

## Práctica 3

# Competición en DrivenData

Miguel Ángel Fernández Gutiérrez

**Inteligencia de Negocio**

5º Doble Grado en Ingeniería Informática y Matemáticas

Universidad de Granada

[mianfg.me](http://mianfg.me)



## Subidas a DrivenData

BEST

0.8626

























CURRENT RANK

121

# COMPETITORS

3434

### SUBMISSIONS

Score	Submitted by	Timestamp	
0.8512	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-29 23:39:17 UTC	 Test
0.7778	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-30 17:55:52 UTC	 #01
0.7656	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-30 18:01:55 UTC	 #02
0.7659	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-30 18:34:27 UTC	 #03
0.8589	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-31 02:08:12 UTC	 #01 (fix)
0.8405	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-31 02:08:46 UTC	 #02 (fix)
0.8412	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2021-12-31 02:09:09 UTC	 #03 (fix)
0.8592	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2022-01-01 22:16:21 UTC	 #04
0.8592	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2022-01-01 22:16:44 UTC	 #05
0.8622	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2022-01-02 00:33:48 UTC	 #06
0.8608	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2022-01-03 11:56:43 UTC	 #07
0.8626	<a href="#">MiguelAngel_Fernandez_UGR_IN</a> 	2022-01-03 12:16:33 UTC	 #08

Nota. Las subidas #01, #02 y #03 tenían un error en el código (las probabilidades del atributo objetivo `seasonal_vaccine` eran las mismas de `h1n1_vaccine` al crear el `pandas.DataFrame` del envío por un error en los índices), y es por ello por lo que las puntuaciones son menores. Ignoraremos tales subidas y tendremos en cuenta las subidas acabadas en *(fix)*. El código proporcionado en los *notebooks* `n.ipynb` y `n_holdout.ipynb` (con  $n \in \{01, 02, 03\}$ ) es funcional y tiene arreglado tal fallo.

# Índice

<b>1. Introducción</b>	<b>3</b>
1.1. Estructura del código . . . . .	3
1.2. Estrategia seguida . . . . .	3
1.3. Primera aproximación a los datos . . . . .	3
<b>2. Resumen de pruebas realizadas</b>	<b>5</b>
<b>3. Descripción de pruebas realizadas</b>	<b>7</b>
3.1. Código proporcionado por el profesor . . . . .	7
3.2. LightGBM con estrategia de preprocesado básica . . . . .	7
3.3. LightGBM con cambios en preprocesado . . . . .	8
3.4. LightGBM con cambio de parámetros . . . . .	9
3.5. LightGBM con preprocesado básico . . . . .	9
3.6. LightGBM con StratifiedKFold . . . . .	10
3.7. <i>Hyperparameter tuning</i> en LightGBM . . . . .	10
3.8. <i>Hyperparameter tuning</i> en XGBoost . . . . .	10
3.9. Cambios en preprocesado con LightGBM óptimo . . . . .	11
3.10. Cambios en preprocesado: añadir clase <code>unknown</code> . . . . .	11
3.11. Cambios en preprocesado: eliminar <code>hhs_geo_region</code> . . . . .	12
3.12. <i>Hyperparameter tuning</i> en RandomForest . . . . .	12
3.13. <i>Hyperparameter tuning</i> en CatBoost . . . . .	12
3.14. LightGBM con SMOTE . . . . .	13
3.15. LightGBM con SMOTENC . . . . .	13
<b>4. Bibliografía</b>	<b>14</b>

# 1. Introducción

En esta práctica abordamos una competición en DrivenData, en concreto, *Flu Shot Learning: Predict H1N1 and Seasonal Flu Vaccines*.

## 1.1. Estructura del código

El código proporcionado se estructura en las carpetas:

- `common`, que contiene el código de uso común, en concreto, de preprocesado y visualización, así como la semilla aleatoria.
- `submissions` contiene los csv de los envíos a DrivenData.
- `data` contiene los datos a usar.
- Cada envío se encuentra en un Jupyter Notebook de la forma `ID.ipynb`, en el directorio raíz.

## 1.2. Estrategia seguida

Para cada modelo se pueden computar tres *scores*:

- **Score en DrivenData.** Score obtenido en la subida.
- **Estimación Cross-Validation.** Es la puntuación obtenida al entrenar el modelo sobre todo el conjunto de *test* usando Cross-Validation. Es un buen *proxy* de la puntuación de DrivenData.
- **Estimación Holdout.** Haciendo uso de *holdout*, obtenemos una estimación mucho más estable de la puntuación que se obtendrá en DrivenData (como puede verse en la Figura 2).

## 1.3. Primera aproximación a los datos

Realizando un análisis de los datos (ver notebook `00.ipynb`) y observando la descripción del benchmark de DrivenData<sup>1</sup> obtenemos información útil de cara a ir haciendo las pruebas:

- Hay muchos valores perdidos.
- Los atributos están incorrelados. Si hubiese atributos altamente correlacionados, deberíamos dejar uno dado que pueden introducir ruido en los modelos, pero en nuestro *dataset* no podemos eliminar ningún atributo (ver Figura 1).
- Los atributos objetivo tienen las clases muy desbalanceadas (ver Figura 2).

---

<sup>1</sup><https://www.drivendata.co/blog/predict-flu-vaccine-data-benchmark/>

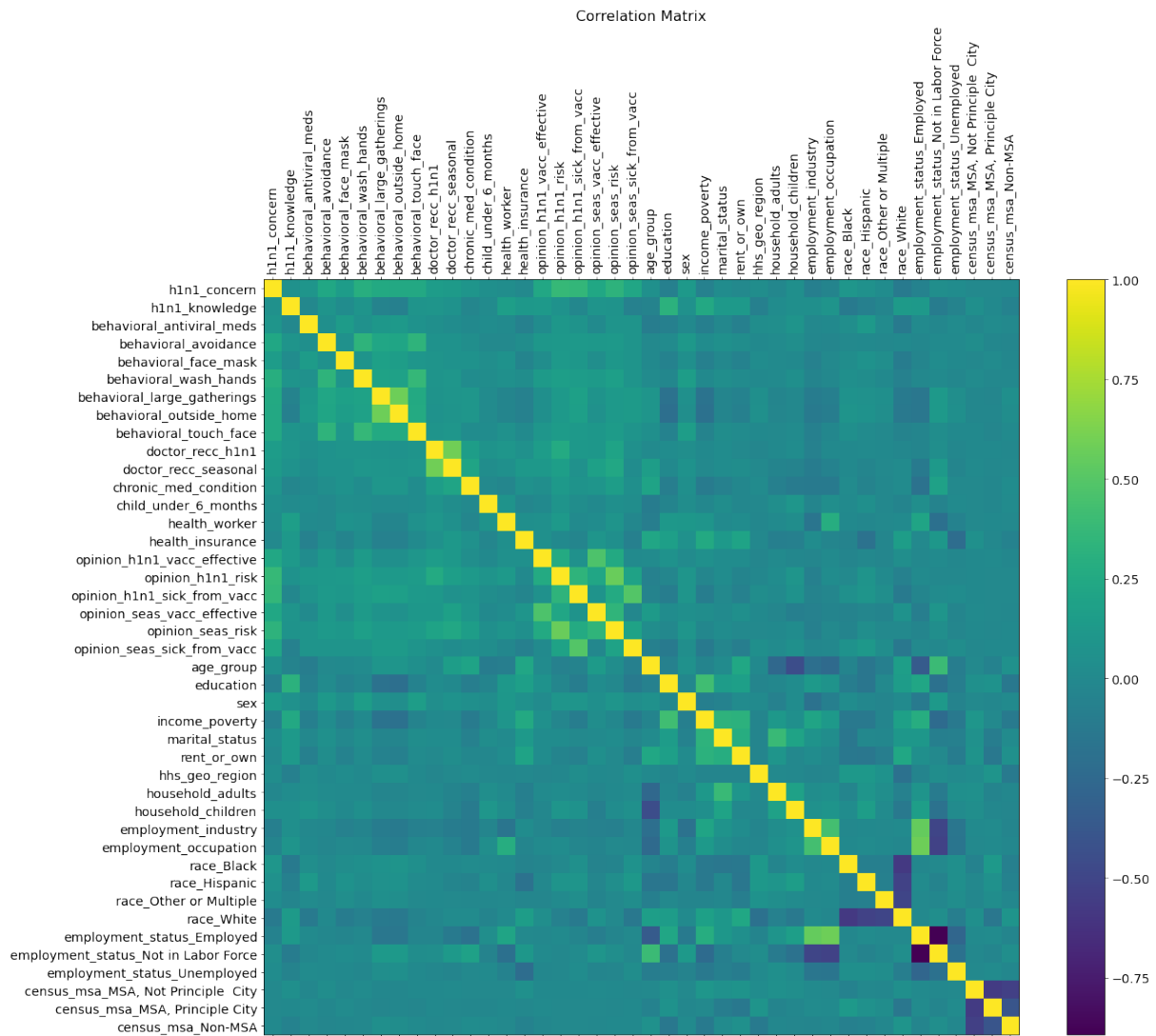


Figura 1: Matriz de correlación

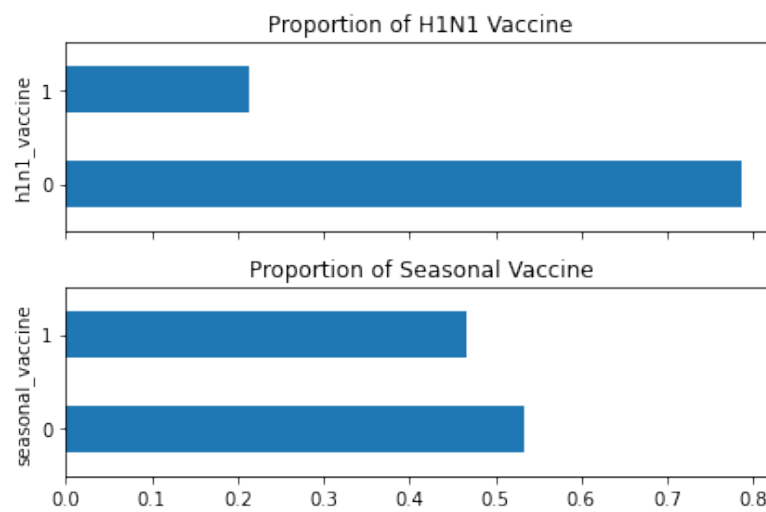
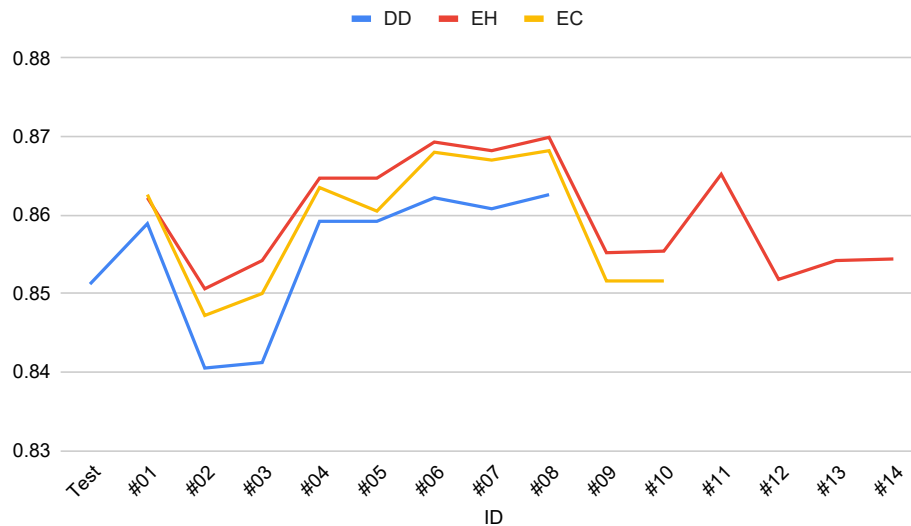


Figura 2: Proporción de clases en cada atributo objetivo

## 2. Resumen de pruebas realizadas



**Figura 3:** Evolución de scores ROC AUC para todas las pruebas

Ap. <sup>1</sup>	ID <sup>2</sup>	Descripción	Parámetros	Scores <sup>3</sup>
3.1	Test	Código proporcionado por el profesor		DD: 0.8512
3.2	#01 (fix)	LightGBM Estrategia de preprocesado básica	n_estimators: 200	DD: 0.8589 EH: 0.8622 EC: 0.8626
3.3	#02 (fix)	LightGBM Cambios en preprocesado	n_estimators: 200	DD: 0.8405 EH: 0.8506 EC: 0.8472
3.4	#03 (fix)	LightGBM Cambio de parámetros	n_estimators: 50	DD: 0.8412 EH: 0.8542 EC: 0.8500
3.5	#04	LightGBM Estrategia de preprocesado básica (del profesor)	n_estimators: 200	DD: 0.8592 EH: 0.8647 EC: 0.8635

<sup>1</sup>Apartado de esta memoria en la que se detalla la prueba

<sup>2</sup>Identificador de la subida. Coincide con la descripción de la subida a DrivenData, en caso de hacerla (ver *Subidas a DrivenData* tras la portada)

<sup>3</sup>DD: Score en DrivenData, EH: Estimación Holdout, EC: Estimación Cross-Validation

Ap.	ID	Descripción	Parámetros	Scores
3.6	#05	LightGBM Uso de StratifiedKFold	n_estimators: 200	DD: 0.8592 EH: 0.8647 EC: 0.8605
3.7	#06	LightGBM <i>Hyperparameter tuning</i>	Mejor estimador: n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	DD: 0.8622 EH: 0.8693 EC: 0.8680
3.8	#07	XGBoost <i>Hyperparameter tuning</i>	Mejor estimador: eta: 0.1 max_depth: 5 subsample: 0.75	DD: 0.8608 EH: 0.8682 EC: 0.8670
3.9	#08	LightGBM con pa- rámetros de #06 Cambios en preprocesado	n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	DD: 0.8626 EH: 0.8699 EC: 0.8682
3.10	#09	LightGBM Cambios en preprocesado (añadir clase unknown)	n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	EH: 0.8552 EC: 0.8516
3.11	#10	LightGBM Cambios en preprocesado (eliminar hhs_geo_region)	n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	EH: 0.8554 EC: 0.8516
3.12	#11	RandomForest <i>Hyperparameter tuning</i>	Mejor estimador: bootstrap: True max_depth: 50 max_features: auto min_samples_leaf: 4 min_samples_split: 5 n_estimators: 1800	EH: 0.8652
3.13	#12	CatBoost <i>Hyperparameter tuning</i>	Mejor estimador: depth: 6 learning_rate: 0.04 iterations: 100	EH: 0.8518
3.14	#13	LightGBM <i>Hyperparameter tuning</i> + SMOTE	n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	EH: 0.8542
3.15	#14	LightGBM <i>Hyperparameter tuning</i> + SMOTENC	n_estimators: 200 colsample_bytree: 0.5 learning_rate: 0.05 num_leaves: 50	EH: 0.8544

### 3. Descripción de pruebas realizadas

#### 3.1. Código proporcionado por el profesor

En primer lugar, realizamos una subida de prueba a DrivenData. Esta subida consiste en ejecutar el código proporcionado por el profesor en `ejemplo_flu.py`, en el que se aplica `RandomForestClassifier` con los parámetros por defecto. Obtenemos una puntuación relativamente buena, y claramente superior a la del *benchmark* proporcionado por DrivenData de 0.8185 (por regresión logística).

Scores obtenidos. DD: 0.8512

#### 3.2. LightGBM con estrategia de preprocesado básica

Probamos a usar el algoritmo LightGBM, por sus buenos resultados en problemas de clasificación. Este algoritmo es más sofisticado que Random Forest, además de que es más rápido y eficiente. Además, es uno de los mejores algoritmos de *boosting*. Probaremos con `n_estimators=200`.

Además, cambiaremos un poco la estrategia de preprocesado:

- Para los atributos categóricos, imputaremos con la moda.
- Para los atributos numéricos, imputaremos con la mediana.
- Usaremos la función `preprocessing.trans_discretize` para pasar los atributos categóricos a numéricos (tiene un efecto equivalente a `LabelEncoder`, se explica con más detalle en el apartado siguiente).

Estas imputaciones se realizan mediante la función `preprocessing.impute` en `preprocessing.py`, que dado un `df` (de tipo `pandas.DataFrame`) aplica una de las siguientes estrategias de imputación:

- `strategy='mean'` reemplaza los valores perdidos con la media.
- `strategy='median'` reemplaza los valores perdidos con la mediana.
- `strategy='mode'` reemplaza los valores perdidos con la moda.
- `strategy='fill'` reemplaza los valores perdidos con el valor proporcionado en el parámetro `fill`.

Obtenemos unos resultados notablemente mejores, posiblemente debido al uso de LightGBM. Intentaremos refinar los resultados mediante cambios en el preprocesado.

Scores obtenidos. DD: 0.8589, EH: 0.8622, EC: 0.8626



### 3.3. LightGBM con cambios en preprocesado

Vamos a probar a incorporar un preprocesado un tanto más complejo. Para ello, hemos implementado dos programas en `preprocessing.py`:

- `preprocessing.trans_onehot`: dado un atributo `attr` con las categorías `{c1, c2, ..., cn}`, creamos  $n$  atributos `attr_ci` donde:

$$\text{attr\_ci} = \begin{cases} 1 & \text{si } \text{attr} = \text{ci} \\ 0 & \text{si } \text{attr} \neq \text{ci} \end{cases}$$

- `preprocessing.trans_discretize`: dado un atributo `attr` y un diccionario `dict`, transforma el valor `val` de cada instancia de `attr` por `dict[val]`.

Para este preprocesado, tenemos en cuenta algo relevante en la discretización de atributos categóricos: al discretizar, estamos imponiendo un orden (y, en principio, no estamos diciendo qué orden). Hay algunos atributos que tienen cierto orden implícito en las categorías, por ejemplo: `age_group` podría discretizarse en orden ascendente de edad, e incorporaríamos cierto sentido al orden de la discretización.

Vamos a realizar por tanto estas transformaciones:

Atributo	Descripción	Transformación aplicada
<code>age_group</code>	Grupo de edad	<code>preprocessing.trans_discretize</code> 1 = 18 - 34 Years 2 = 35 - 44 Years 3 = 45 - 54 Years 4 = 55 - 64 Years 5 = 65+ Years
<code>education</code>	Nivel educativo	<code>preprocessing.trans_discretize</code> 1 = <12 Years 2 = 12 Years 3 = Some College 4 = College Graduate
<code>race</code>	Raza	<code>preprocessing.trans_onehot</code>
<code>sex</code>	Sexo	<code>preprocessing.trans_discretize</code> (a binario) 0 = Male 1 = Female
<code>income_poverty</code>	Ingresos anuales respecto a umbrales de pobreza	<code>preprocessing.trans_discretize</code> 1 = Below Poverty 2 = <= \$75,000, Above Poverty 3 = > \$75,000
<code>marital_status</code>	Estado marital	<code>preprocessing.trans_discretize</code> (a binario) 0 = Not married 1 = Married

Atributo	Descripción	Transformación aplicada
rent_or_own	Situación del hogar (alquiler o propiedad)	<code>preprocessing.trans_discretize</code> (a binario) 0 = Rent 1 = Own
employment_status	Estado de empleo	<code>preprocessing.trans_onehot</code>
hhs_geo_region	Residencia	*
census_msa	Residencia respecto a áreas metropolitanas	<code>preprocessing.trans_onehot</code>
employment_industry	Industria donde trabaja	*
employment_occupation	Tipo de ocupación	*

Con \* se han destacado aquellas variables que corresponden a categorías más complejas, y en los que especificar una forma de discretización no tiene mucho sentido. En ellas, usamos `preprocessing.trans_discretize` sin especificar el diccionario. Nótese que si hiciésemos `onehot` insertaríamos demasiados atributos.

Ejecutamos el algoritmo y obtenemos unos resultados notablemente peores. Además, la diferencia entre los scores de *train* y *test* es notable, por lo que puede ser que nuestro modelo esté entrando en *overfitting*. Veremos cómo podemos suplir esto en pruebas posteriores.

Scores obtenidos. DD: 0.8405, EH: 0.8506, EC: 0.8472

### 3.4. LightGBM con cambio de parámetros

Vamos a ejecutar LightGBM cambiando el parámetro `n_estimators` a 50 con el objetivo de evitar el *overfitting*, y dejando el preprocesado anterior a pesar de haber empeorado los resultados para ver si mejora. Obtenemos unos resultados ligeramente mejores, pero aún inferiores a los del primer intento con LightGBM. Dudamos que pueda tratarse del preprocesado, así que lo modificaremos de nuevo en el siguiente apartado.

Scores obtenidos. DD: 0.8412, EH: 0.8542, EC: 0.8500

### 3.5. LightGBM con preprocesado básico

Volveremos a usar el preprocesado proporcionado por el profesor en `ejemplo_flu.py` y con los parámetros de LightGBM que mejor han funcionado hasta ahora: `n_estimators=200`. Obtenemos de este modo los mejores resultados hasta ahora.

Scores obtenidos. DD: 0.8592, EH: 0.8647, EC: 0.8635

### 3.6. LightGBM con StratifiedKFold

En un primer análisis del *dataset* pudimos ver cómo las clases estaban claramente desbalanceadas en ambos atributos objetivo. En *scikit-learn* disponemos de la clase *StratifiedKFold* a usar en lugar de *KFold* para balancear en Cross-Validation. Sin embargo, *StratifiedKFold* no funciona con multioutput. Para ello, realizaremos ciertas modificaciones en el código para tratar con ambos atributos objetivo independientemente. Tras ejecutar esta variación, no vemos ningún cambio en el score de *DrivenData*, lo cual nos indica que esta opción no es viable.

Sin embargo, la estrategia de separar los modelos para cada atributo objetivo la usaremos más adelante, dado que la mayoría de utilidades de *scikit-learn*, así como de *imbalanced-learn* y de los distintos algoritmos no funcionan bien con *MultiOutputClassifier*.

Scores obtenidos. DD: 0.8592, EH: 0.8647, EC: 0.8605

### 3.7. Hyperparameter tuning en LightGBM

Procederemos mejorando *LightGBM*, el modelo que mejor resultado ha dado hasta ahora. Para ello, usaremos *hyperparameter tuning*: dado que las medidas de Cross-Validation son un buen *proxy* de las medidas con datos desconocidos, obtendremos el modelo con mejor score en CV. Usaremos el siguiente espacio de búsqueda:

```
search_space = {
    'n_estimators': [50, 200],
    'colsample_bytree': [i/10.0 for i in range(3, 6)],
    'learning_rate': [0.05, 0.1],
    'num_leaves': [30, 50]
}
```

El mejor modelo obtenido tiene los parámetros siguientes:

```
{'params': {'n_estimators': 200,
    'colsample_bytree': 0.5,
    'learning_rate': 0.05,
    'num_leaves': 50},
    'roc_auc': 0.8693384996292246}
```

Obtenemos la mejor puntuación hasta ahora. Esto nos indica que *LightGBM* es un buen algoritmo para nuestro problema.

Scores obtenidos. DD: 0.8622, EH: 0.8693, EC: 0.8680

### 3.8. Hyperparameter tuning en XGBoost

Dado que hemos obtenido muy buenos resultados al tunear los hiperparámetros en *LightGBM*, intentaremos hacer lo mismo con otros algoritmos. Probaremos con *XGBoost* y con el siguiente espacio de búsqueda:

```
search_space = {  
    'eta': [0.05, 0.1, 0.2],  
    'max_depth': [3, 5, 8, 10],  
    'subsample': [0.5, 0.75, 1]  
}
```

El mejor modelo obtenido tiene los parámetros siguientes:

```
{'params': {'eta': 0.1, 'max_depth': 5, 'subsample': 0.75},  
 'roc_auc': 0.8682050936387379}
```

Sin embargo, no logramos mejorar los resultados de LightGBM.

Scores obtenidos. DD: 0.8608, EH: 0.8682, EC: 0.8670

### 3.9. Cambios en preprocesado con LightGBM óptimo

Vamos a intentar mejorar el resultado de LightGBM aplicando un preprocesado más complejo. Retomaremos el preprocesado del apartado 3.3, y la misma estrategia de valores perdidos que el apartado 3.2. Ejecutamos el algoritmo y obtenemos el mejor resultado hasta ahora. Este resultado mejora considerablemente con respecto al resultado #06. Esto nos indica que el preprocesado es muy relevante.

Scores obtenidos. DD: 0.8626, EH: 0.8699, EC: 0.8682

Nota. De aquí en adelante, dado que disponemos de pocos intentos en DrivenData, sólo subiremos aquellos que mejoren los scores que usamos como *proxy*, principalmente el que mejore EH.

### 3.10. Cambios en preprocesado: añadir clase unknown

En el foro de la competición<sup>2</sup> se sugiere una estrategia de preprocesado que se basa en la observación de que en las clases `employment_occupation` y `employment_industry` hay muchos valores nulos debidos a principios de protección de datos.

Impugnar la clase mayoritaria en estos casos puede introducir información en nuestro *dataset* que no es correcta. Por ello, introduciremos una categoría `unknown` para los datos perdidos mediante la función `preprocessing.impute` con `strategy='fill'`.

Los resultados que obtenemos, sin embargo, son notablemente peores.

Scores obtenidos. EH: 0.8552, EC: 0.8516

---

<sup>2</sup><https://community.drivendata.org/t/nul-values-in-columns-containing-personal-data-of-the-volunteers/5997>

### 3.11. Cambios en preprocesado: eliminar `hhs_geo_region`

Observando los datos también vemos cómo el atributo `hhs_geo_region` tiene cierta variabilidad, por lo que nos preguntamos si es posible que esté introduciendo ruido en el *dataset*. Intentamos ignorar este atributo, y obtenemos unos resultados ligeramente peores que en la alternativa anterior de preprocesado; esto nos indica que `hhs_geo_region` tiene información relevante para nuestra clasificación.

Scores obtenidos. EH: 0.8554, EC: 0.8516

### 3.12. *Hyperparameter tuning* en RandomForest

Intentamos ahora probar otros algoritmos. Comenzaremos por Random Forest, que a pesar de ser un algoritmo más sencillo que los anteriores, puede ser que nos dé buenos resultados. Usaremos *hyperparameter tuning* usando el preprocesado de #08, el mejor candidato hasta el momento.

Dado que los tiempos de entrenamiento son mayores, usaremos una búsqueda aleatoria en el siguiente espacio de búsqueda:

```
search_space = {
    'bootstrap': [True, False],
    'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None],
    'max_features': ['auto', 'sqrt'],
    'min_samples_leaf': [1, 2, 4],
    'min_samples_split': [2, 5, 10],
    'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]
}
```

El mejor modelo obtenido tiene los parámetros siguientes:

```
{'params': {'bootstrap': True,
            'max_depth': 50,
            'max_features': 'auto',
            'min_samples_leaf': 4,
            'min_samples_split': 5,
            'n_estimators': 1800},
 'roc_auc': 0.8651691969281137}
```

No obtenemos una buena puntuación en ningún momento, incluso tras varias ejecuciones, no llegando a 0.86 en ninguno de los *proxies*.

Scores obtenidos. EH: 0.8652

### 3.13. *Hyperparameter tuning* en CatBoost

Probaremos también el algoritmo CatBoost, pues es posible que de buenos resultados. Procedemos del mismo modo que con Random Forest, haciendo una búsqueda no exhaustiva en el siguiente espacio:

```
search_space = {
    'depth': [4, 5, 6, 7, 8, 9, 10],
    'learning_rate': [0.01, 0.02, 0.03, 0.04],
    'iterations': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
}
```

El mejor modelo obtenido tiene los parámetros siguientes:

```
{'params': {'depth': 6, 'learning_rate': 0.04, 'iterations': 100},
 'roc_auc': 0.8518087063929591}
```

Scores obtenidos. EH: 0.8518

### 3.14. LightGBM con SMOTE

Al explorar el *dataset* observamos el desbalance de la distribución de las clases en los atributos objetivo. Probaremos insertar atributos sintéticos usando SMOTE. Para ello, usaremos *imbalanced-learn*.

Dado que el SMOTE de *imbalanced-learn* no funciona con `MultiOutputClassifier`, como en intentos anteriores entrenaremos individualmente dos modelos para clasificar cada atributo objetivo. Además, volveremos a hacer un *hyperparameter tuning* dado que es posible que al insertar más atributos los parámetros obtenidos no sean los óptimos. Finalmente, hacemos uso de `GridSearchCV` de *scikit-learn* y lo integramos en un `Pipeline` de *imbalanced-learn*, para agilizar la implementación.

Obtenemos unos resultados muy mediocres mediante esta técnica. Nos preguntamos si hay alguna otra forma de usar atributos sintéticos.

Scores obtenidos. EH: 0.8542

### 3.15. LightGBM con SMOTENC

Probamos a usar SMOTENC en lugar de SMOTE, pues es un algoritmo de generación de instancias sintéticas más orientado a atributos categóricos. Este algoritmo se implementa de forma idéntica a SMOTE, pero pasando los atributos categóricos.

Obtenemos una leve mejora respecto a SMOTE, pero aún quedamos lejos de la mejor puntuación obtenida.

Scores obtenidos. EH: 0.8544

## 4. Bibliografía

Towards Data Science, *The right way of using SMOTE with Cross-validation*.

<https://towardsdatascience.com/the-right-way-of-using-smote-with-cross-validation-92a8d09d00c7>

scikit-learn, *Cross-validation: evaluating estimator performance*.

[https://scikit-learn.org/stable/modules/cross\\_validation.html](https://scikit-learn.org/stable/modules/cross_validation.html)

imbalanced-learn, *SMOTENC*.

[https://imbalanced-learn.org/dev/references/generated/imblearn.over\\_sampling.SMOTENC.html](https://imbalanced-learn.org/dev/references/generated/imblearn.over_sampling.SMOTENC.html)

Towards Data Science, *Essential guide to Multi-Class and Multi-Output Algorithms in Python*.

<https://towardsdatascience.com/essential-guide-to-multi-class-and-multi-output-algorithms-in-python-3041fea55214>

Machine Learning Mastery, *How to Develop a Light Gradient Boosted Machine (LightGBM) Ensemble*

<https://machinelearningmastery.com/light-gradient-boosted-machine-lightgbm-ensemble/>

Eijaz Allibhai, *Hold-out vs. Cross-validation in Machine Learning*.

<https://medium.com/@eijaz/holdout-vs-cross-validation-in-machine-learning-7637112d3f8f>