


LDA, QDA y KNN

Este trabajo práctico está basado en una base de datos para estudio estadístico de diagnóstico de diabetes.

(a) Exploración de datos:

1. Cargar el archivo `MulticlassDiabetesDataset.csv`. : La variable “Class” representa el verdadero diagnóstico (la clase 0 significa Normal, la 1 Prediabetes y la 2 Diabetes). Será la variable a clasificar en esta tarea.
2. La base de datos utilizada está basada en <https://www.kaggle.com/datasets/yasserhessein/multiclass-diabetes-dataset>. Leer la documentación para interpretar correctamente las variables.
3. Reportar la cantidad de muestras por clase.
4. Utilice un `pairplot` (seaborn). Explicar que significa dicho gráfico. Justifique por que la característica HbA1c es clave en la clasificación.
5. Utilice el comando `train_test_split` (sklearn) para definir dos conjuntos de datos. El conjunto de entrenamiento debe contener el 80 % de las muestras, el resto serán de testeo.

(b) Análisis de discriminante:

1. Implementar una clase `LDA_QDA`, que funcione para cualquier cantidad de características. El código debe estar estructurado de la siguiente manera.


```
class LDA_QDA:
    # Inicializar atributos y declarar hiperparámetros.
    def __init__(self,...,LDA=False): #LDA selecciona entre LDA y QDA.

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

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

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

    #Alternativa práctica para el testeo soft
    def predict_discriminant(self,X)
```

2. Entrenar un clasificador LDA utilizando solamente la característica HbA1c. Repetir para un clasificador QDA.
3. Para cada algoritmo graficar la función discriminante para cada clase e indicar las fronteras de decisión. Comparar resultados.
4. Reportar *accuracy*, matriz de confusión y macro-F1 utilizando los datos de testeo. ¿Por qué dan tan diferentes el *accuracy* y la F1?
5. Repetir los incisos **2.**, **3.** y **4.** incorporando el BMI como característica. : Para el análisis gráfico se recomienda utilizar curvas de nivel.
6. Repetir los incisos **2.** y **4.** utilizando todas las características.

(c) Vecinos más cercanos:

1. Implementar una clase KNN, que funcione para cualquier cantidad de características. 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):

```

2. Entrenar un clasificador 9NN utilizando solamente la característica HbA1c.
3. Graficar $P(y|x)$ para cada clase e indicar las fronteras de decisión.
4. Reportar *accuracy*, matriz de confusión y macro-F1 utilizando los datos de testeo.
5. Repetir los incisos **2.** y **4.** incorporando el BMI como característica. Graficar la frontera de decisión.
6. Repetir los incisos **2.** y **4.** utilizando todas las características.