Desafío - Análisis lineal discriminante

- Para realizar este desafío debes haber estudiado previamente todo el material disponibilizado correspondiente a la unidad.
- Una vez terminado el desafío, comprime la carpeta que contiene el desarrollo de los requerimientos solicitados y sube el `.zip` en el LMS.
- Desarrollo desafío:
    - El desafío se debe desarrollar de manera Individual.
    - Para la realización del desafío necesitarás apoyarte del archivo *Apoyo Desafío - Análisis lineal discriminante*.

Requerimientos

Trabajaremos con una base de datos sobre clientes morosos de un banco. Dentro de ésta se registran las siguientes observaciones:

- `default`: Variable Binaria. Registra si el cliente entró en morosidad o no.
- `income`: Ingreso promedio declarado por el cliente.
- `balance`: total del saldo en la cuenta de crédito.
- `student`: Variable binaria. Registra si el cliente es estudiante o no.

Ejercicio 1: Preparación de ambiente de trabajo

- Importe los módulos básicos para el análisis de datos.
- Importe las clases `LabelEncoder`, `StandardScaler` y `LabelBinarizer` de `preprocessing.`
- Importe las funciones `train_test_split` y `cross_val_score` de `model_selection.`
- Importe la función `classification_report` de `metrics`.
- Importe las clases `LinearDiscriminantAnalysis` y `QuadraticDiscriminantAnalysis`.
- Agregue la base de datos en el ambiente de trabajo.
- Inspeccione la distribución de cada atributo.

Ejercicio 2: Modelo base

- Recuerde que los modelos de `sklearn` no soportan datos que no sean numéricos. Transforme los atributos pertinentes con `LabelEncoder`.
- Genere muestras de validación y entrenamiento, reservando un 33% de los datos como validación.
- Genere un modelo con `LinearDiscriminantAnalysis` sin modificar los hiper parámetros. Genere métricas de evaluación utilizando `classification_report`.
- Comente sobre cuál es el desempeño del modelo en cada clase, así como en general.

Ejercicio 3: Refactorización 1 - información a priori

- Dado que trabajamos con modelos generativos, podemos incluir información exógena. Para este caso agregaremos dos distribuciones:
    - Asumamos que hay un 50/50 de morosos y no morosos.
    - Asumamos que hay un 60/40 de morosos y no morosos.
- Por cada modelo, reporte las métricas de clasificación.

Ejercicio 4: Refactorización 2 - oversampling

Digresión: Synthetic Over(Under)Sampling

- Por lo general podemos intentar aliviar el problema del desbalance de clases mediante la ponderación dentro del algoritmo. Otra alternativa es el muestreo con reemplazo dentro de los conjuntos de entrenamiento. Estos métodos clásicos se conocen como **Oversampling** cuando repetimos registros aleatorios de la clase minoritaria, y **Undersampling** cuando eliminamos aleatoriamente registros de la clase mayoritaria.
- Un contratiempo de estos métodos clásicos es que pueden replicar información sesgada que afecte el desempeño de generalización del modelo. Si los datos son malos, estaremos replicando estas fallas.
- Otra solución es generar ejemplos de entrenamiento sintéticos mediante el entrenamiento de ejemplos de la clase minoritaria. A grandes rasgos la solución funciona de la siguiente forma: En función a un subconjunto de datos correspondientes a la clase minoritaria, entrenamos algún modelo no supervisado o generativo como Naive Bayes, KMeans o KNearestNeighbors para generar representaciones sintéticas de los datos **en el espacio de atributos de la clase específica** mediante $x_{nuevo-ejemplo} = x_i + \lambda(x_{zi} - x_i)$ es un ejemplo de entrenamiento de la clase minoritaria y λ es un parámetro de interpolación aleatorio $\lambda \sim Uniforme(0, 1)$.

- Uno de los problemas más graves de esta base de datos, es el fuerte desbalance entre clases. Ahora generaremos observaciones sintéticas mediante SMOTE (Synthetic Minority Oversampling Technique). Para ello, debemos agregar el paquete a nuestro ambiente virtual. En nuestro terminal agregamos `conda install -c conda-forge imbalanced-learn`. Incorpore SMOTE en el ambiente de trabajo con la siguiente sintáxis `from imblearn.over_sampling import SMOTE`.
- Para implementar oversampling, debemos generar nuevos objetos que representan nuestra muestra de entrenamiento incrementada artificialmente. Para ello implemente la siguiente sintaxis:

```
from imblearn.over_sampling import SMOTE
# Instanciamos la clase
oversampler = SMOTE(random_state=11238, ratio='minority')
# generamos el eversampling de la matriz de entrenamiento y
X_train_oversamp, y_train_oversamp = oversampler.fit_sample(X_train,
y_train)
```

- Vuelva a entrenar el modelo con los datos aumentados de forma artificial y comente sobre su desempeño.

Ejercicio 5: Refactorización 3 - QDA

- Por último, implemente un modelo `QuadraticDiscriminantAnalysis` con los datos aumentados artificialmente. Genere las métricas de desempeño.
- Comente a grandes rasgos sobre el mejor modelo en su capacidad predictiva.