Checkpoint 3 - Grupo 16 GPWin

Introducción

Para la realización del presente trabajo, se partió del dataset depurado obtenido en el "checkpoint1". Se utilizó este dataset para todos los modelos, salvo para el caso de SVM donde se realizó un trabajo de normalización estándar. Para un caso de XGBoost de hizo una optimización bayesiana de hiperparámetros.

Construcción del modelo

Para el modelo **KNN**, se procedió a optimizar hiperparametros mediante random search. Dentro de los parámetros que se buscaron optimizar se encuentran la cantidad de puntos vecinos (dándole un rango entre 2-20), los pesos (weights), el algoritmo utilizado y la métrica. Para los pesos se consideraron dos variaciones (uniforme y por distancia), es decir en un caso el hiperparámetro otorga a todos los vecinos igual importancia mientras que en el otro caso pondera, otorgando más importancia a aquellos vecinos más cercanos. Para el caso del algoritmo se tuvieron en cuenta 'ball_tree', 'kd_tree', 'brute', especialmente el primero dado que debiera ser más eficiente para datos con alta dimensionalidad como es nuestro caso. Por último en las métricas de distancia se consideraron la euclidiana ,manhattan','chebyshev.

En el caso del modelo **RF** (random forest) , se realizó la optimización tanto con GridSearch como con RandomSearch, sin embargo por los tiempos de ejecución y la calidad de los resultados, nos decantamos por el primero.

La optimización de los hiperparametros en este caso se llevó a cabo sólo para la métrica deseada F1-Score.

Dentro de los parámetros del randomForest, se consideraron que las variables en cada nodo fueran la totalidad de las variables disponibles. (se buscó dejarlo de esta forma automáticamente pero al no poder, se utilizó un valor arbitrario muy elevado).

A su vez, se seteo oob_score=True para permitirle al random forest que calcule la calidad del modelo utilizando los datos "out of bag" es decir aquellos que no se utilizaron para cada árbol particular (si bien esto es recomendado para muestras pequeñas) nosotros decidimos utilizarlo más allá de contar con un conjunto de datos de test.

Respecto al GridSerach, se buscó la optimización del modelo con los siguientes hiperparámetros: "criterion", permitiendo "entropía" "gini" "log-loss", una cantidad máxima de árboles "n-iterator" que podía tomar ciertos valores discretos comenzando en 1 y teniendo como máximo 300.

En cuanto al "min_samples_leaf" dado que consideramos que el set de observaciones que tenemos es bastante grande, podiamos utilizar valores de este

hiperparametro más bajos, sin riesgo a realizar overfiting y logrando así una mejor respuesta a los datos de entrada. Se consideraron en GS para este parámetro los valores 1/5/10. Para el último hiperparam, "min_sample_split" se dieron valores discretos entre 2 y 32, sabiendo que números bajos tienden a ajustar mejor a los valores de entrada pero podrían ocasionar nodos muy puros y por tanto el sobre entrenamiento del modelo.

Para la construcción del modelo del **SVM**, primero normalizamos los datos con un escalador estándar. Luego entrenamos un SVM con kernel polinómico de grado 3 con parámetros arbitrarios C=1, gamma=1 y coef0=0.1. Hicimos una prueba con un SVM con kernel radial y los parámetros default que trae, que son C=1, gamma=auto. El mejor modelo lo obtuvimos haciendo una optimización de los hiperparámetros C y gamma con el kernel radial fijo, corriendo una búsqueda exhaustiva de 100 combinaciones. El conjunto de hiperparámetros que optimizó mejor la métrica f1 fue una regularización C=6 y el parámetro de kernel gamma=0.01.

Para la construcción del modelo de **XGBoost** partimos los datos en train y test, sin normalizarlos e hicimos una prueba con los hiperparámetros default. Luego hicimos una optimización de los hiperparámetros eta (learning_rate), la profundidad máxima de cada árbol (max_depth), el minimum loss para partir una hoja (min_split_loss), el parámetro de regularización L1 alpha y el parámetro de regularización L2 lambda, ejecutando 1000 combinaciones distintas de ellos de forma aleatoria optimizando la métrica f1. Finalmente, el mejor modelo lo obtuvimos optimizando los parámetros mencionados anteriormente además de otros (min_child_weight, n_estimators, subsample, colsample_bytree), estableciendo la métrica de stop de crecimiento de crecimiento de los árboles eval_metric en mlogloss y corriendo las sucesivas optimizaciones de forma bayesiana (o sea que se buscan los hiperparámetros a medir en base a los hiperparámetros usados anteriormente, esto se hace siguiendo un modelo probabilístico que estima la probabilidad de lograr un resultado deseado con la combinación elegida, ese modelo elegido especialmente para XGBoost fue el de estimadores de Parzen estructurado en árboles o TPE). Se llegó a la decisión de usar eso dada la utilidad y popularidad de este modelo. La ejecución de la optimización fue facilitada por el framework optuna. Luego de 3000 combinaciones se obtuvo que el mejor constaba de un max_depth de 9, 481 estimadores internos, un learning_rate eta de ~0.046, un gamma de ~0.00061, y parámetros de regularización alpha de ~9.18e-6 y lambda de ~0.024.

Para el armado del **ensamble tipo voting** primero probamos usar como estimadores 4 modelos, constando cada uno del mejor modelo de cada categoría (SVM, KNN, RF y XGBoost) con un sistema de voting "mayoría gana". Luego realizamos un ensamble con los mejores dos modelos, es decir, el mejor RF y el mejor XGBoost, con un sistema de voting que realiza las predicciones tomando el argmax de la suma de las probabilidades de los modelos individuales, lo cual resultó en una mejora del ~4.5% del f1 score respecto al ensamble con los 4 modelos, y en Kaggle la mejora fue del ~5.6%.

Para el armado del **ensamble tipo stacking** primero usamos como estimadores base 4 modelos, igual que para el de tipo voting, constando cada uno del mejor modelo de cada categoría (SVM, KNN, RF y XGBoost), junto con un meta-modelo de tipo regresión logística y el passthrough activado. Igual que antes, realizamos también un ensamble con los mejores dos modelos, es decir, el mejor RF y el mejor XGBoost, con un modelo de regresión logística como meta-modelo y el passthrough desactivado, este ejercicio terminó en una disminución del ~0.0003% del f1 score en test pero una mejora del ~0.0008% en Kaggle, convirtiéndose así en nuestro modelo más performante. Se puede entender que la diferencia entre ambos modelos de stacking es en términos relativos despreciable y no se percibió una mejora tan sustancial como la del ensamble tipo voting.

Cuadro de Resultados

Presentamos un cuadro de resultados donde exponemos las métricas F1, Precision y Recall que obtuvimos de nuestros modelos en test y la métrica F1 obtenida de las predicciones subidas a Kaggle.

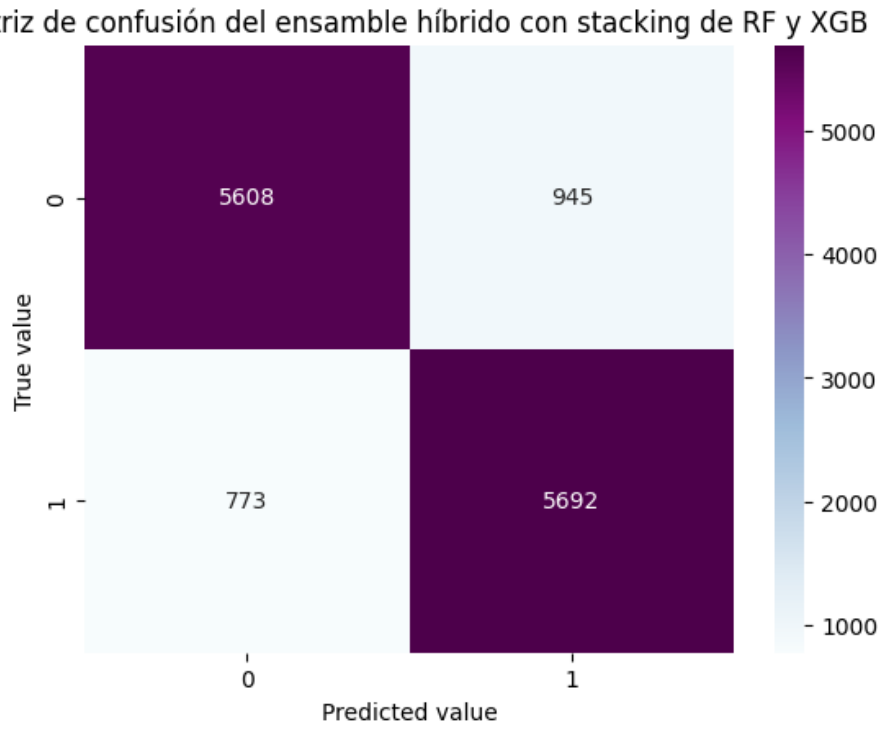
Modelo	F1-Test	Precision Test	Recall Test	Kaggle
KNN	0.7709	0.7328	0.8131	0.72445
SVM	0.8250	0.8093	0.8412	0.78975
Random Forest	0.8647	0.8561	0.8734	0.85337
XGBoost	0.8656	0.8499	0.8819	0.85489
Voting	0.8694	0.8546	0.8847	0.85803
Stacking	0.8688	0.8576	0.8804	0.86055

Descripción de modelos:

- KNN: modelo KNN con hiperparámetros optimizados, usa k=17 vecinos y distancia Manhattan
- SVM: modelo SVM con hiperparámetros optimizados, usa kernel radial y regularización C=6 y coeficiente de kernel gamma=0.01
- Random Forest: modelo RF con hiperparámetros optimizados, tiene 150 estimadores y el splitting se rige por el criterio gini.
- XGBoost: modelo XGB con hiperparámetros optimizados bayesianamente, tiene 481 estimadores y cada árbol tiene una profundidad máxima de 9.
- Voting: ensamble híbrido de voting, los modelos que utiliza son el RF y el XGB mencionados arriba, realiza las predicciones tomando el argmax de la suma de las probabilidades de los modelos individuales

- Stacking: ensamble híbrido de stacking, los modelos que utiliza son el RF y el XGB mencionados arriba, utiliza como meta-modelo una regresión logística. Con un score f1 muy similar al modelo de voting, este fue el que mejor performó en Kaggle.

Matriz de Confusión



Esta es la matriz de confusión del modelo de stacking híbrido utilizando el modelo RF y el XGB que mejor performaron. De aquí se extrajeron las métricas de recall y precision para luego armar el f1 score. Observamos que la métrica de recall es mayor a la de precision, lo cual corresponde a los modelos internos, dado que el RF y el XGB también predicen con un recall más elevado que el precision.

Tareas Realizadas

Integrante	Tarea
DIEM, Walter Gabriel	SVM-XGBoost-Voting-Stacking-Informe
MAIOLO, Alejandro	RF-Voting-Stacking-Informe
RUIZ, Karen Belén	KNN-Informe