Checkpoint 3 - Grupo 03

Introducción

En este checkpoint no transformamos más el dataset (ni el de entrenamiento ni el de testeo): trabajamos con los datos "*as-is*".
Aplicamos *cross validation* donde nos fue posible, priorizando búsqueda por *Grid Search*; salvo quizá en el caso del **XGBoost**: los parámetros eran tantos que se volvió inviable. Para esos casos, aprovechamos que tienen buen rendimiento y simplemente usamos *Random Search* con muchas combinaciones.
La única "modificación" que se hizo, fue una copia de los *splits* de entrenamiento/testeo. Esta copia fue normalizada con *StandardScaler()*, que demostró mejorar predicciones para la **Support Vector Machine**.

Construcción del modelo

Optimizamos con *Cross Validation* donde nos fue posible.

K-Nearest Neighbors (KNN):
Optimizamos *n_neighbors*, *weights*, *algorithm* y *metric*.

Support Vector Machine (SVM):
Optimizamos *kernel*, *gamma* y *coef0*.

Random Forest (RF):
Optimizamos *criterion*, *min_samples_leaf*, *min_samples_split* y *n_estimators*.

eXtreme Gradient Boosting (XGBoost):
Optimizamos *n_estimators*, *max_depth*, *max_leaves*, *gamma*, *reg_alpha*, *reg_lambda* y *learning_rate*.

Ensamble Híbrido (Voting):
Acá tuvimos preferencia en *voting=**hard*** en vez de *voting=**soft***. Utilizamos los mejores modelos de los clasificadores con mejor rendimiento: RF y XGBoost. Esto es puesto que para varias iteraciones era más rápido y son además los de mejor puntaje.

Ensamble Híbrido (Stacking):
También utilizamos los mejores modelos de RF y XGBoost. El "meta-modelo" fue un *LogisticRegressionCV()*.

Cuadro de Resultados

Modelo	F1-Score	Precision	Recall	Accuracy	Kaggle	**Hiperparámetros**
KNN	0.7904957	0.7661821	0.8164031	0.7805342	0.76599	**n_neighbors=**25 **metric="**minkowski" **p=**1 **weights=**"distance"
SVM	0.8546662	0.8492996	0.8601011	0.8516497	0.84444	**kernel=**"poly" **coef0=**8.5
Random Forest	0.8824841	0.8940763	0.8711887	0.8823286	0.88087	**criterion=**"entropy" **min_samples_leaf=**1 **min_samples_split=**3 **n_estimators=**100 **random_state=**1
XGBoost	0.8801817	0.8458476	0.8845589	0.8778632	0.86179	**random_state=**1 **n_estimators=**72 **max_leaves=**0 **max_depth=**19 **learning_rate=**0.2285714 **gamma=**3.6734694
Voting	0.8852862	0.8891447	0.8814609	0.8841479	0.87462	**estimators=**[RF, XGB] **voting=**"soft"
Stacking	0.8870505	0.8912113	0.8829284	0.8859671	0.87612	**estimators=**[RF, XGB] **final_estimator=***LogisticRegressionCV()* **cv=***KFold(**n_splits=**5)*

- **KNN** calcula los "vecinos" más cercanos (datos con valores similares) para ponderar en qué grupo clasificarlos.
- **SVM** se especializa en problemas no linealmente separables: utiliza funciones *kernel* para lograr separar conjuntos de datos elevándolos a una dimensión mayor.
- **RF** hace uso de muchos árboles con *features* elegidas al azar cada vez, y luego promedia el rendimiento de cada uno.
- **XGBoost**, una variante el *Gradient Boosting*, va creando árboles progresivamente, y con un coeficiente extra (llamado *learning rate*) va sumando de a poco los aportes de dichos árboles, donde cada uno se crea en base a errores del anterior.
- Ensamble híbrido tipo **Voting** utiliza muchos modelos y, mediante votación por mayoría, decide sobre el valor final de una predicción.
- Ensamble híbrido tipo **Stacking** también utiliza varios modelos, y además un "meta modelo" extra que maneja los modelos de la otra capa para utilizarlos según lo vea mejor.

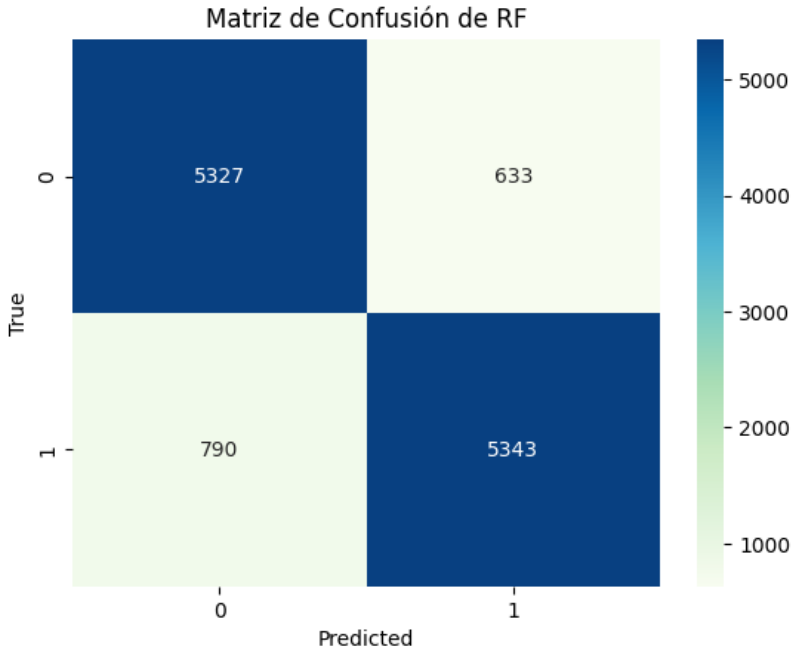
Matriz de Confusion

Nuestro mejor modelo fue una iteración del *Random Forest*.

La iteración en particular se hizo con *splits*:

- Train: 80%
- Test: 20%

Y la semilla para generar dichos *splits* era **3**.



Tareas Realizadas

Integrante	Tarea
Franco Lighterman Reismann	Construcción de Modelos/Ensambles Armado de Informe
Marcos García Neira	Búsqueda de Hiperparámetros Armado de Informe
Martín Andrés Maddalena	Armado de Informe