“HR Analytics: Job Change of Data Scientists” Parte II

Juan Ignacio Ron

https://www.kaggle.com/arashnic/hr-analytics-job-change-of-data-scientists

Indice

Resumen de la primera parte

Una compañía vinculada al mundo del Big Data y Data Science quiere ampliar su equipo de científicos de datos. Dado que brinda cursos sobre la temática, quiere buscar candidatos entre sus alumnos, más específicamente, entre aquellos que están dispuestos a cambiar de empleo. Esto le ayudaría a reducir los tiempos y costos de contratación y capacitación.

La empresa tiene una base datos con casi 20 mil alumnos. La misma tiene un total de **12 variables** que incluyen información demográfica, educativa y laboral de los alumnos, así como también si buscan o no cambiar de empleo. La mayoría de las dimensiones son **categóricas** y algunas presentan gran cantidad de **datos faltantes**.

En el este set de datos inicial observamos que la cantidad de alumnos que buscan cambiar de trabajo es una proporción menor: sólo 25% del total de los casos observados.

Resumen de la primera parte

A partir de un primer análisis concluimos que el **estudiante promedio** es hombre, proviene de una ciudad de alto desarrollo, tiene experiencia relevante, formación de grado o superior en STEM (Ciencias, Tecnología, Ingeniería y Matemática) trabaja en una empresa privada, tiene menos de diez años de experiencia, y cambió de trabajo hace relativamente poco tiempo.

Por otro lado, el **estudiante que busca trabajo** en general proviene de una ciudad de nivel de desarrollo bajo-medio, no tiene experiencia previa relacionada al Big Data / Data Science, posee título de grado y tiene menos de diez años de experiencia laboral.

Abordando el problema desde Machine Learning

Especificación del problema

La variable '**target**' representa la búsqueda (cuando toma un valor de 1) o ausencia de búsqueda de empleo (valor de 0), y es la clave para resolver el problema que enfrentamos.

Tal como hicimos en la primera parte, se puede analizar la relación de las distintas variables con respecto al target para entender cómo influye cada una de ellas en la búsqueda laboral. No obstante, **la capacidad de análisis alcanza un límite** cuando tratamos de analizar el comportamiento de tres o más variables en simultáneo. Por otro lado, **es difícil medir cuánto influye cada variable** sobre el target. Podemos tener una noción o intuición, pero es difícil de cuantificar su relación.

El aporte del Machine Learning

Estos dificultades pueden resolverse rápidamente a partir de un modelo de **Machine Learning**. Podríamos implementar un algoritmo que procese el cuerpo de datos y "aprenda" la relación entre cada variable y nuestra variable objetivo, en un proceso que se llama "**entrenamiento**". El algoritmo calcula la contribución de las variables a la probabilidad de buscar empleo. La forma de calcular esta probabilidad y la medida de contribución a dicha probabilidad depende de cada tipo de modelo.

Luego, podríamos utilizar lo que el algoritmo "aprendió" para predecir, con cierto margen de error, cuál será el target en nuevos alumnos. Es decir que el algoritmo utilizará lo aprendido para predecir sobre casos que no ha visto con anterioridad. En esta instancia, llamada "**testeo**", podremos evaluar la performance del modelo, su capacidad predictiva, comparando los valores reales de target con sus estimaciones. Superado el testeo, podremos decir que nuestro modelo se puede aplicar en lo sucesivo para nuevos alumnos.

Esta es una solución superadora para el problema que enfrentamos: **no sólo conoceremos la importancia de las distintas variables de una manera cuantitativa** (la contribución a la probabilidad)**, sino que además tendremos una predicción puntual para cada alumno.**

Modelo a utilizar

Para resolver este problema utilizaremos el algoritmo clasificador de **LightGBM**. Un clasificador es un modelo que asigna a cada observación una etiqueta, en función de las variables independientes. En este caso, asignará un valor de 1 cuando estime que un alumno está en la búsqueda de empleo, o un valor de 0 cuando no lo esté (tal como ocurre con nuestro set inicial de datos).

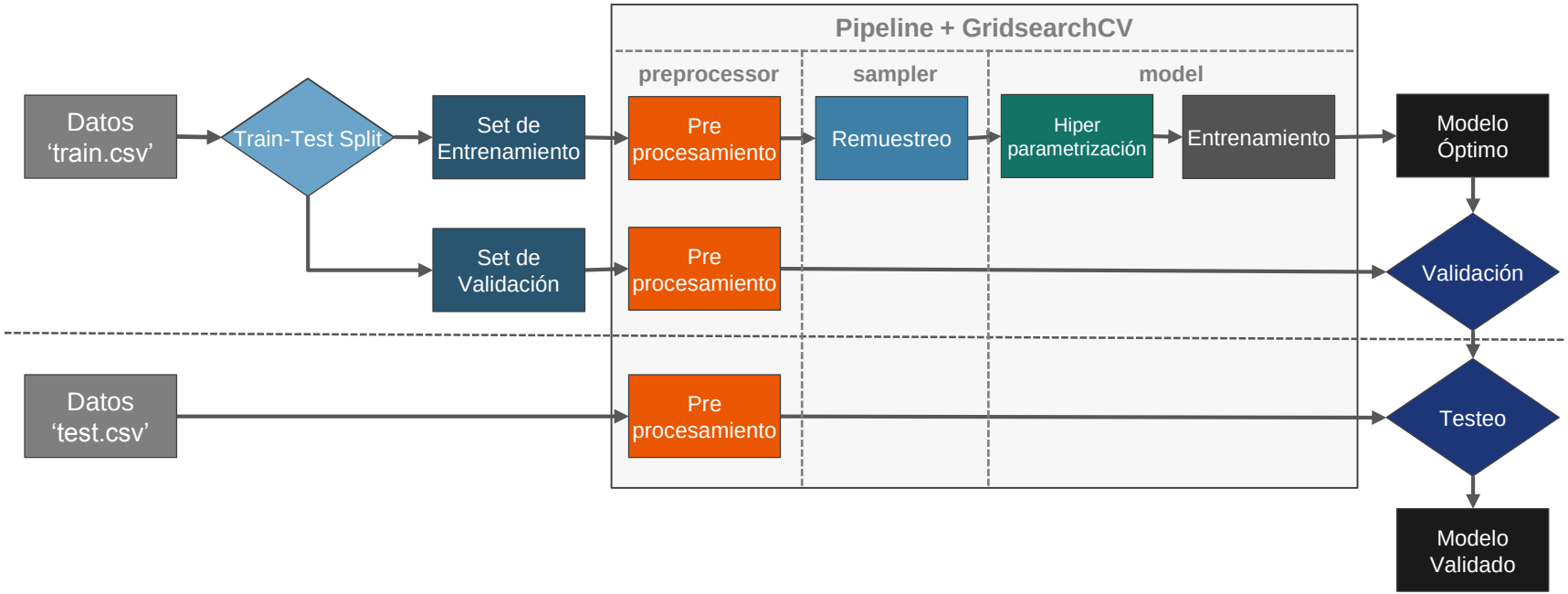
Este algoritmo fue desarrollado por Microsoft y está basado en **árboles de decisión** y en **potenciación de gradiente**. Por sus características particulares, funciona más rápido que otros modelos basados en árboles, y además tiene una menor tendencia al overfitting o **sobreajuste**.

Wikipedia ofrece una explicación un poco más detallada de cómo opera. **Aquí** se puede consultar su documentación.

Implementación del modelo

Flujo de trabajo

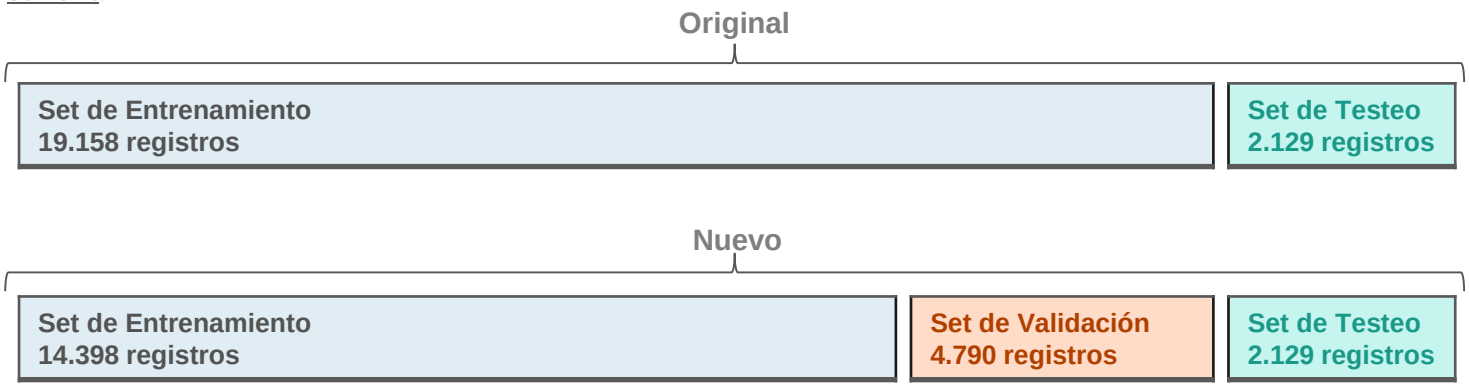
Desarrollar el modelo pensado para este problema conlleva una serie de etapas, que decidimos representar en el siguiente gráfico. En las próximas secciones explicaremos en detalle cada una.



Sets de Entrenamiento y Testeo

La implementación de modelos de Machine Learning requiere la separación del set de datos en al menos dos partes: un set para el entrenamiento y un set para la fase de testeo. Primero, el modelo aprende los datos de entrenamiento. Después se evalúa la precisión de sus predicciones con los datos de test.

En este caso los sets de entrenamiento y de testeo vienen separados por defecto en dos archivos distintos: 'train.csv' y 'test.csv'. Si bien no es algo estrictamente necesario (más adelante aplicaremos una validación cruzada de 5 particiones), dejaremos el set de testeo para una evaluación final, y partiremos el set de entrenamiento (75-25%) como si fuera nuestro único set, respetando la proporción de las clases en cada subset.



Preprocesamiento – Variables categóricas y numéricas

Variables Numéricas

```
['city_development_index', 'training_hours']
```

Debemos estandarizar sus valores, para que la escala en que están medidos no afecte el aprendizaje del algoritmo. Utilizaremos el módulo *StandardScaler* de *Sci-kit Learn*.

Descartaremos la variable 'id_enrollee' (dígito identificador del alumno). Es una variable numérica que no aporta ninguna información, y que incluso puede entorpecer el apredizaje del modelo.

Variables Categóricas

```
['city', 'gender', 'relevent_experience', 'enrolled_university', 'education_level',
 'major_discipline', 'experience', 'company_size', 'company_type', 'last_new_job']
```

Para implementar nuestro modelo debemos transformarlas en variables dummy. Utilizaremos el módulo *OneHotEnconder* de la biblioteca *Sci-kit Learn*.

Preprocesamiento - Tratamiento de los datos faltantes

El faltante de datos es un problema muy común en la vida real, que puede resolverse de diferentes maneras:

1) Eliminar los registros con algún valor faltante, o que tengan cierto porcentaje de sus columnas con valores faltantes.
2) Eliminar las variables con algún valor faltante, o que tengan cierto porcentaje de sus registros con valores faltantes.

Estas primeras dos alternativas deben manejarse con cuidado, ya que estaremos descartando información que puede ser valiosa, sobre todo si la proporción de faltantes es elevada. Otras alternativas son:

3) Tratar los faltantes como un valor más dentro de los posibles. Para ello se asigna un '*0*' en las variables numéricas y un '*empty*' en las categóricas.
4) Imputar un valor a los campos faltantes. Puede ser el valor promedio o la mediana para los campos numéricos, o el valor más frecuente (moda) para los categóricos.

En este trabajo abordaremos las dos últimas posibilidades. Existen otras **alternativas** que por su complejidad hemos descartado para el caso.

Desbalanceo de clases

Otro inconveniente con el que debemos lidiar es el desbalanceo de las clases, también común en la vida real. Hablamos de **desbalanceo** cuando una de las etiquetas es más frecuente que el resto. Esto trae inconvenientes para la implementación del algoritmo, en dos sentidos. Primero, porque las métricas comúnmente utilizadas pierden relevancia.

Para dar un ejemplo, supongamos que tenemos un dataset con dos clases balanceadas. Si tuviera un modelo que predice con el 80% de efectividad (acierta 8 de 10 veces), podríamos decir que es un buen modelo.
Supongamos ahora que tenemos otro dataset, desbalanceado, donde la clase 1 aparece sólo el 20% de las veces. Si nuestro modelo predijera con un 80% de efectividad, ya no podemos decir que sea un buen modelo. Se podría construir un modelo totalmente obsoleto, que predijera la clase 0 para todas las observaciones, y que tendría precisamente el 80% de efectividad (sólo se equivocaría en el 20% de las veces, las que corresponden a la clase 1).

¿Cómo resolvemos este primer inconveniente? Debemos analizar otras métricas, que dependen del problema que estemos tratando de resolver, y que comentaremos más adelante.

El segundo inconveniente aparejado al desbalanceo tiene que ver con el propio aprendizaje del modelo. Al entrenarlo con clases desbalanceadas puede volverse demasiado complejo para el algoritmo detectar la relación entre las variables y el target, sobre todo si el desbalanceo es **severo**.

Remuestreo

Para poder abordar el desbalanceo tenemos distintas alternativas:

1) No hacer nada: es una opción factible si el desbalanceo es moderado y las variables no son demasiado complejas.
2) **Oversampling** o sobremuestreo: de manera aleatoria (o en base a otro criterio) se toman observaciones de la clase minoritaria y se copian en el dataset original hasta igualar la cantidad de observaciones de la clase mayoritaria, o bien hasta alcanzar cierta proporción predeterminada entre clases.
3) **Subsampling** o submuestreo : de manera aleatoria (o en base a otro criterio) se toman observaciones de la clase mayoritaria y se eliminan del dataset hasta igualar la cantidad de observaciones de la clase minoritaria, o bien hasta alcanzar cierta proporción predeterminada entre clases.
4) **SMOTE**: a partir de las observaciones minoritarias se crean de manera artificial otras nuevas hasta igualar la cantidad de observaciones de la clase minoritaria, o bien hasta alcanzar cierta proporción predeterminada.

Estas estrategias se pueden combinar entre sí. En este caso, las abordamos de manera separada.

Hiperparametrización del modelo

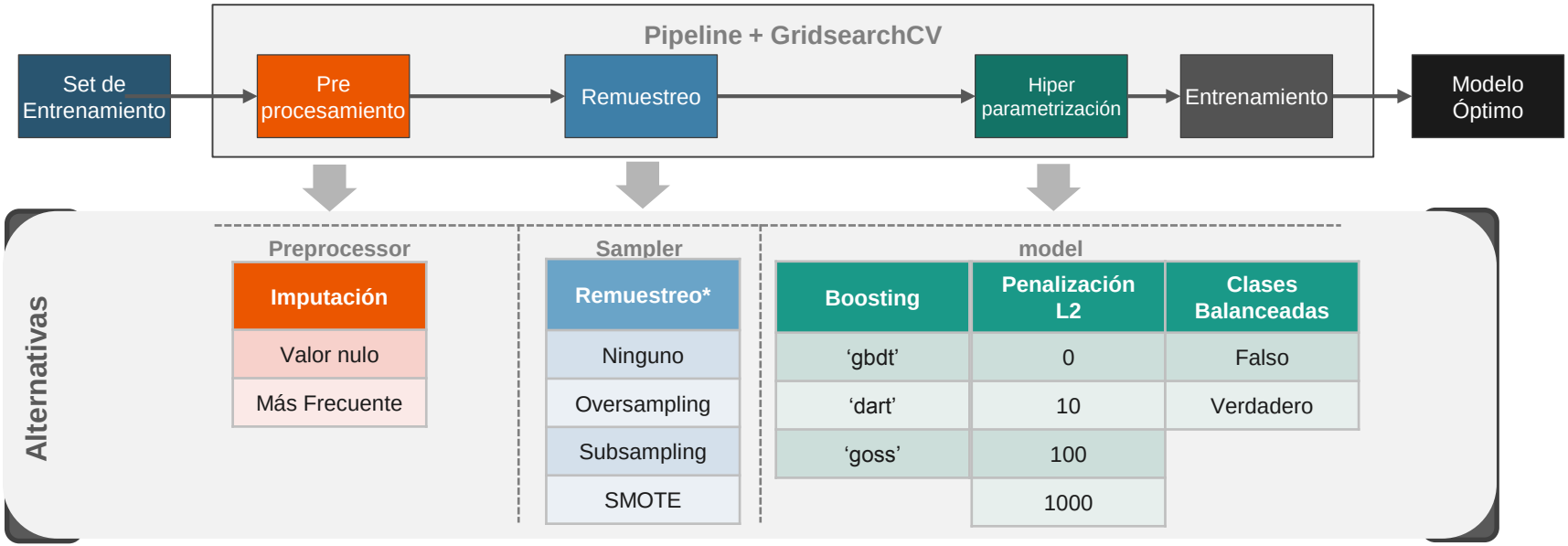
LightGMB Classifier tiene al menos 20 hiperparámetros para especificar (**aquí** una guía rápida). En este caso consideramos sólo tres:

1) '**boosting _type**': se refiere al método de boosting que utiliza el modelo, con tres posibilidades distintas. Por defecto viene definido como '*gbdt*', que indica el método de descenso de gradiente tradicional, y las otras alternativas son '*dart*' y '*goss*'.
Probaremos las tres alternativas.

2) '**reg_lambda**': indica el grado de **regularización** bajo norma L2 para los coeficientes del modelo. Es un campo tipo *float*, que por defecto viene dado en '*.0*' y equivale a no aplicar regularización.
Iniciaremos el algoritmo con cuatro alternativas: 0, 10, 100 y 1000.

3) '**is_unbalance**': un parámetro específico para indicar si las clases están desbalanceadas o no. Por defecto definido como '*False'*, es decir que asume clases balanceadas.
Probaremos las dos posibilidades.

Entrenamiento del modelo - Alternativas



* Para cada tipo de resampleo se evaluaron cinco alternativas distintas, alterando la proporción clase mayoritaria/clase minoritaria entre 0.5 y 1.

A partir de todas las combinaciones posibles para cada etapa hay un total de **¡912 modelos a evaluar!**

Pipeline + GridSearch

Contamos con dos herramientas muy poderosas para poder evaluar y comparar las distintas alternativas posibles, de una manera sencilla y sistematizada. Por un lado, creamos un **Pipeline** (de la librería *imbalanced-learn*) que combine las distintas etapas de nuestro proceso:

```
full_pipeline = imbPipeline(steps = [
    ('preprocessor', preprocessing_step_1),
    ('sampler', oversampling_step),
    ('model', lgbm_step)
])
```

El Pipeline nos permite sistematizar la implementación del proceso, haciendo que la salida de una etapa se convierta en la entrada del siguiente. Una vez entrenado, el Pipeline puede replicarse en el set de testeo, o en cualquier cuerpo de datos posterior, sin caer en filtración de datos ni errores.

En segundo lugar, utilizamos el módulo **GridSearchCV**, que busca la combinación óptima de alternativas para cada etapa del Pipeline, de acuerdo a una métrica preestablecida. En este caso, tomamos las métricas *ROC-AUC* y *Recall*, que comentaremos más adelante.

Este esquema de búsqueda funciona de manera exhaustiva, literalmente evalúa cada una de las combinaciones posibles que definimos. Para hacerlo trabaja con un esquema de **validación cruzada**, en este caso de 5 particiones. Una vez hallado el mejor modelo, éste se vuelve a entrenar sobre el set de entrenamiento completo.

Modelo Optimo, Métricas y Validación

Modelo óptimo - Características

Una vez completado el proceso de búsqueda, la herramienta GridSearchCV nos permite acceder al mejor modelo y a la combinación óptima de parámetros a partir del atributo *'best_params_'*.

```
{'model__boosting_type': 'dart',
 'model__is_unbalance': False,
 'model__reg_lambda': 100,
 'preprocessor': ColumnTransformer(transformers=[('num',
                                  Pipeline(steps=[('imputer',
                                                   SimpleImputer(fill_value=0,
                                                                 strategy='constant')),
                                                  ('scaler', StandardScaler())]),
                                  ['city_development_index', 'training_hours']),
                                 ('cat',
                                  Pipeline(steps=[('imputer',
                                                   SimpleImputer(fill_value='Empty',
                                                                 strategy='constant')),
                                                  ('encoder',
                                                   OneHotEncoder(handle_unknown='ignore'))]),
                                  ['city', 'gender', 'relevent_experience',
                                   'enrolled_university', 'education_level',
                                   'major_discipline', 'experience',
                                   'company_size', 'company_type',
                                   'last_new_job'])]),
 'sampler': RandomOverSampler(sampling_strategy=1),
 'sampler__sampling_strategy': 1}
```

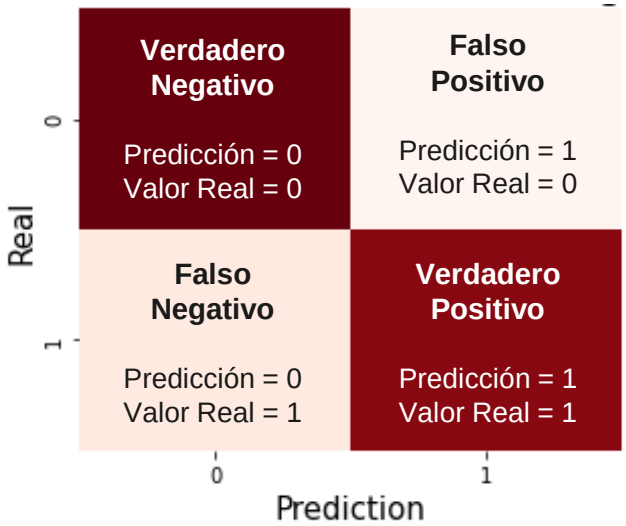
La combinación que arrojó mejores resultados fue aquella que hizo **oversampling** sobre la clase minoritaria **hasta igualar** el tamaño de la clase mayoritaria.

Además, completó los faltantes con el valor **'Empty'** (no hay faltantes en variables numéricas).

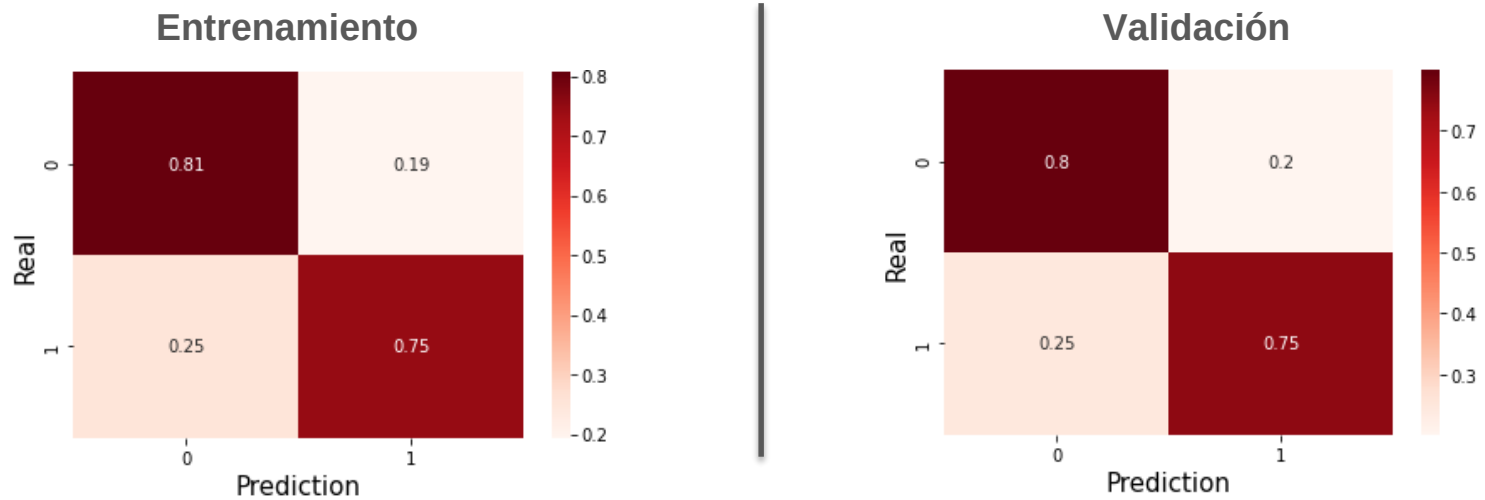
Por otro lado, el modelo implementa boosting por '**dart**' con una **penalización L2** de 100.

Evaluación de métricas – Matriz de Confusión

La primera herramienta que utilizamos para evaluar la performance del modelo es la matriz de confusión. Esta matriz muestra los cuatro resultados posibles a partir de la comparación entre el valor real y el valor predicho:



Evaluación de métricas – Matriz de Confusión



Los resultados se muestran como proporción del total de cada clase (real). Como primer comentario, la performance del algoritmo parece ser buena, por la relación entre aciertos (verdaderos positivos y negativos) versus los errores (falsos positivos y negativos).

En segundo lugar, el rendimiento de nuestro modelo en el set de entrenamiento es coherente con el de validación. Esto indicaría que no se está produciendo un sobreajuste u overfitting.

Evaluación de métricas – Recall, Precision y F1 score

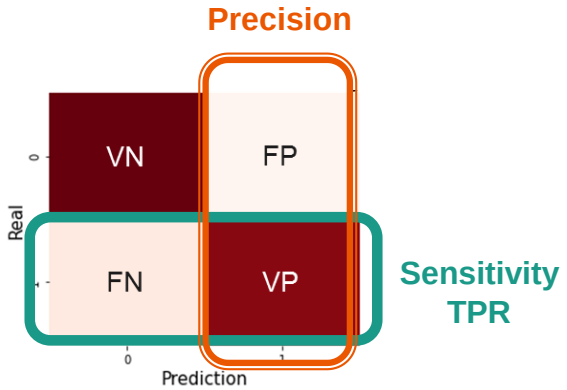
La segunda herramienta es el '*classification_report*' que genera *sklearn*. Esta tabla muestra las principales métricas utilizadas en problemas de clasificación:

- *Recall*: es la proporción de predicciones correctas sobre el total de observaciones reales. Para la clase 1 se llama *Sensitividad* o *True Positive Rate* (TPR) y 1 equivale a VP / (VP + FN). Para la clase 0 se llama *Especificidad* o *True Negative Rate* (TNR) y equivale a VN / (VN + FP).

- *Precision*: equivale a la cantidad de aciertos sobre el total de predicciones. Para la clase 1 equivale a VP / (VP + FP), y para la clase 0 equivale a VN / (VN + FN).

- *F1-score*: es la media armónica entre *Recall* y *Precisión*.

- *Accuracy*: no es relevante al ser un problema de clases desbalanceadas. Equivale a (VN + VP) / Total.



Evaluación de métricas – Recall, Precision y F1

Entrenamiento

	precision	recall	f1-score	support
0.0	0.91	0.81	0.85	10785
1.0	0.56	0.75	0.64	3583
accuracy			0.79	14368
macro avg	0.73	0.78	0.75	14368
weighted avg	0.82	0.79	0.80	14368

Validación

	precision	recall	f1-score	support
0.0	0.91	0.80	0.85	3596
1.0	0.55	0.75	0.64	1194
accuracy			0.79	4790
macro avg	0.73	0.78	0.74	4790
weighted avg	0.82	0.79	0.80	4790

En clasificación con clases desbalanceadas, depende de la naturaleza del problema la importancia de uno u otro indicador. En aquellos casos en donde sea más importante la necesidad de maximizar la correcta detección de observaciones (clase 1), tendrá relevancia el *Recall*. En aquellos donde predomine maximizar los aciertos, será mayor el peso de la *Precisión*.

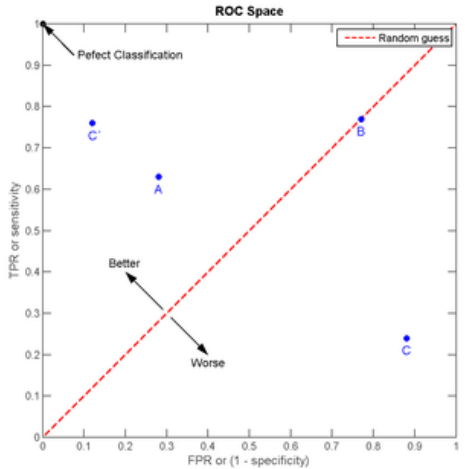
En esta caso, optamos por centrarnos en el *Recall*. Es decir que nos interesa detectar la mayor cantidad de casos de clase 1, aquellos que buscan cambiar de trabajo. En ese sentido, la performance del modelo es buena y, nuevamente, congruente entre entrenamiento y validación.

Evaluación de métricas – ROC-AUC

Otra forma de para evaluar la performance del modelo es la curva ROC (Receiver Operating Characteristic).

El **umbral de decisión** es el punto de separación que el clasificador toma de referencia para asignar la etiqueta 1 o la etiqueta 0 a un caso determinado. Este umbral se define en términos de probabilidad de pertenencia a la clase, y por defecto se ubica en 0.5. Es decir que una determinada observación será clasificada como de clase 1 si su probabilidad de pertenencia es mayor a 0.5, o clase 0 si es menor.

La **curva ROC** mide como varían los indicadores de *Recall* y *FPR (1-Especificidad)* a medida que ese umbral varía, desde 0 (100% probabilidad de pertenencia a la clase 1) hasta 1 (100% pertenencia a la clase 0).

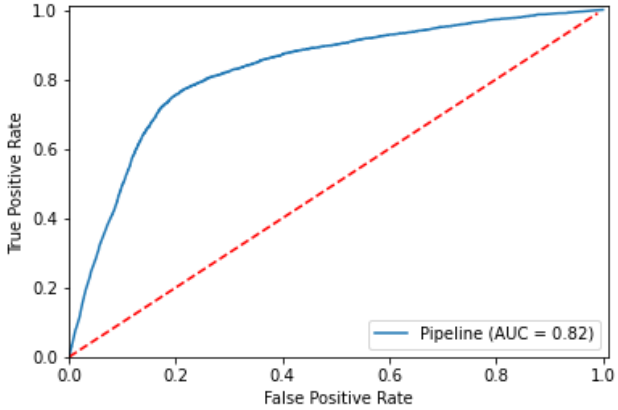


La línea punteada diagonal representa la curva que tendría un clasificador aleatorio, es decir, uno que asignara etiquetas 1 y 0 de manera azarosa. Una curva que esté cercana o por debajo de la línea punteada no sería mejor que asignar las etiquetas de manera aleatoria.

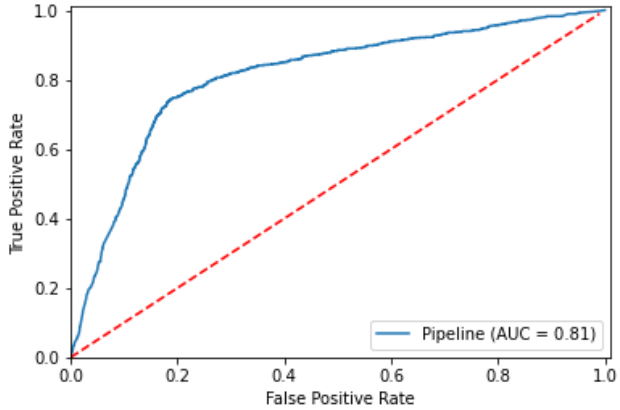
Finalmente, la sigla **AUC** representa el área debajo de la curva, respecto al área total del gráfico (valor óptimo = 1).

Evaluación de métricas – ROC-AUC

Entrenamiento



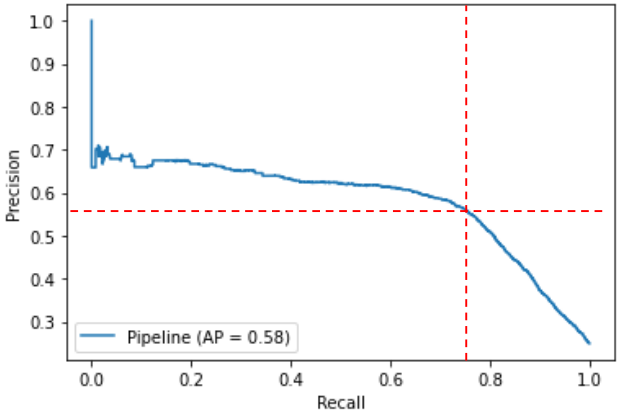
Validación



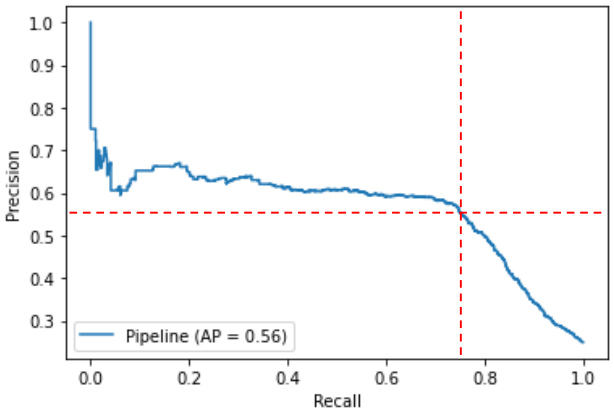
El área bajo la curva de la curva ROC muestra una superficie superior al 0,8 en ambos sets, lo cual es signo de buen desempeño del algoritmo.

Evaluación de métricas – Precision-Recall

Entrenamiento



Validación



Finalmente, mostramos la curva Precisión-Recall. Como su nombre lo indica, muestra la relación entre *Recall* y *Precisión* para los distintos niveles de umbral de decisión.

En este caso, observamos una *Precisión* promedio de 0,58 y 0,56 (línea punteada roja). Además, vemos que es posible elevar el *Recall* del algoritmo en torno del 0,8 manteniendo la Precisión cerca del promedio

Optimizando el umbral de decisión

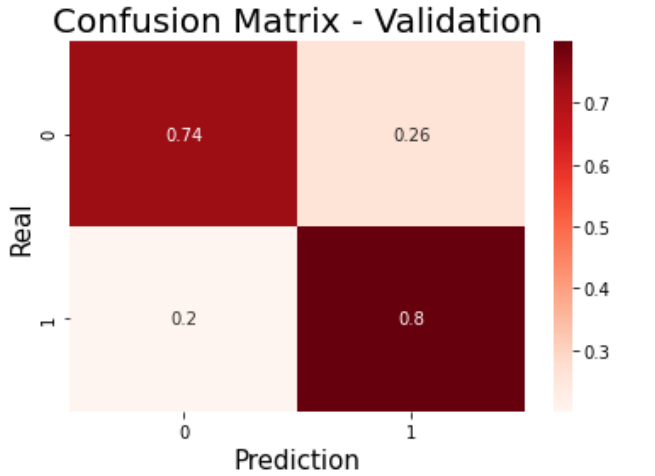
A continuación, mostraremos los resultados de las distintas métricas, elevando el umbral de decisión del modelo a 0.4, aplicado sobre el set de validación

En este caso se obtuvo una ganancia de *Recall* de 0.05 (de 0.75 a 0.80) y una pérdida de la precisión también de 0.05 (de 0.55 a 0.5). Asimismo, el f1-score cayó de 0.64 a 0.62.

Es decir que nuestro modelo óptimo, con el nuevo umbral de decisión, **detecta 8 de cada 10 alumnos que están dispuestos a cambiar de trabajo**, mientras que **de cada 10 predicciones, 5 son correctas**.

Si el foco esta puesto en detectar la mayor cantidad de casos posibles, restando importancia a los “falsos positivos” , **éste será nuestro modelo definitivo.**

	precision	recall	f1-score	support
0.0	0.92	0.74	0.82	3596
1.0	0.50	0.80	0.62	1194
accuracy			0.75	4790
macro avg	0.71	0.77	0.72	4790
weighted avg	0.81	0.75	0.77	4790



Testeo del Modelo

Características del Set de Testeo

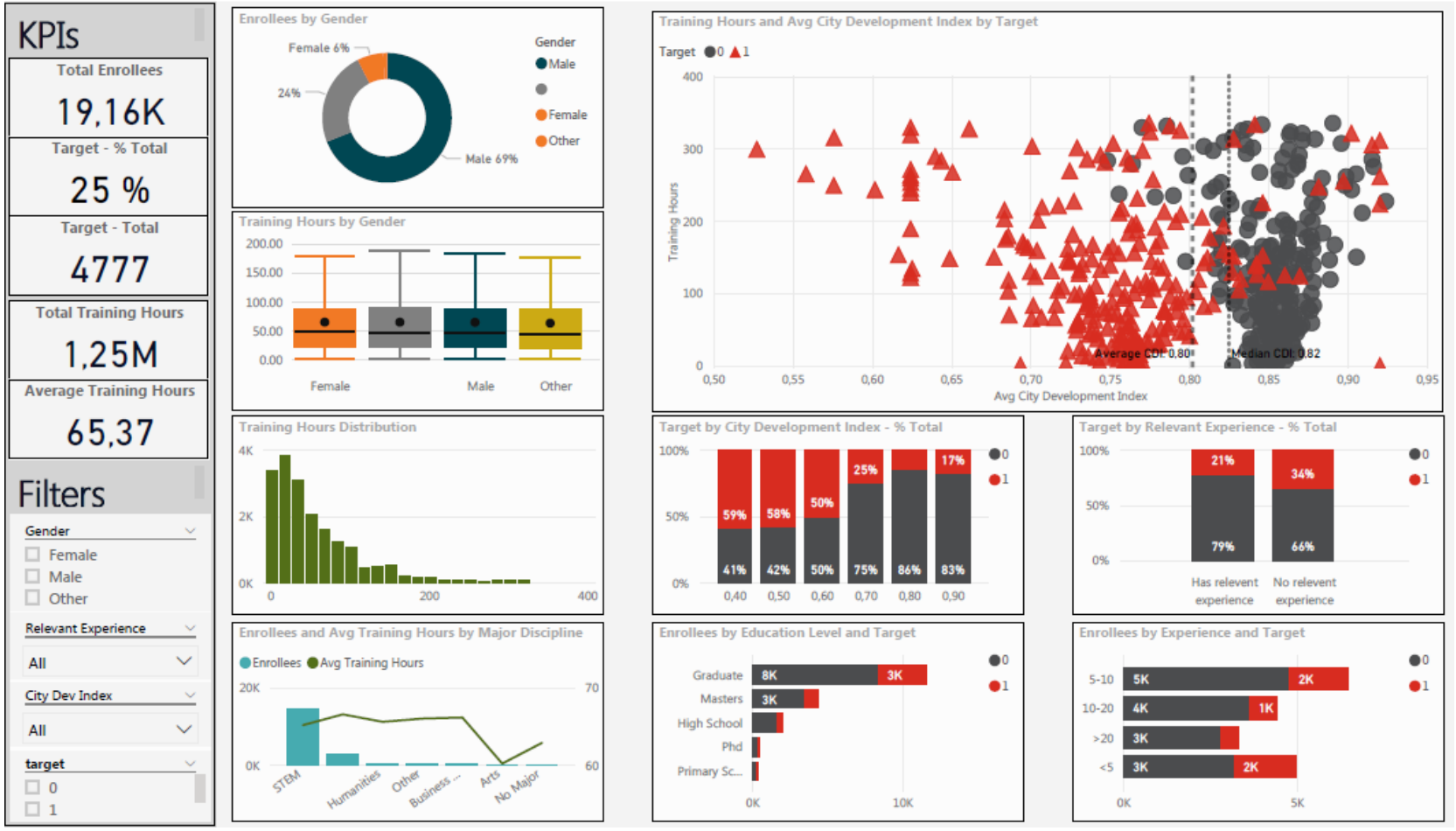
Como medida previa a testear el algoritmo analizaremos las características del set de datos de testeo. Esto no es necesario para la implementación, dado que el Pipeline ya esta armado para procesar nuevos conjuntos de datos, sino que es una medida para analizar la consistencia de los datos (‘*sanity check*’) con respecto a los de entrenamiento. En este caso aplicamos el dashboard elaborado en la primera parte del trabajo.

- La proporción de casos de clase 1 se mantiene en un nivel similar (25% vs 27%).
- La proporción de estudiantes por género, disciplina, nivel educativo también es parecida.
- El promedio de horas de capacitación es el mismo, y la distribución de horas tiene la misma forma.
- Se observa una mayor incidencia de la clase 1 entre los estudiantes procedentes de ciudades de menor nivel de desarrollo.
- La relación entre horas de capacitación, nivel de desarrollo y la búsqueda laboral es más difusa que en el set de entrenamiento.
- El porcentaje de campos nulos es similar y ocurre en las mismas variables.

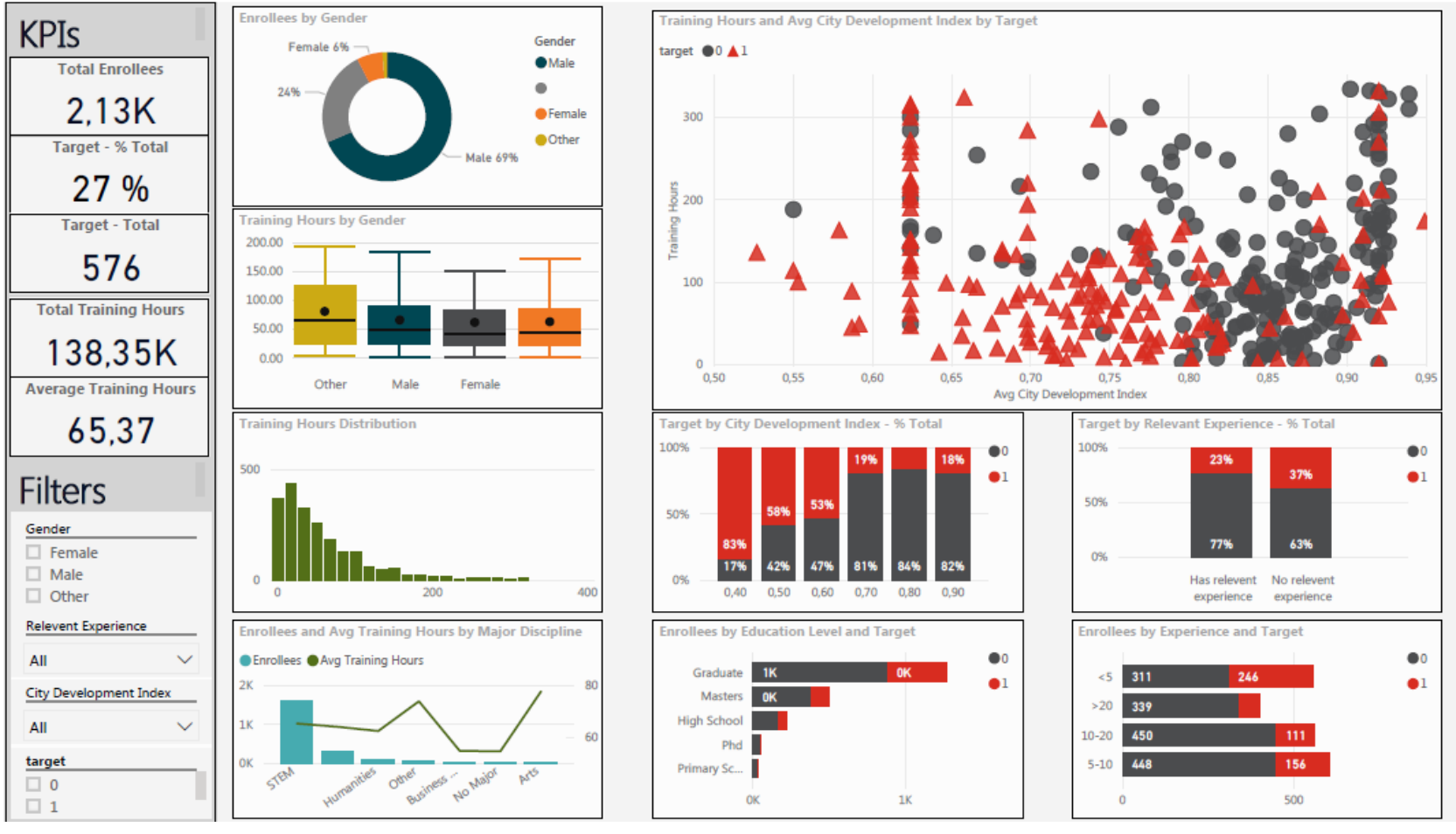
Porcentaje de nulos	
enrollee_id	0.00
city	0.00
city_development_index	0.00
gender	23.86
relevent_experience	0.00
enrolled_university	1.46
education_level	2.44
major_discipline	14.65
experience	0.23
company_size	29.22
company_type	29.78
last_new_job	1.88
training_hours	0.00

En las siguientes filminas mostramos cómo se veía el dashboard alimentado por el set inicial (testeo + validación) y cómo se ve con el nuevo dataset.

Dashboard – Set de entrenamiento + validación



Dashboard – Set de testeo



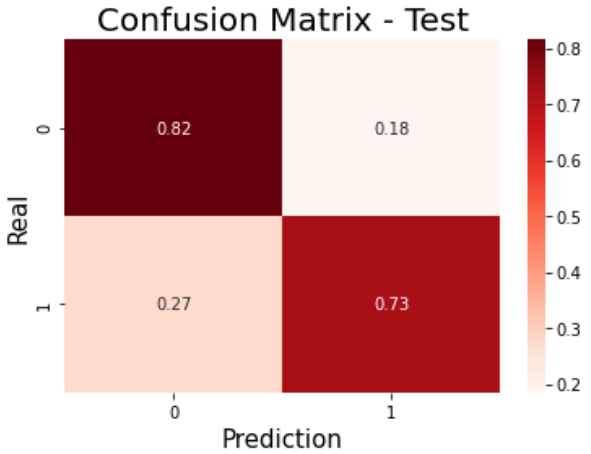
Resultados

A continuación, los resultados obtenidos por nuestro modelo, con el umbral de decisión preestablecido (0.5).

	precision	recall	f1-score	support
0.0	0.89	0.82	0.85	1553
1.0	0.59	0.73	0.65	576
accuracy			0.79	2129
macro avg	0.74	0.77	0.75	2129
weighted avg	0.81	0.79	0.80	2129

Observamos que el rendimiento en términos de *Precisión* y de *f1-score* ha aumentado ligeramente: de 0.55 a 0.59 y de 0.64 a 0.65, respectivamente. Mientras tanto el *Recall*, la variable que hemos establecido como más importante, ha reducido su performance en 0.02 puntos.

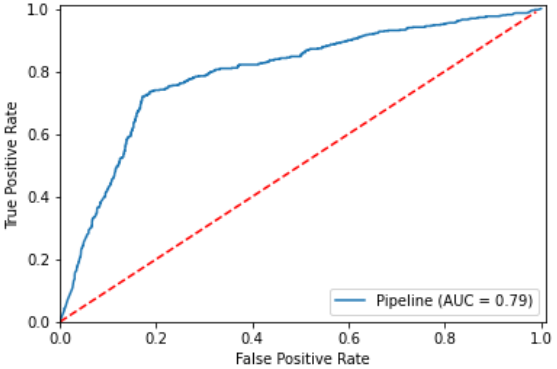
Las diferencias no son significativas respecto a las obtenidas en entrenamiento y validación. Es decir que **el modelo ha mantenido su robustez para predecir pese al encontrarse con un set de datos completamente nuevo**.

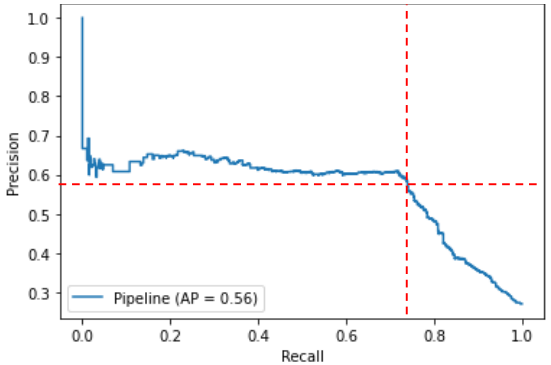


Resultados

Las mismas conclusiones se desprenden del análisis del ROC-AUC, con una diferencia de apenas 0,01 puntos respecto los sets de entrenamiento y validación. La *Precisión* del modelo a los distintos umbrales de decisión tiene un valor promedio en línea con las pruebas anteriores. De nuevo, esta diferencia no es significativa.

Por otro lado, la curva de *Precisión* parece tener un corte más abrupto que el observado en los sets de entrenamiento y validación, y la *Precisión* cae más fuertemente después del umbral de 0.5 (línea punteada roja). De todas maneras, volveremos a probar el umbral de 0.4.



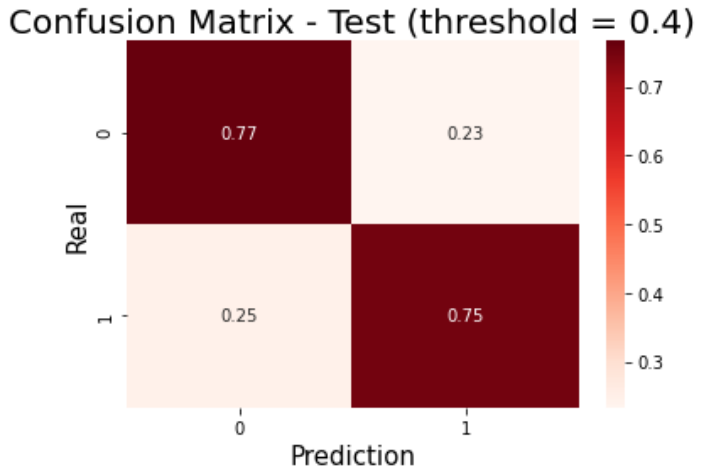


Resultados

Se muestran a continuación los resultados obtenidos con un umbral de 0.4, previamente evaluado.

Para el set de validación, este umbral había logrado una performance de 0.80 en *Recall*, 0.50 de *Precisión* y 0.62 de f1-score. En este caso la *Precisión* se mantiene más alta, pero la variable que definimos como más relevante, el *Recall* se ha mantenido en 0.75. Es decir que se ha producido una perdida de 0.02 puntos de *Precisión* y *f1-score* a cambio de 0.02 puntos en *Recall*. Consideramos que este rendimiento es **más que aceptable**.

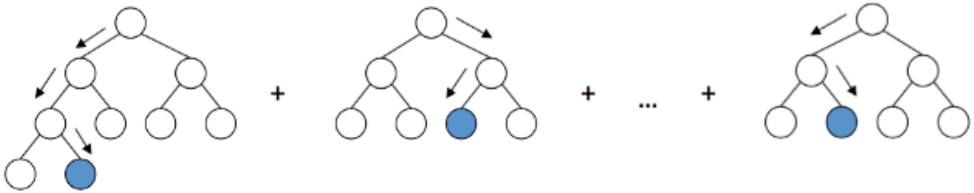
	precision	recall	f1-score	support
0.0	0.89	0.77	0.83	1553
1.0	0.55	0.75	0.63	576
accuracy			0.76	2129
macro avg	0.72	0.76	0.73	2129
weighted avg	0.80	0.76	0.77	2129



Feature Importance

Feature importance

Los algortimos basados en árboles suelen contar con un atributo específico, llamado ‘feature_importance’, que indica precisamente la importancia de las variables para el modelo, es decir, cuáles tienen más peso a la hora realizar la clasificación.



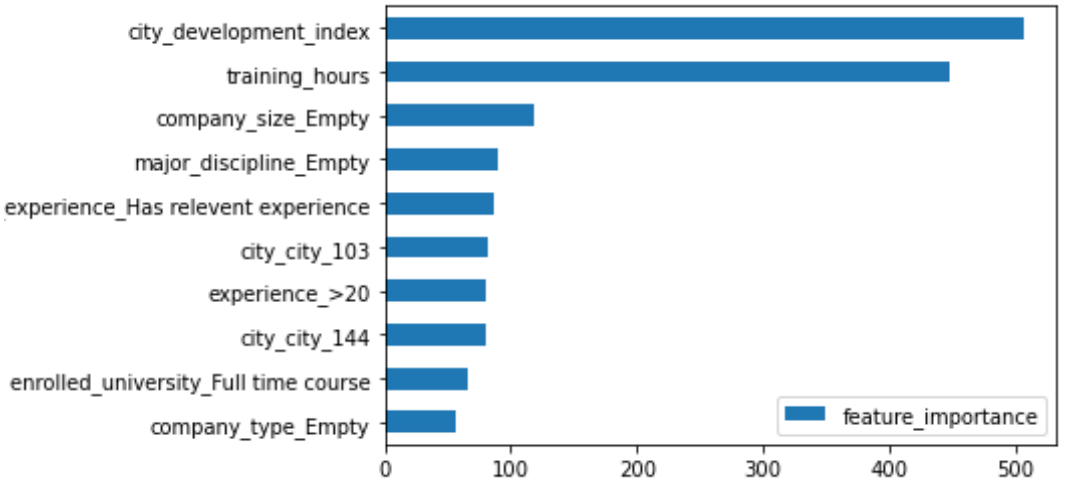
La fórmula en que la importancia calcula depende de cada algoritmo, y en el caso particular de **LightGBM** hay dos alternativas principales: 1) la **cantidad de veces** que se usa la variable para particionar el set de datos, o 2) la **ganancia** que genera la variable para el modelo a partir de su uso. En esta oportunidad tomamos la primera alternativa.

En contraste a lo realizado en la primera parte del trabajo, donde las variables se exploraron y compararon visualmente, este método nos da un valor preciso. En casos donde las variables son muy numerosas, esto nos sirve además para realizar una selección o eliminación de variables irrelevantes.

Feature importance

A continuación mostramos las diez variables de mayor importancia para nuestro modelo óptimo.

	feature_importance
city_development_index	507
training_hours	447
company_size_Empty	118
major_discipline_Empty	89
experience_Has relevent experience	87
city_city_103	82
experience_>20	80
city_city_144	80
enrolled_university_Full time course	66
company_type_Empty	56



Se destaca la importancia de ‘city_development_index’ y ‘training_hours’, con valores muy superiores al resto.
En la primera parte había algún indicio de su importancia, ahora podemos estar seguros de su significancia.

Conclusiones

Conclusiones

- En la primera parte, caracterizamos al alumno promedio a partir de un análisis de las distintas variables existentes (demográficas, educativas, económicas).
- A partir de la relación de cada variable respecto al target, pudimos identificar aquellas dimensiones que están más vinculadas a la búsqueda de trabajo. No obstante, este análisis es limitado, ya que no se pueden explorar más de tres variables de manera simultánea, y además la relación no es cuantificable.
- Asimismo, implementamos un dashboard en PowerBI para procesar conjuntos de datos de los estudiantes y observar las métricas y variables relevantes para el negocio.
- Encontramos en el Machine Learning una herramienta muy útil y eficiente para la resolución del problema, superando además los obstáculos a los cuales nos enfrentamos.
- Para su implementación identificamos tres etapas: preprocesamiento, resampleo y modelización. Asimismo, cada etapa cuenta con distintas alternativas, dando lugar a un total de 912 modelos diferentes para resolver el problema.

Conclusiones

- Todo el proceso de ML fue automatizado a través de un Pipeline y optimizado a través de GridSearchCV, de manera prácticamente automática.

- Generamos un modelo que pudo predecir sobre nuevos conjuntos de datos con un 0.75 de *Recall* y una *Precisión* de 0.55. Es decir que detecta 75 de cada 100 alumnos que buscan trabajo, y de cada 100 alumnos que predice en búsqueda laboral, 55 realmente lo están.

- Los resultados del modelo son consistentes para el set de entrenamiento, validación y de testeo, por lo cual no hay indicios de que se esté produciendo un sobreajuste u overfitting. El modelo a priori es replicable, siempre que las características de los alumnos se mantengan similares en el tiempo.

- Nuestro modelo nos ofrece una medida de la importancia de cada variables en la búsqueda laboral, a partir de su atributo 'feature_importance'. La cantidad de horas de capacitación y el desarrollo de la ciudad de origen del alumno son, por lejos, las variables que más relación guardan respecto a la búsqueda laboral.