TALLER DE PROCESAMIENTO DE SEÑALES

1er Cuatrimestre 2026 - Trabajo Práctico Nº 4 - LDA, QDA y KNN

IMPORTANTE: Solamente se podrá importar numpy, matplotlib.pyplot y `fetch_olivetti_faces` de sklearn.

La base de datos `fetch_olivetti_faces` (sklearn) contiene 400 imágenes de rostros (10 imágenes de cada una de las 40 personas distintas). Se desea hacer un algoritmo que determine la identidad de la persona.

(**a**) *Exploración de datos:*

1. Construir un código que cargue los datos de sklearn, cuando estos estén disponibles, y en caso contrario los cargue del archivo `olivetti.npy`. Elegir 6 imágenes al azar y graficarlas.

2. Separar la base de datos en entrenamiento y testeo, eligiendo al azar 2 imágenes de cada persona para el conjunto de testeo. 👉: Hay dos factores claves para decidir hacer un *split* estratificado. La primera es que hay muy pocas muestras y se corre el riesgo de que alguna clase quede vacía. La segunda es que la proporción de muestras de cada clase no es una variable relevante a aprender, ya que por como se confeccionó la base de datos hay la misma cantidad de imágenes de cada persona.

(**b**) *Análisis del discriminante lineal:*

1. Implementar un algoritmo de LDA para resolver esta tarea. El código debe estar estructurado de la siguiente manera:

```
class LDA:

    # Inicializar atributos y declarar hiperparámetros
    def __init__(self,...

    # Etapa de entrenamiento
    def fit(self,X,y):

    # Etapa de testeo soft
    def predict_proba(self,X):

    # Etapa de testeo hard
    def predict(self,X):

    # Cómputo del Accuracy
    def accuracy(self,X,y):

    # Simular muestras de forma sintética
    def sampling(self, n_samples=1):
```

2. Entrenar un algoritmo LDA con la base de datos construida. 👉: Combinando 320 imágenes no se podrá generar una covarianza de rango completo. Antes de invertirla se deberá regularizarla como $\tilde{\Sigma} = \Sigma + \lambda \cdot I$.

3. Indicar el *accuracy* de testeo.

4. Simular 3 muestras sintéticas y graficarlas.

(**c**) *Análisis del discriminante cuadrático:* Repetir el inciso (**b**) para el algoritmo QDA. 👉: Incluso más sensible que la inversa es el cómputo de $\log|\Sigma_k|$. Antes de entrenar el algoritmo, proponer algún método intuitivo que permita seleccionar λ.

(**d**) *Vecinos más cercanos:*

1. Implementar un algoritmo KNN para resolver esta tarea. El código debe estar estructurado de la siguiente manera:

```
class KNN:

    # Inicializar atributos y declarar hiperparámetros
    def __init__(self,...

    # Etapa de entrenamiento
    def fit(self,X,y):

    # Etapa de testeo soft
    def predict_proba(self,X):

    # Etapa de testeo hard
    def predict(self,X):

    # Cómputo del Accuracy
    def accuracy(self,X,y):
```

2. Graficar el *accuracy* de testeo en función de la cantidad de vecinos K seleccionada.