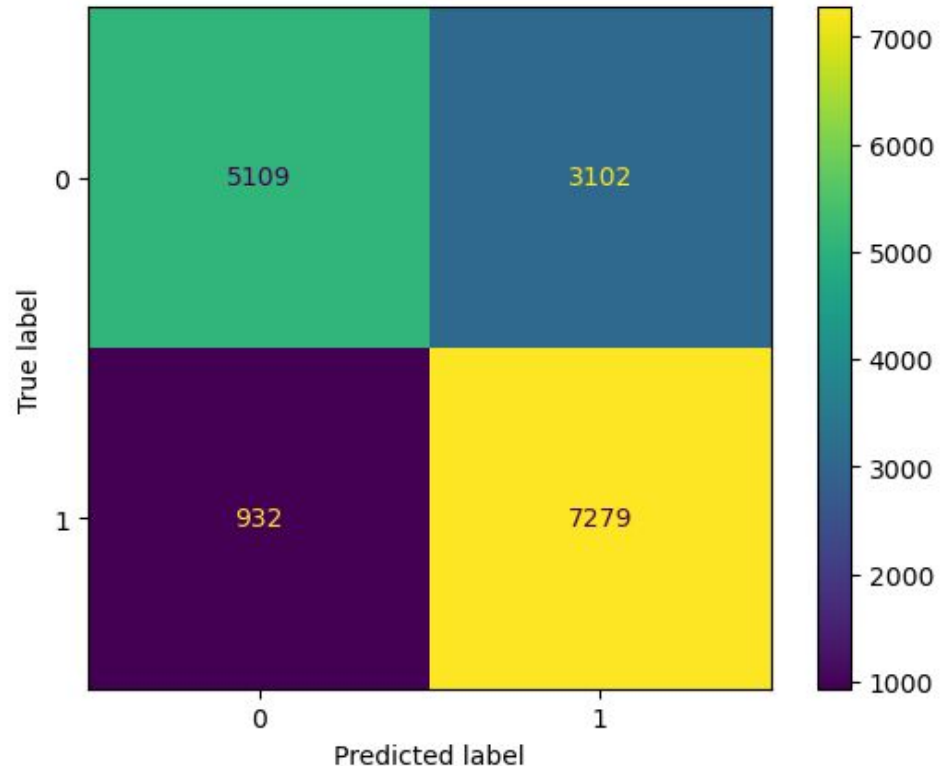
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_03)
```

0.8864937279259529

```
accuracy_score(y_test, preds_1_03)
```

0.7543539154792351



Usar otros encodings

Luego de este encoding seguimos teniendo más información ya que estos features eran continuos de 0 a 1

Por este motivo algunos algoritmos le seguían dando más importancia a estos features ya que podían sacar más información de ellos

Entonces ahora aplicamos la otra alternativa:

Usar otros encodings

Encoding binario:

SleepTime: $(5 \leq x \leq 12)$: 1 y el resto: 0

MentalHealth y PhysicalHealth: $(0 \leq x \leq 10)$: 1 y el resto: 0

BMI: $(18 \leq x \leq 30)$: 1 y el resto: 0

Veamos los resultados

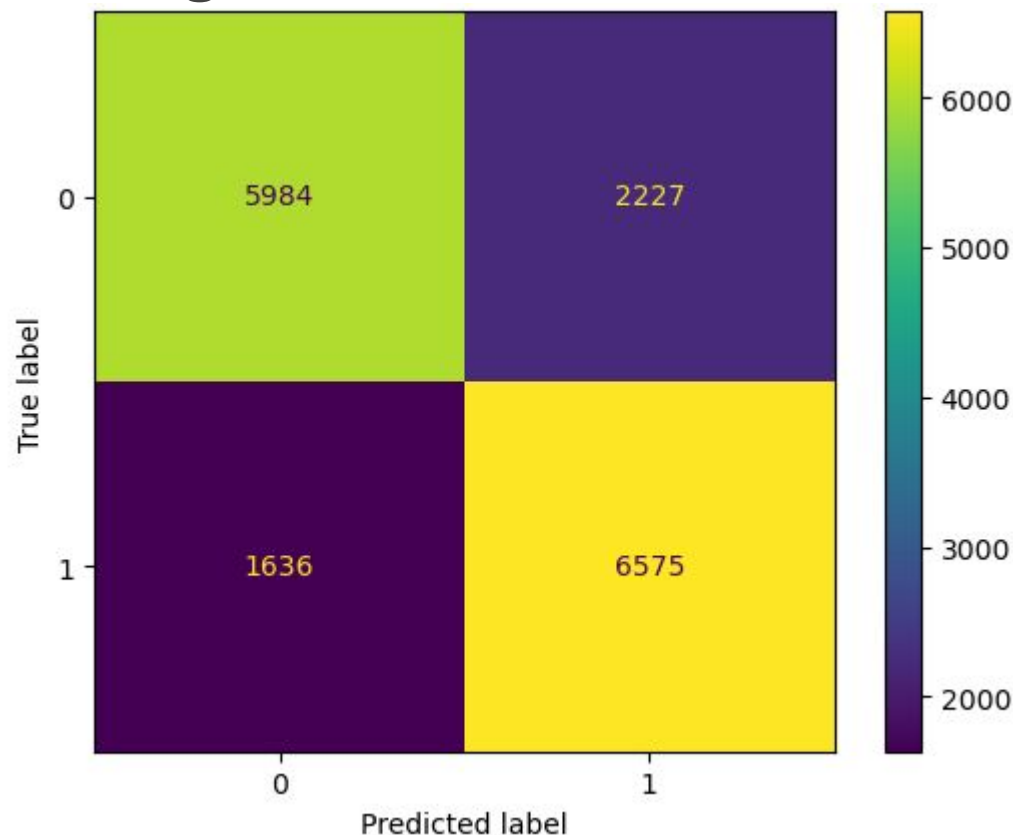
Entrenamiento con otros encodings

Modelo de LightGBM:

```
recall_score(y_test, preds)
```

0.8007550846425526

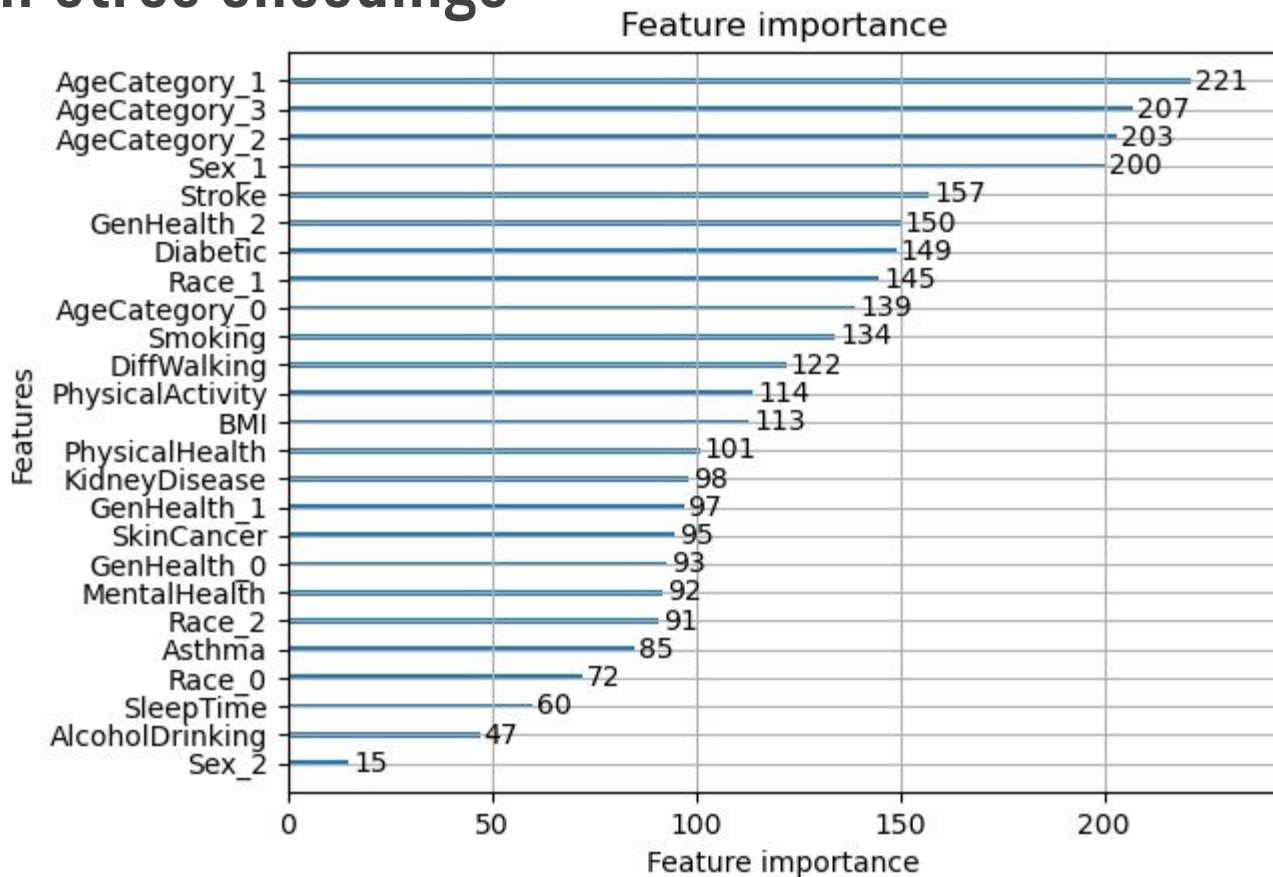
Encoding Binario



Entrenamiento con otros encodings

Modelo de
LightGBM:

Encoding Binario



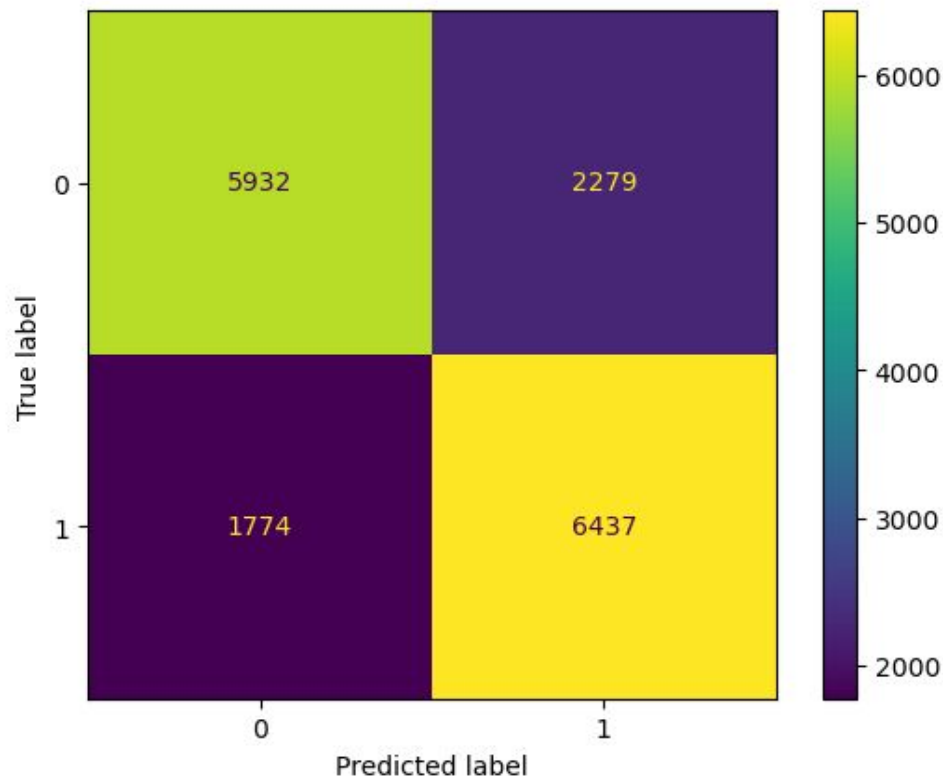
Entrenamiento con otros encodings

Modelo de
RandomForest:

```
recall_score(y_test, preds)
```

```
0.7839483619534771
```

Encoding Binario



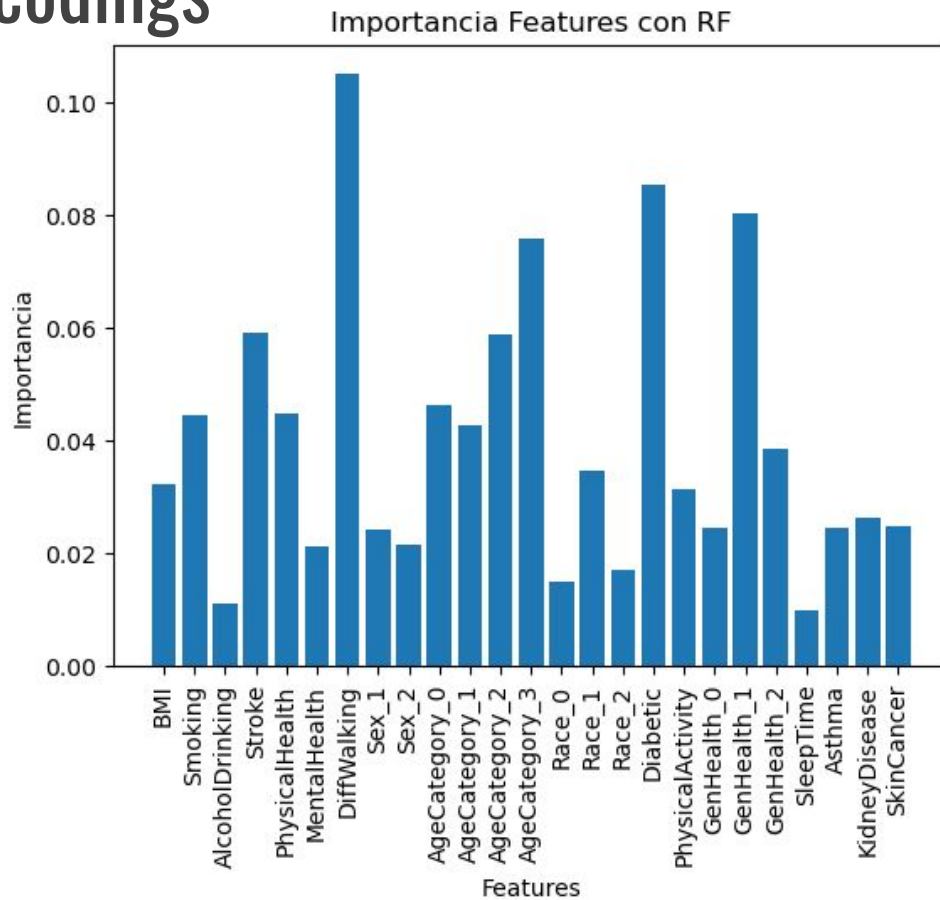
Entrenamiento con otros encodings

Modelo de

Random

Forest:

Encoding Binario



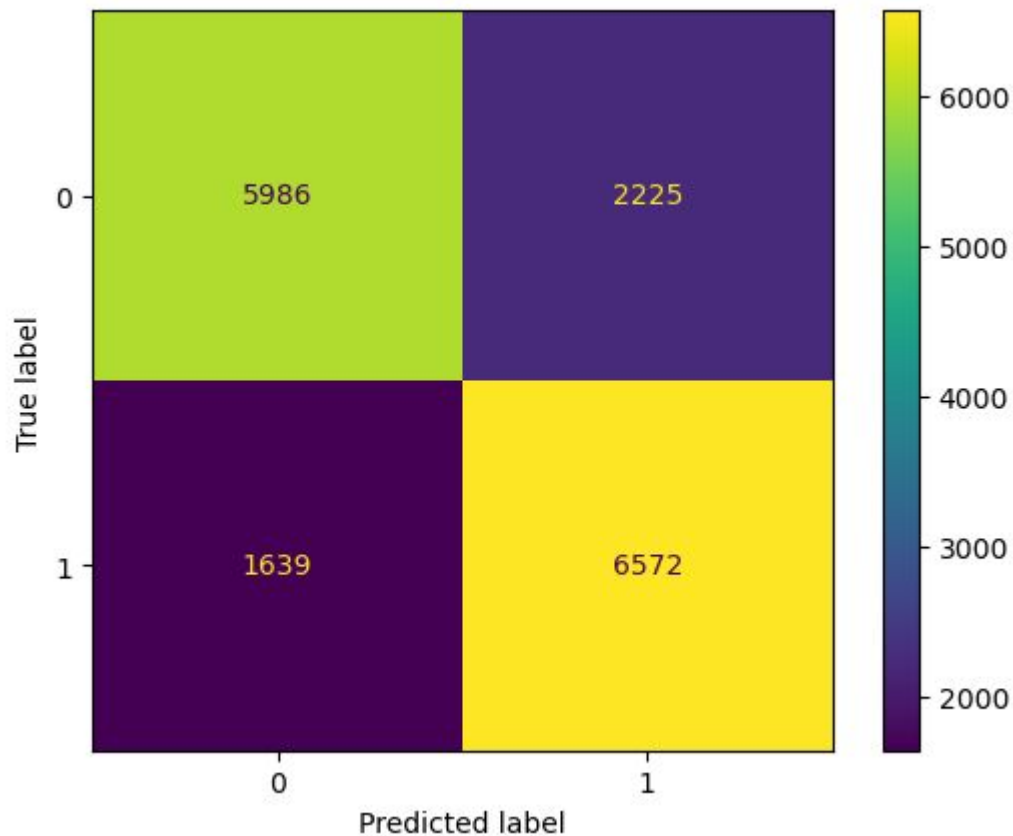
Entrenamiento con otros encodings

Modelo de XGBoost:

```
recall_score(y_test, preds)
```

```
0.8003897211058336
```

Encoding Binario



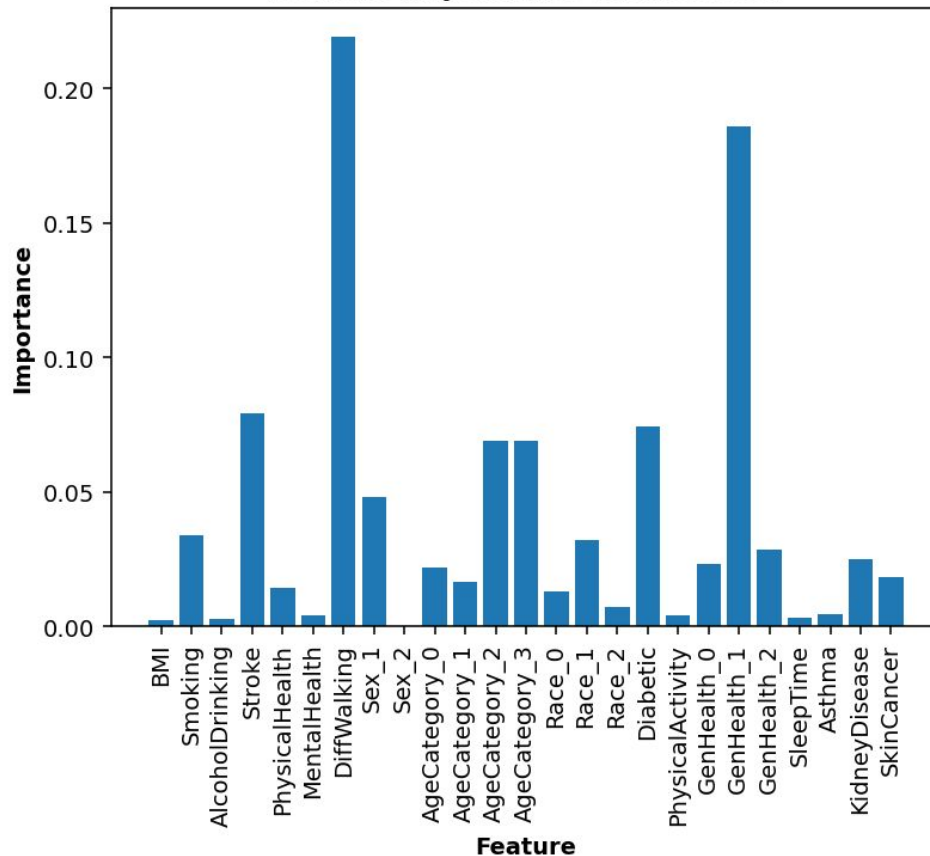
Entrenamiento con otros encodings

Modelo de

XGBoost:

Encoding Binario

Feature importance for XGBoost



Entrenamiento con otros encodings

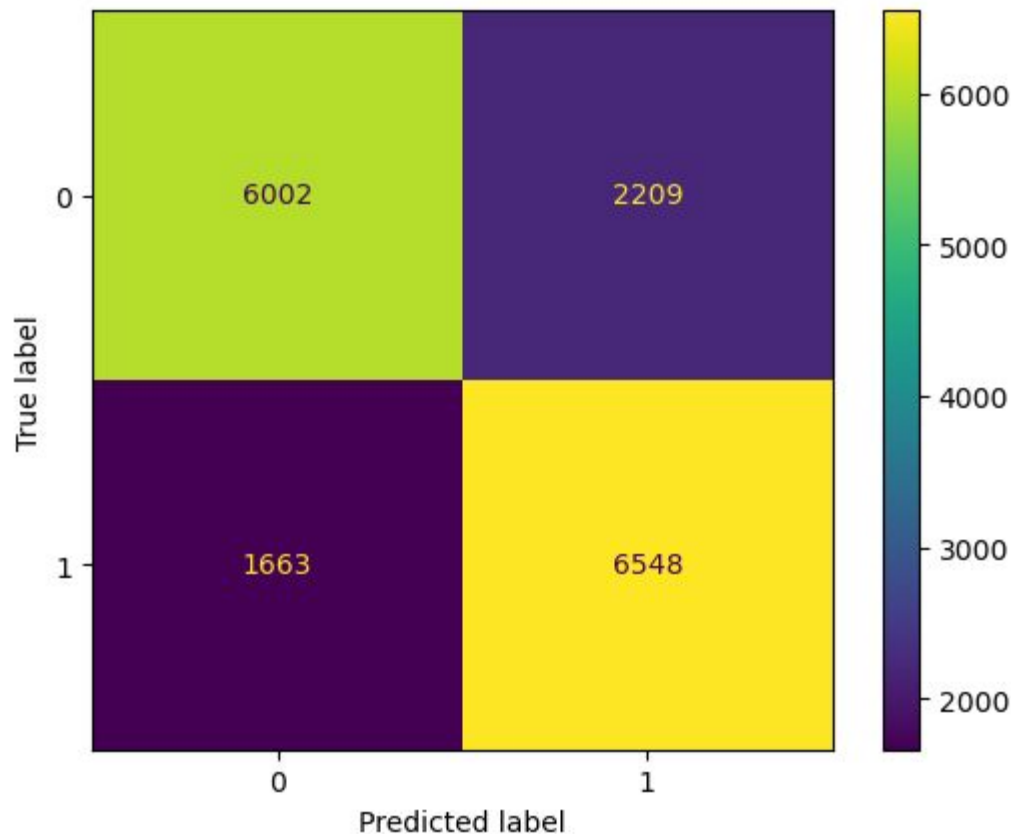
Modelo de

VotingClassifier:

```
recall_score(y_test, preds)
```

```
0.7974668128120813
```

Encoding Binario



Entrenamiento con otros encodings

Red Neuronal: Primer modelo

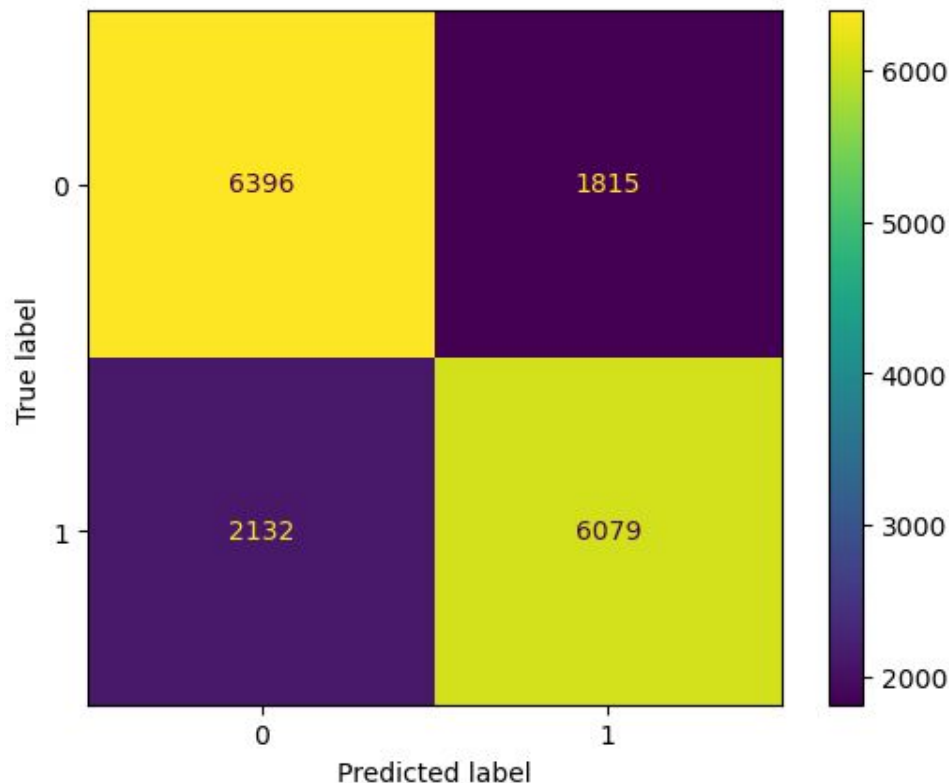
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_05)
```

```
0.7403483132383388
```

```
accuracy_score(y_test, preds_05)
```

```
0.7596516867616612
```



Entrenamiento con otros encodings

Red Neuronal: Primer modelo

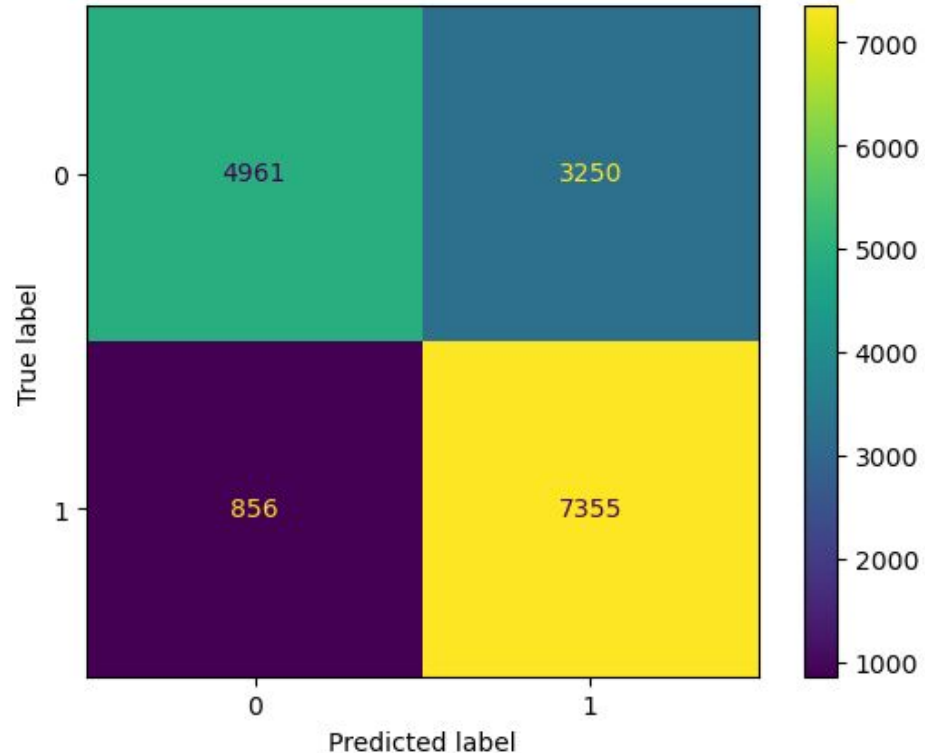
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_03)
```

```
0.8957496041895019
```

```
accuracy_score(y_test, preds_03)
```

```
0.7499695530386068
```



Entrenamiento con otros encodings

Red Neuronal:
Segundo modelo

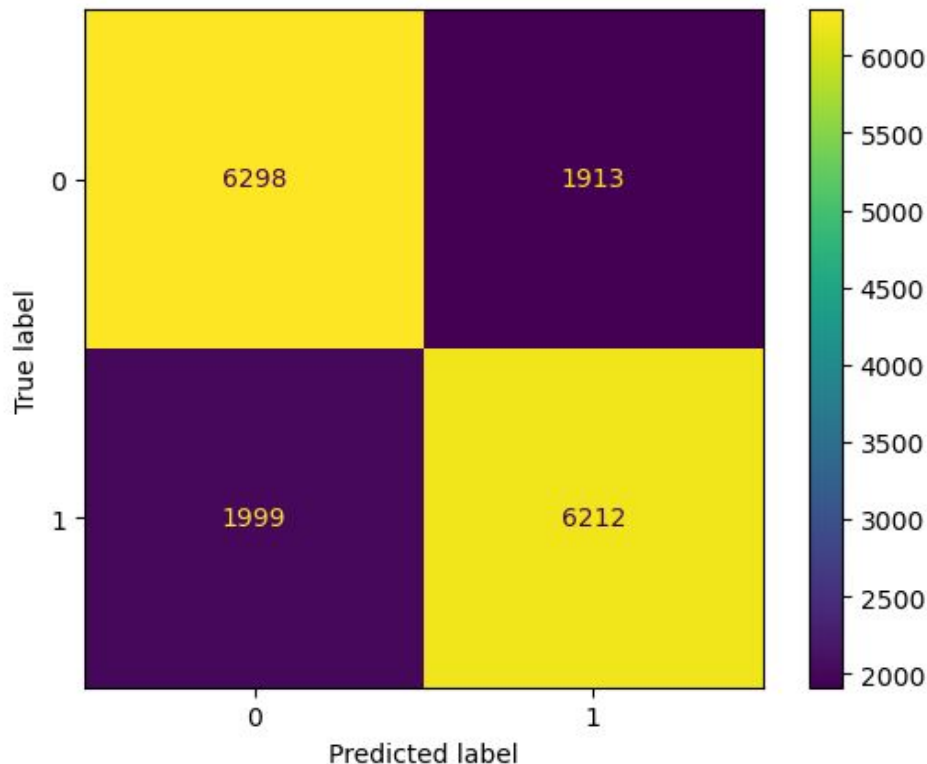
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_05)
```

```
0.7565460966995494
```

```
accuracy_score(y_test, preds_1_05)
```

```
0.7617829740591889
```



Entrenamiento con otros encodings

Red Neuronal:
Segundo modelo

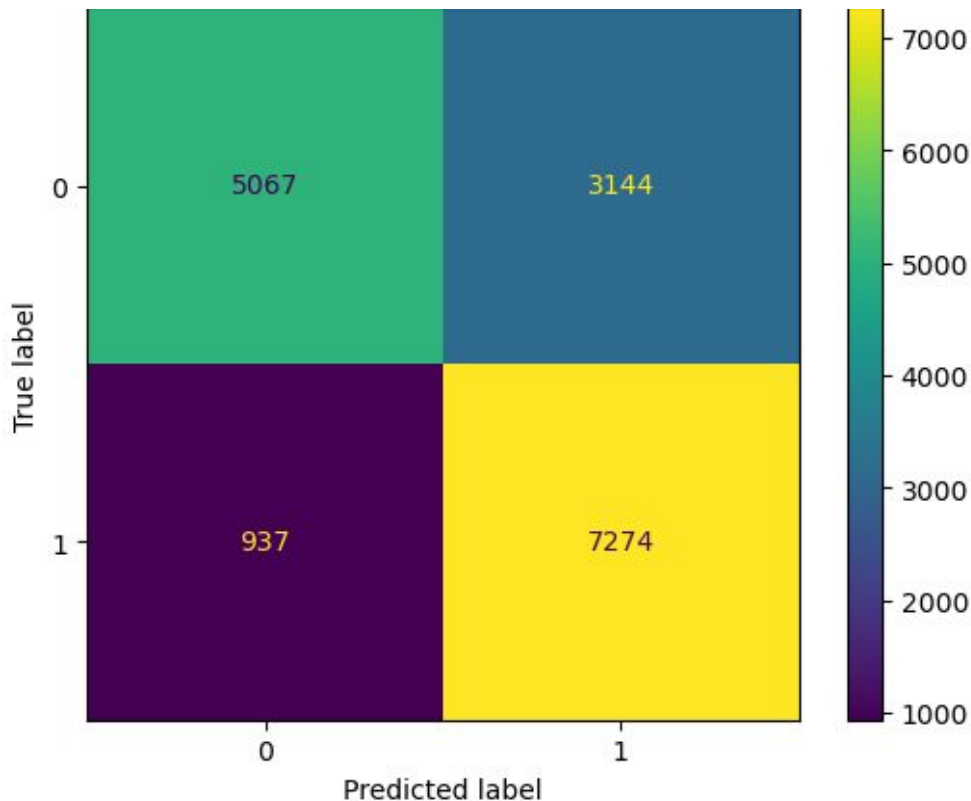
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_03)
```

```
0.885884788698088
```

```
accuracy_score(y_test, preds_1_03)
```

```
0.7514919011082694
```



Usar otros encodings

Ahora con este encoding vemos que la feature importance de los modelos cambia, priorizando otras features por sobre las encodeadas

Sin embargo, vemos que no producen un resultado muy diferente al que veníamos teniendo

Utilizar solo los features más importantes

Utilizar solo los features más importantes

Basados en los feature importance de los modelos utilizados y en la matriz de correlación con HeartDisease elegimos los 5 features más importantes

Estos son DiffWalking, PhysicalHealth, Diabetic, BMI, SleepTime

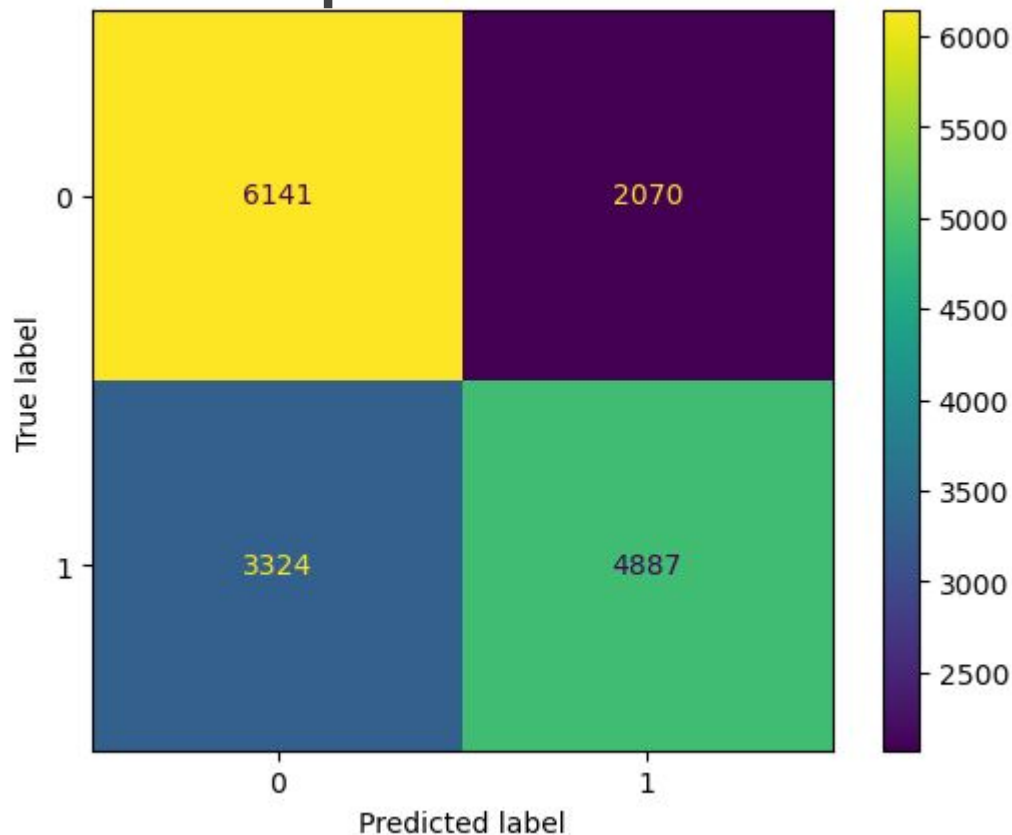
El encoding para los features numéricos será el primero utilizado (normalización estándar)

Entrenamiento con 5 features más importantes

Modelo de LightGBM:

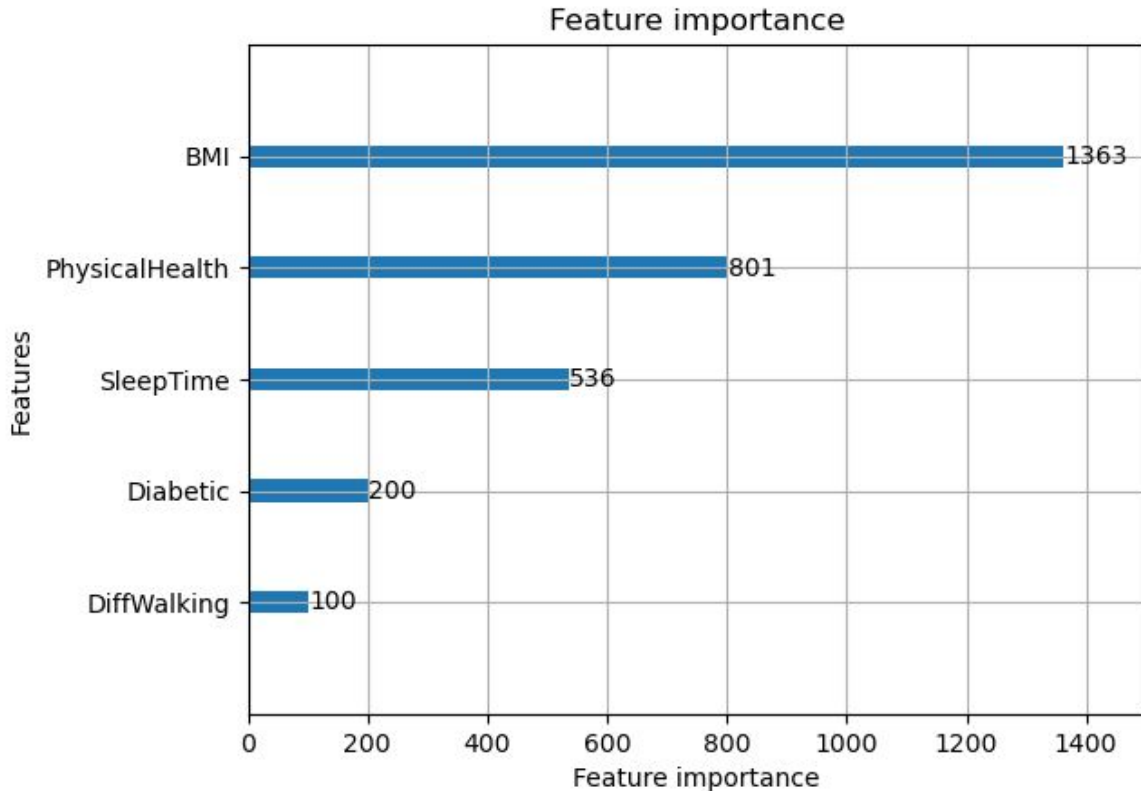
```
recall_score(y_test, preds)
```

0.5951772013153087



Entrenamiento con 5 features más importantes

Modelo de
LightGBM:

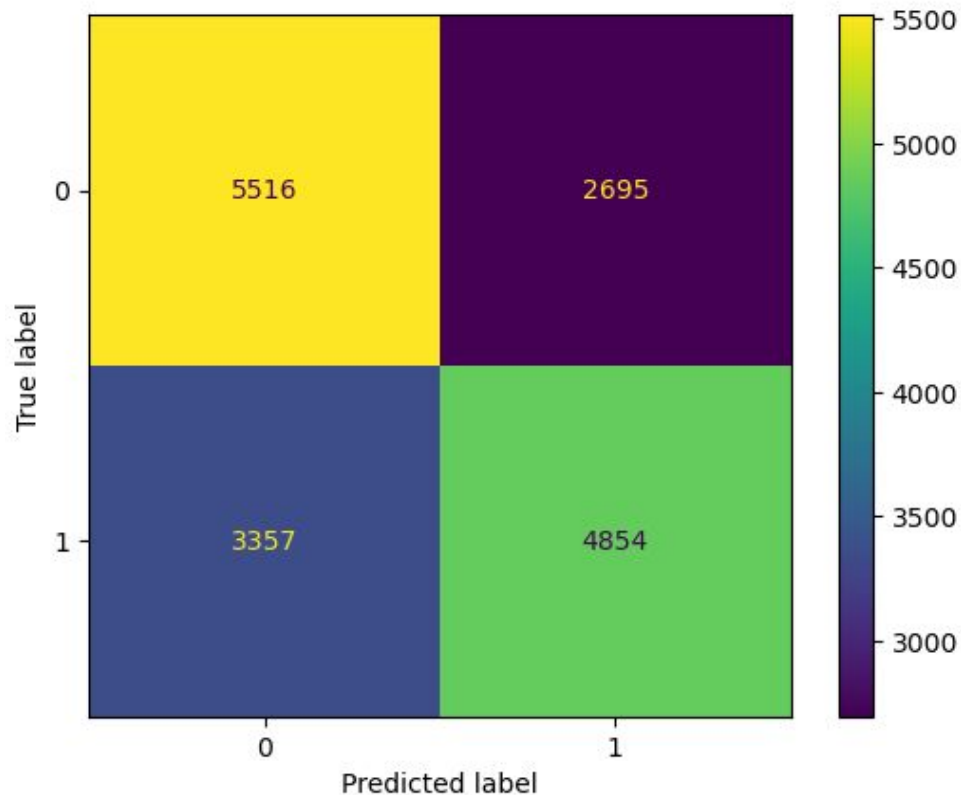


Entrenamiento con 5 features más importantes

Modelo de
RandomForest:

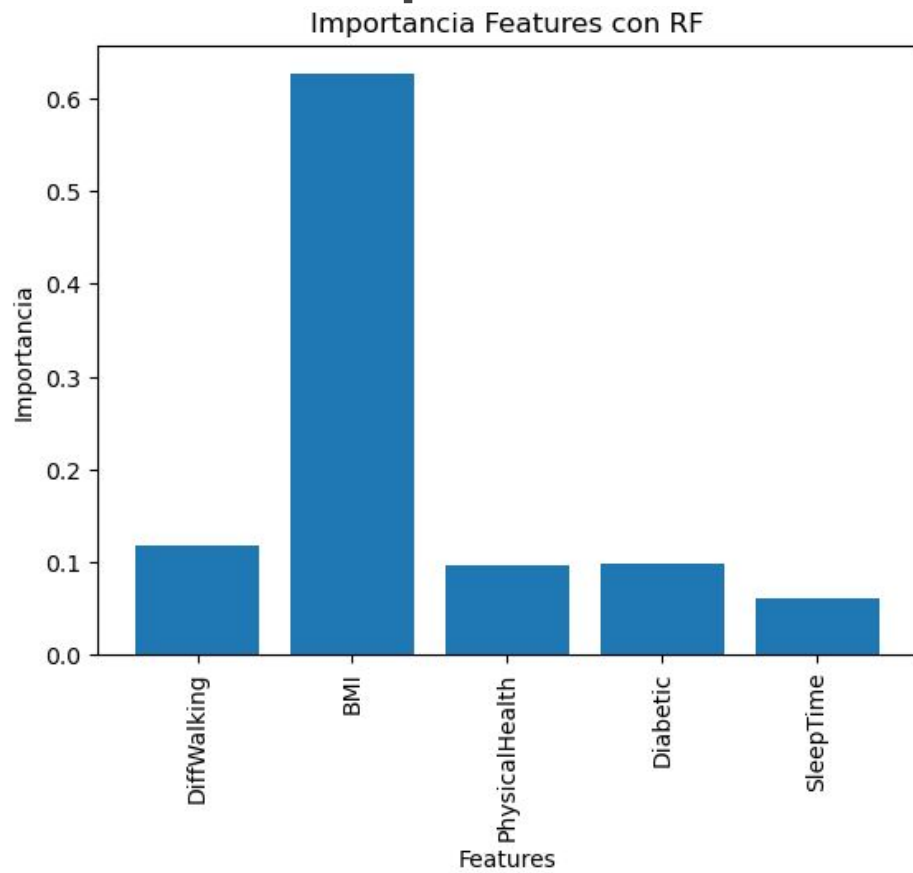
```
recall_score(y_test, preds)
```

```
0.5911582024113994
```



Entrenamiento con 5 features más importantes

Modelo de
Random
Forest:

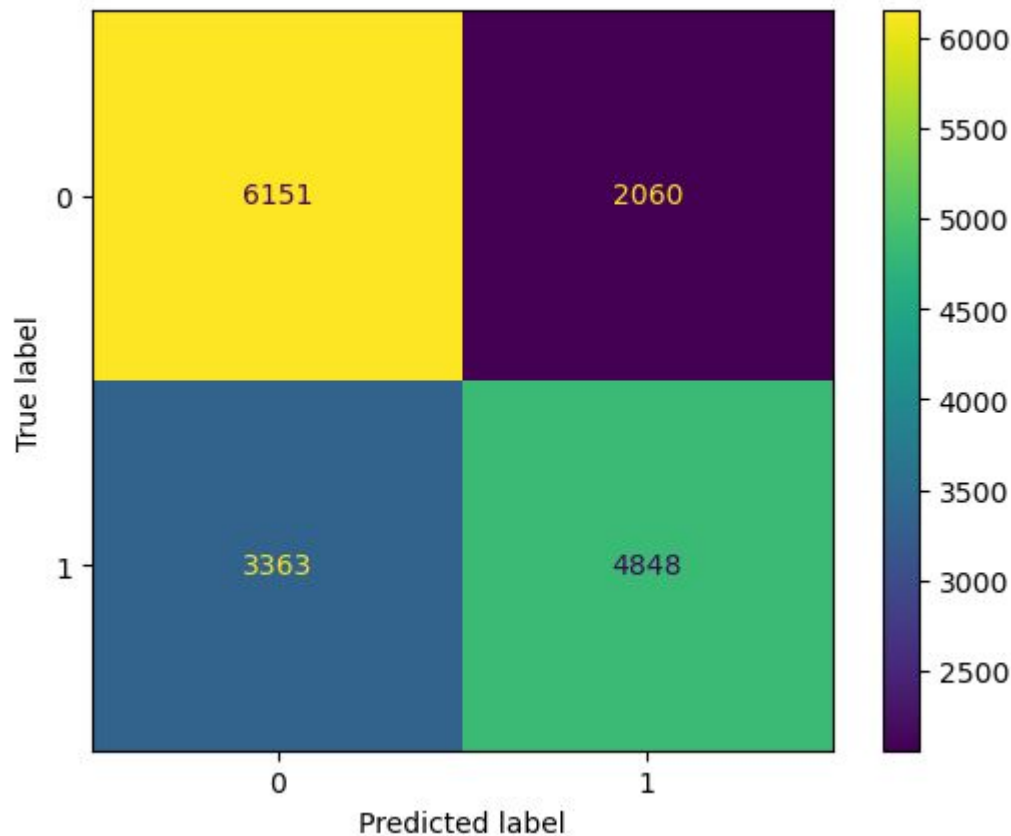


Entrenamiento con 5 features más importantes

Modelo de XGBoost:

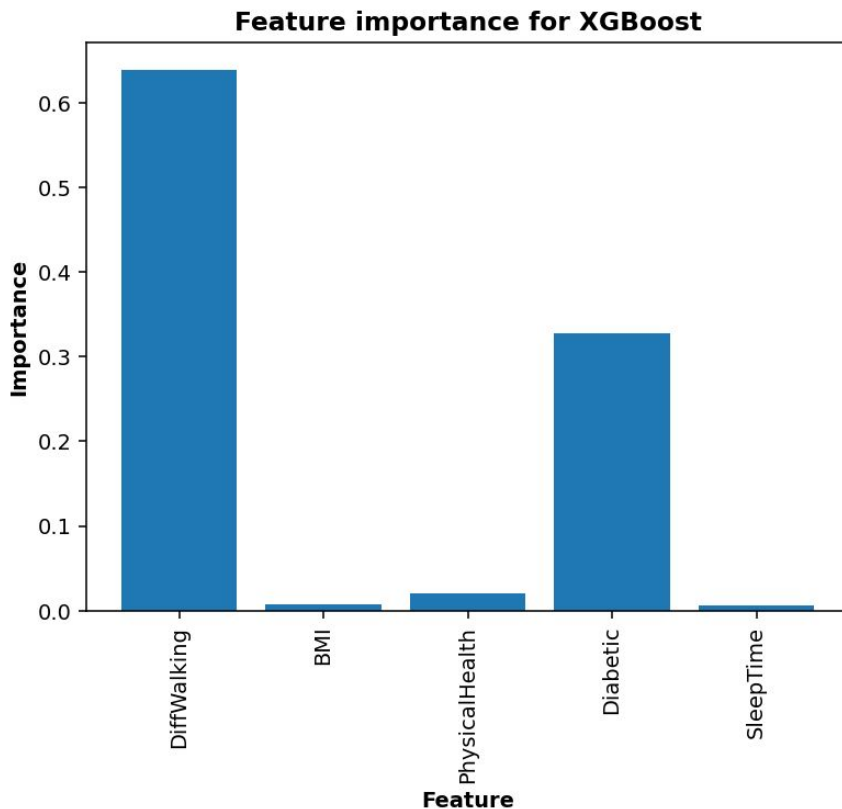
```
recall_score(y_test, preds)
```

0.5904274753379612



Entrenamiento con 5 features más importantes

Modelo de
XGBoost:



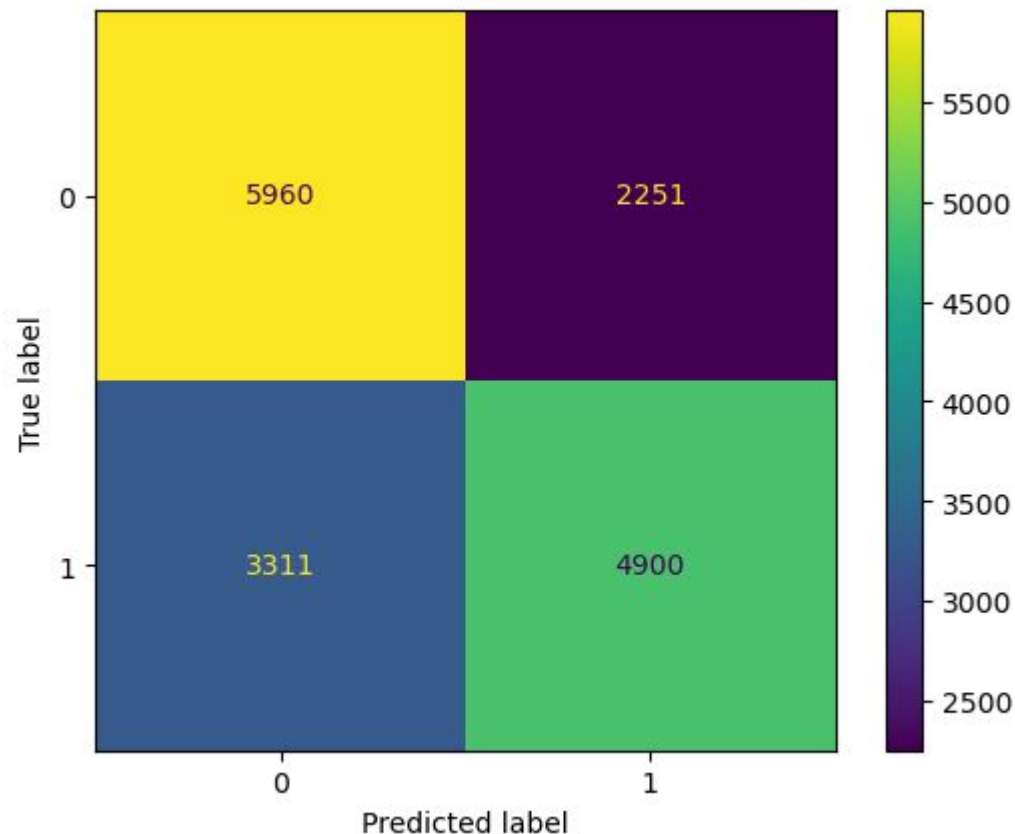
Entrenamiento con 5 features más importantes

Modelo de

VotingClassifier:

```
recall_score(y_test, preds)
```

```
0.5967604433077579
```



Entrenamiento con 5 features más importantes

Red Neuronal: Primer modelo

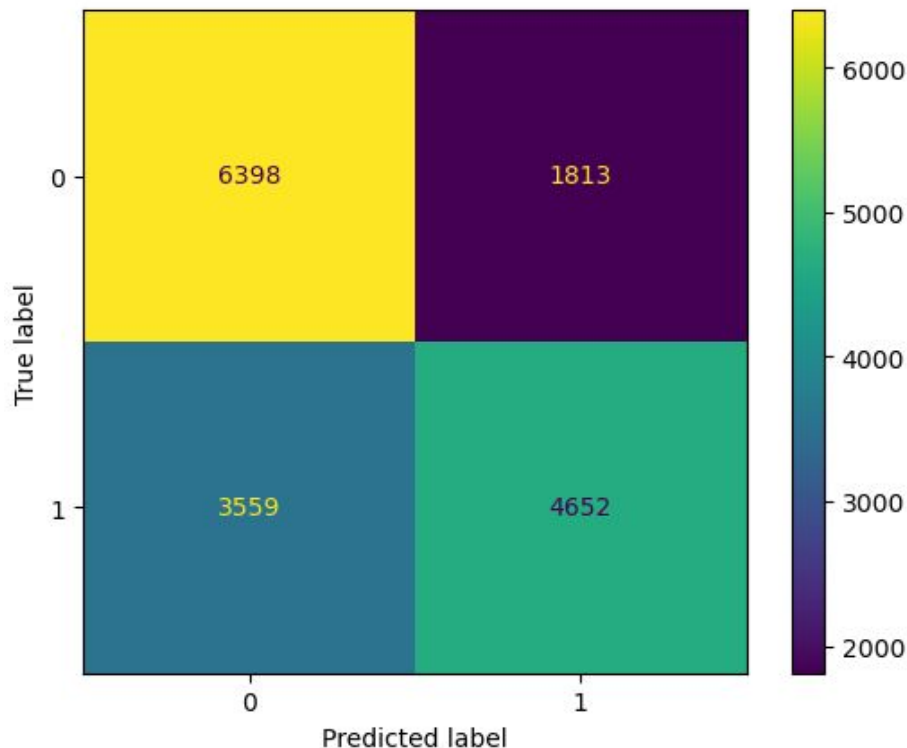
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_05)
```

```
0.566557057605651
```

```
accuracy_score(y_test, preds_05)
```

```
0.6728778467908902
```



Entrenamiento con 5 features más importantes

Red Neuronal: Primer modelo

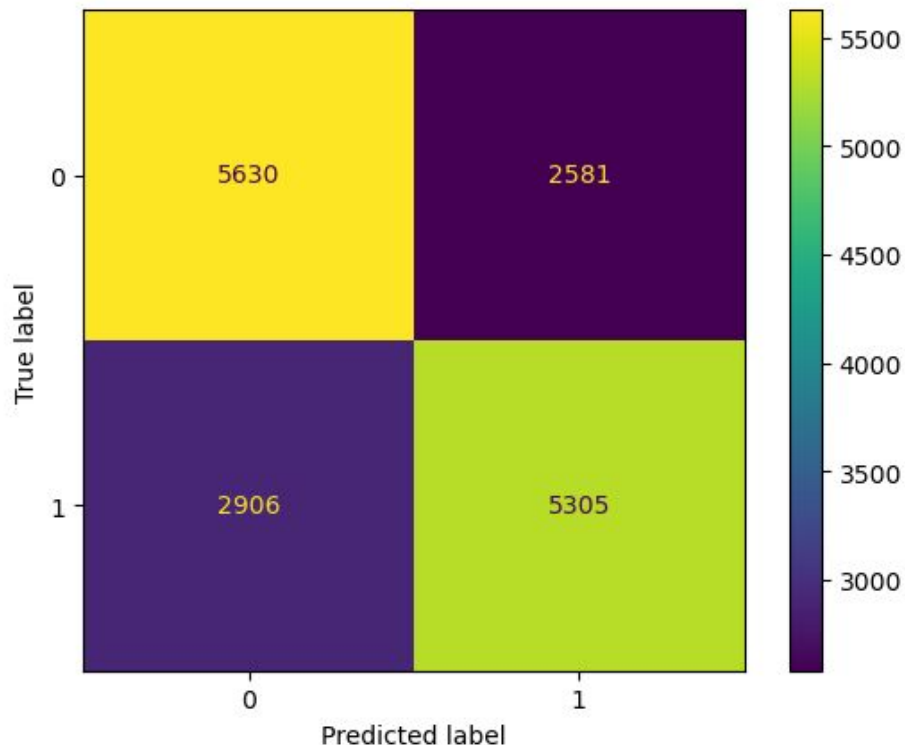
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_03)
```

```
0.6460845207648277
```

```
accuracy_score(y_test, preds_03)
```

```
0.6658750456704421
```



Entrenamiento con 5 features más importantes

Red Neuronal:
Segundo modelo

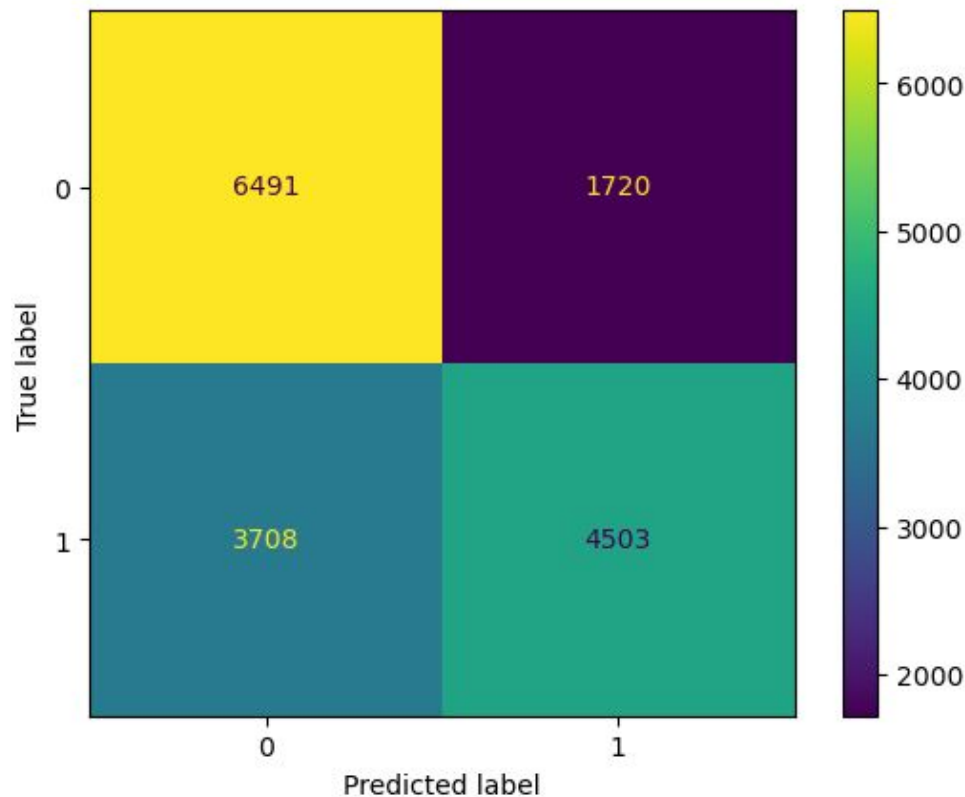
Si aplicamos un threshold en 0,5 (es decir de 0 a 0,5 clasificamos como 0 y de 0,5 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_05)
```

```
0.5484106686152722
```

```
accuracy_score(y_test, preds_1_05)
```

```
0.6694677871148459
```



Entrenamiento con 5 features más importantes

Red Neuronal:
Segundo modelo

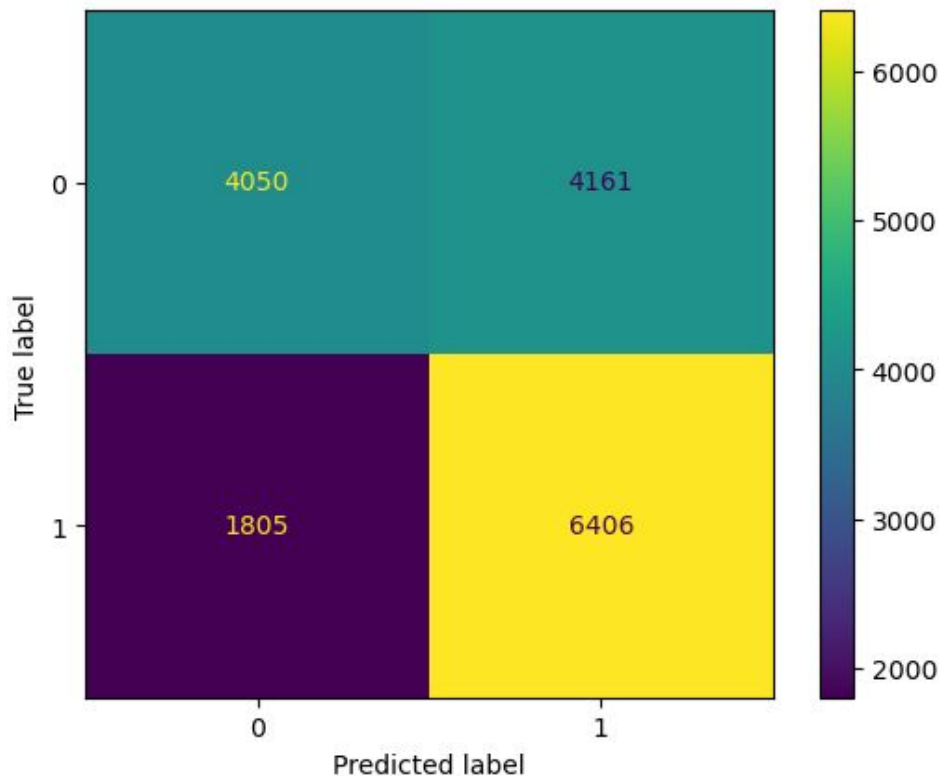
Si aplicamos un threshold en 0,3 (es decir de 0 a 0,3 clasificamos como 0 y de 0,3 a 1 clasificamos como 1)

```
recall_score(y_test, preds_1_03)
```

```
0.7801729387407137
```

```
accuracy_score(y_test, preds_1_03)
```

```
0.6367068566557058
```



Utilizar solo los features más importantes

Como vimos recién reducir nuestro DataSet a los 5 features más importantes empeora nuestros modelos

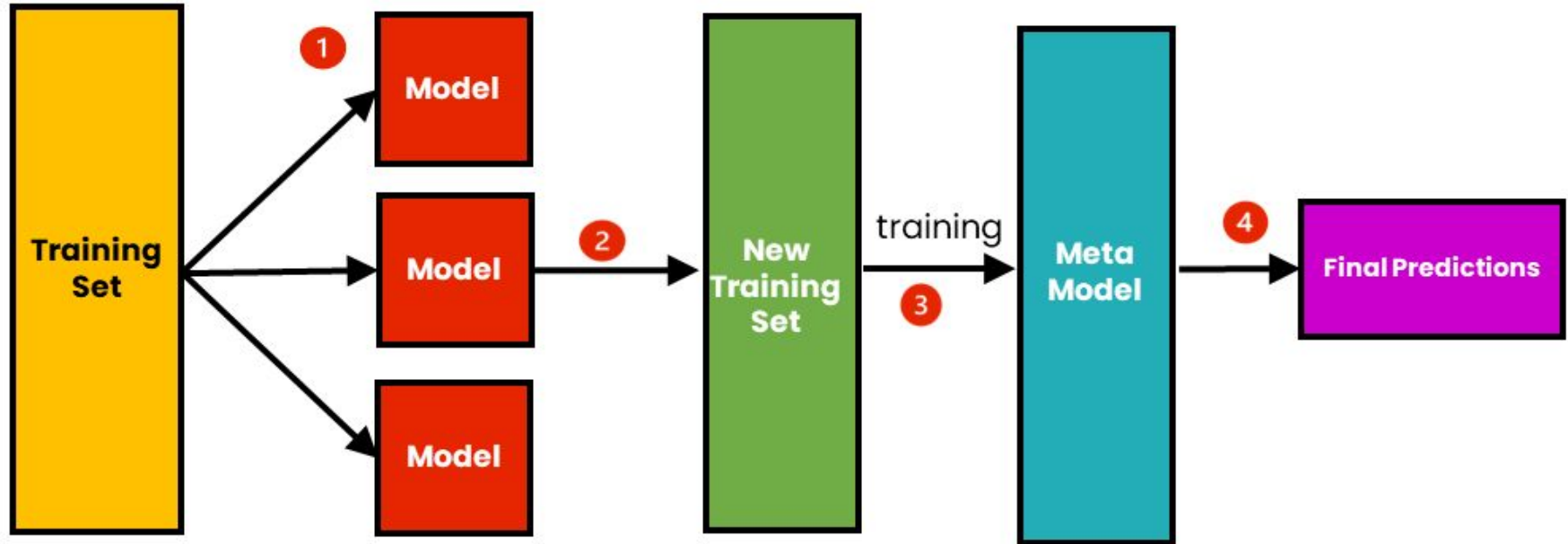
Como ventaja tenemos que podemos ahorrar gran cantidad de memoria, y mejorar la velocidad

Poniendo en la balanza las ventajas y desventajas, para nuestro problema, al estar tratando un problema de salud preferimos necesitar más datos y que sea más lento pero obtener un mejor recall, ya que es importante equivocarnos lo menos posible

Stacking: Encadenamiento de modelos

Stacking: Encadenamiento de modelos

The Process of Stacking



Stacking: Encadenamiento de modelos

En nuestro caso lo haremos de 3 formas distintas:

1. Usaremos LGBM, RandomForest, XGBoost para luego aplicar una regresión logística
2. LGBM -> RandomForest -> XGBoost
3. LGBM, RandomForest, XGBoost para luego aplicar una red neuronal

Entrenamiento con encadenamiento de modelos

Vamos con el primer modelo

LGBM, RandomForest, XGBoost para luego aplicar una regresión logística

Usaremos para esto el stacking classifier de sklearn

Documentación:

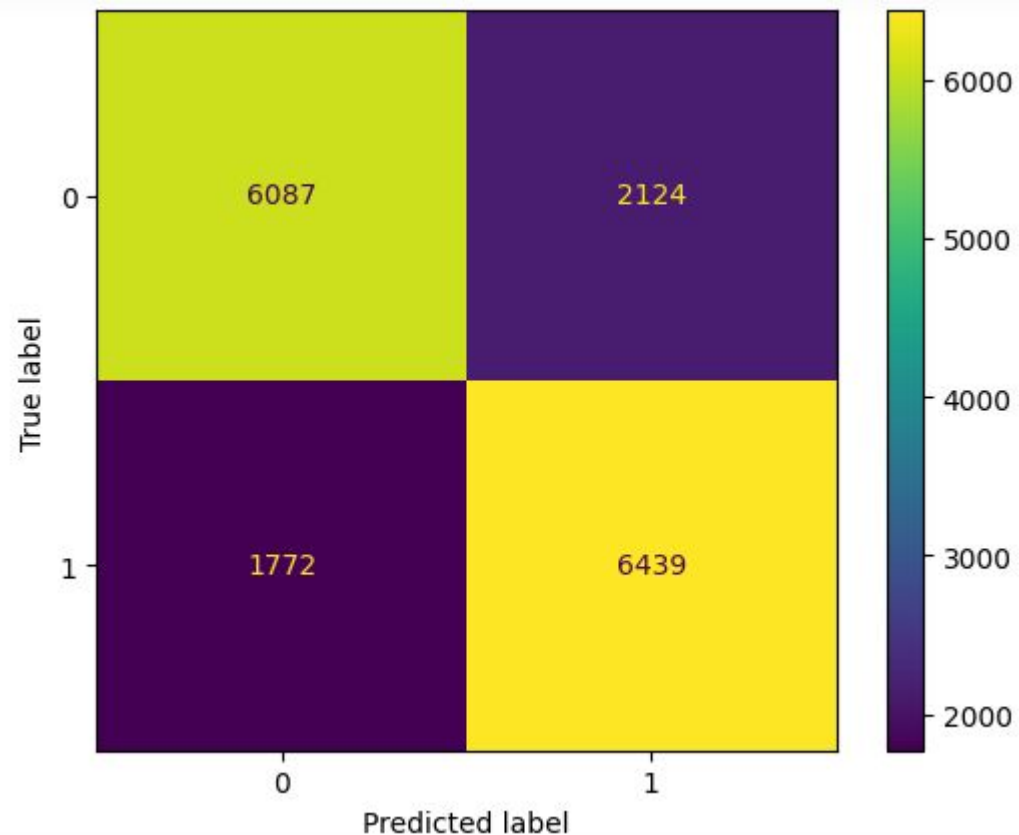
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html>

Entrenamiento con encadenamiento de modelos

Primer modelo:

```
recall_score(y_test, preds)
```

0.784191937644623



Entrenamiento con encadenamiento de modelos

Vamos con el segundo modelo

LGBM -> RandomForest -> XGBoost

Entrenaremos un modelo de LGBM, luego esas predicciones irán al dataset para entrenar un modelo de RandomForest, y por último, lo mismo para entrenar un modelo de XGBoost

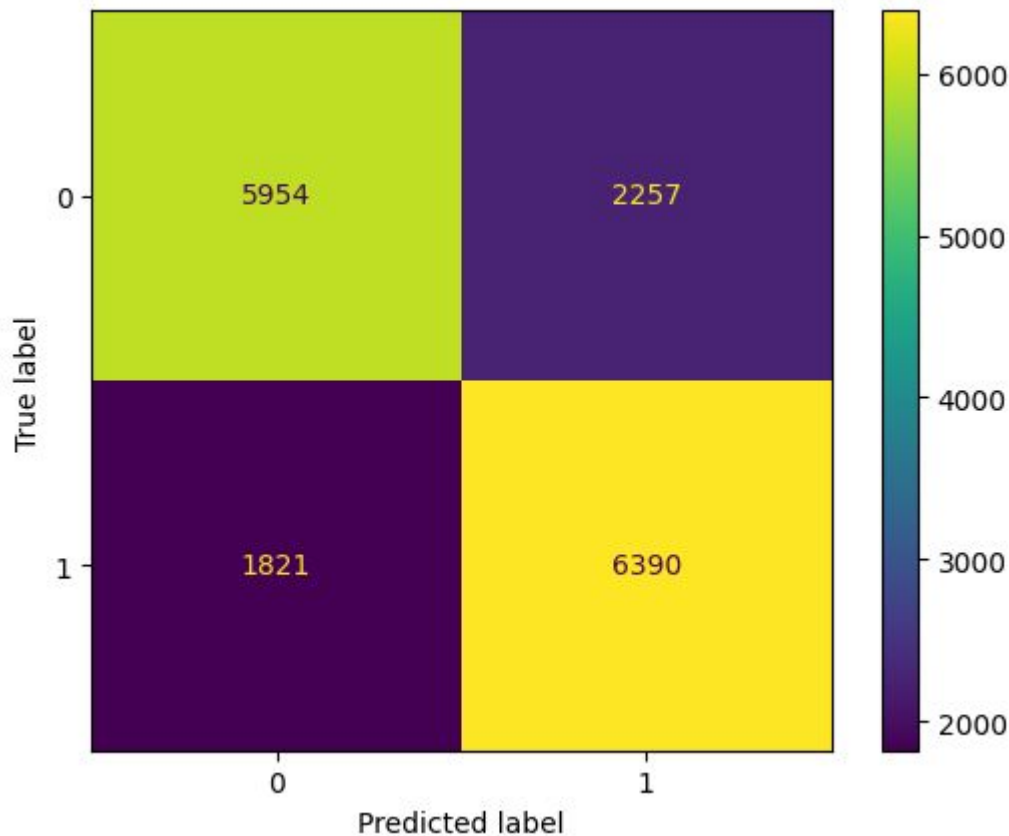
El orden fue elegido arbitrariamente

Entrenamiento con encadenamiento de modelos

Segundo modelo:

```
recall_score(y_test, preds)
```

```
0.7782243332115455
```



Entrenamiento con encadenamiento de modelos

Vamos con el último modelo

LGBM -> RandomForest -> XGBoost -> Red Neuronal

Es muy parecido al anterior modelo, pero le agregamos una red neuronal al final

El orden fue elegido arbitrariamente nuevamente

La red neuronal es la del primer modelo de red neuronal que usamos antes

Entrenamiento con encadenamiento de modelos

Último modelo:

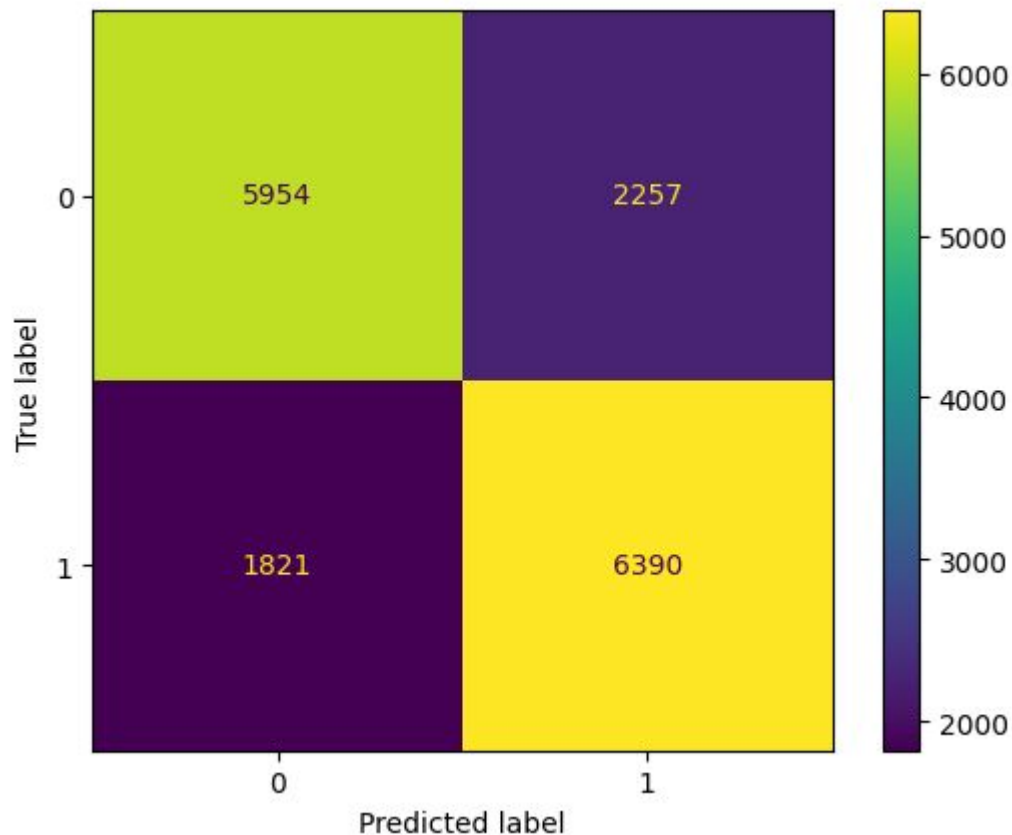
Con threshold 0.5

```
recall_score(y_test, preds_05)
```

0.7782243332115455

```
accuracy_score(y_test, preds_05)
```

0.7516745828766289



Entrenamiento con encadenamiento de modelos

Último modelo:

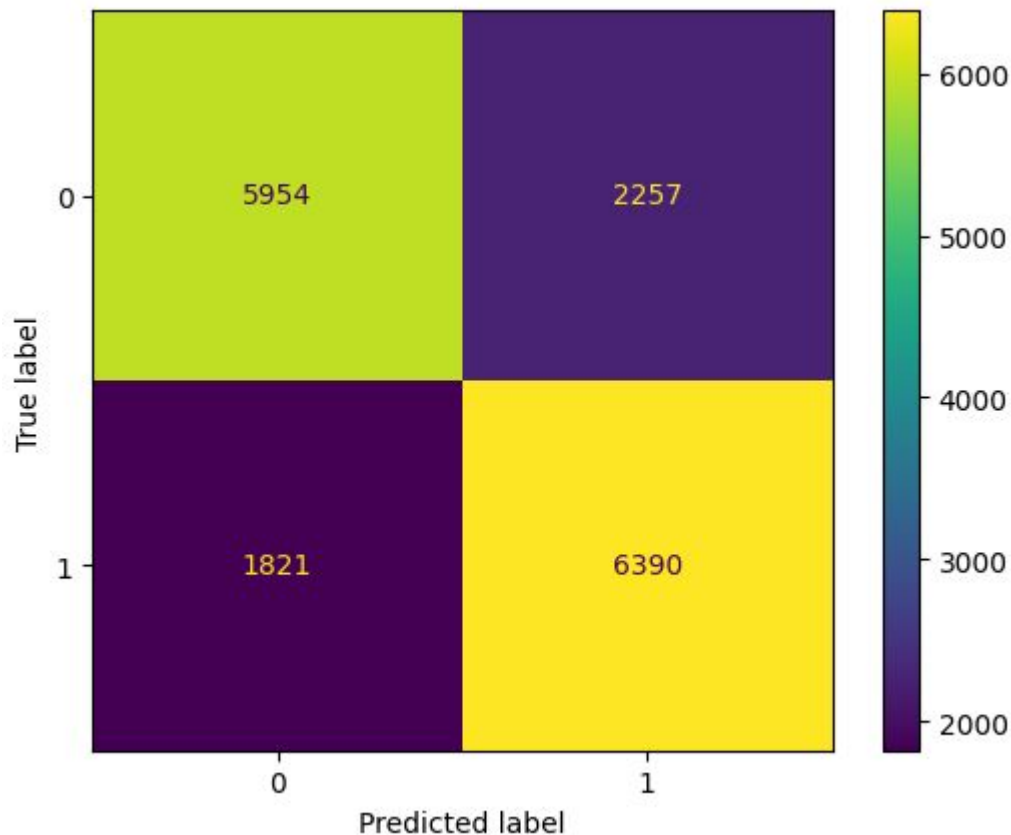
Con threshold 0.3

```
recall_score(y_test, preds_03)
```

0.7782243332115455

```
accuracy_score(y_test, preds_03)
```

0.7516745828766289



Entrenamiento con encadenamiento de modelos

Vemos que son exactamente iguales aunque cambiemos el threshold

Esto se da porque ahora, analizando un poco las predicciones, vemos que las probabilidades que arroja el modelo son muy cercanas al 0 o muy cercanas al 1

Comparación y conclusión

Comparación

Luego de entrenar diversos modelos, utilizando distintas técnicas y variando en las formas haremos una tabla comparativa basados en el recall para seleccionar el modelo que más nos convenza

Los modelos son LGBM, RandomForest, XGBoost, VotingClassifier, RedNeuronal 1 y 2 con threshold 0.5, RedNeuronal 1 y 2 con threshold 0.3

Comparación

Las técnicas utilizadas fueron:

- Entrenar con reducciones de UMAP para 3 dimensiones
- Entrenar sin reducciones
- Entrenar con encoding MinMax columnas numéricas
- Entrenar con encoding Binario columnas numéricas
- Entrenar con los 5 features más importantes
- Entrenar con encadenamiento de modelos

Comparación

	Con reduc	Sin reduc	MinMax	Binario	F + imp	Stacking
LGBM	0.79856	0.78808	0.78833	0.80075	0.59517	Primer Modelo: 0.78419
RF	0.78504	0.77822	0.77810	0.78394	0.59115	
XGB	0.79734	0.79052	0.79052	0.80038	0.59042	Segundo Modelo: 0.77822
VotingC	0.79905	0.79198	0.79222	0.79746	0.59676	
Red 0.5	0.73547	0.74838	0.74448	0.74034	0.56655	Tercer Modelo: 0.77822
Red_2 0.5	0.74448	0.71952	0.75545	0.75654	0.54841	
Red 0.3	0.89721	0.90220	0.89842	0.89574	0.64608	Tercer Modelo: 0.77822
Red_2 0.3	0.88113	0.88296	0.88649	0.88588	0.78017	

Conclusión

Luego de entrenar todos los distintos modelos y distintas técnicas utilizadas decidimos quedarnos con la primer Red Neuronal con threshold 0.3 sin reducciones

Es con el modelo al que mejor score llegamos, y al no tener reducciones tenemos menos memoria consumida y es menos costoso predecir (ya que antes teníamos que predecir también para el algoritmo de la reducción que era UMAP)

Conclusión

Pese a que haya un modelo un poquito superior en recall que el resto, vemos que esta diferencia es sutil

Casi todos los modelos llegan a un score parecido sea cual sea la técnica aplicada

Podemos concluir que llegamos a un máximo para cada modelo, donde hagamos lo que hagamos no va a cambiar mucho el resultado

Conclusión

El desarrollo de este trabajo fue un proceso iterativo incremental

Fuimos de a poco descubriendo un poco más sobre nuestros datos y aplicando distintos modelos según se nos venían a la mente

Más que nada esto aplica a la segunda parte del trabajo, donde se nos iban ocurriendo distintos encodings, distintas formas de combinar los algoritmos, etc.

Datos y Notebooks

Por último

Datos utilizados:

<https://www.kaggle.com/datasets/kamilpytlak/personal-key-indicators-of-heart-disease>

Desarrollo del trabajo:

<https://github.com/pedrogrin/TP-IA-FIUBA/upload/main>

Muchas gracias por ver!