 **INSTITUTO POLITÉCNICO NACIONAL**

**ESCUELA SUPERIOR DE CÓMPUTO**

Ejercicio de Lab9: SMOTE y Perceptrón Simple

**Alumno:** Delgado Acosta Luis Bernardo

Franco Calderas Sergio Alberto

**Grupo**: 5BV1

**SMOTE**

SMOTE (Synthetic Minority Over-sampling Technique) es una técnica de sobremuestreo diseñada para abordar el desequilibrio de clases en conjuntos de datos de clasificación. El desequilibrio de clases ocurre cuando una clase tiene una cantidad significativamente mayor de ejemplos que otra, lo que puede llevar a que los modelos de aprendizaje automático tengan un sesgo hacia la clase mayoritaria y no aprendan correctamente la clase minoritaria. SMOTE genera muestras sintéticas de la clase minoritaria en lugar de simplemente duplicar ejemplos existentes. Utiliza una técnica de interpolación entre los ejemplos existentes de la clase minoritaria.

Pasos de SMOTE:

1. Primero se identifica cuál es la clase minoritaria
2. Para cada ejemplo en la clase minoritaria, se buscan sus k vecinos más cercanos dentro de la misma clase.
3. Se crea nuevas muestras sintéticas interpolando entre el ejemplo y sus vecinos más cercanos. Esta interpolación se realiza con un factor aleatorio, que controla el grado de interpolación entre los puntos.
4. Por último, las nuevas muestras sintéticas se agregan al conjunto de datos original, aumentando el número de ejemplos de la clase minoritaria.

**Implementación:**

Para esta práctica se utilizó el dataset de ‘glass’ uno de los datasets que tiene desequilibrio de clases, lo que es importante para probar la función de ‘smote’.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

*Fig.1 Identificación de la clase minoritaria*

**Texto

Descripción generada automáticamente**

*Fig.2 Generación de muestras sintéticas*

*Texto

Descripción generada automáticamente*

*Fig.3 Combinación de los datos sintéticos con los originales*

**Resultados:**

Aplicamos dos clasificadores, el primero es el knn con k=10 y el segundo es el 1nn. Y los métodos de validación son ‘Hold-Out’ y ’10-Fold Cross’.

--- Validación en glass.csv ---

Before Counter({2: 53, 1: 53, 7: 20, 5: 10, 3: 10, 6: 4})

After SMOTE: Counter({2: 53, 1: 53, 6: 53, 7: 20, 5: 10, 3: 10})

---- KNN ----

Hold-Out (70/30 Estratificado): (SIN SMOTE)

Accuracy: 0.9365079365079365

---- 1NN ----

Hold-Out (70/30 Estratificado): (SIN SMOTE)

Accuracy: 0.9682539682539683

---- KNN ----

Hold-Out (70/30 Estratificado): (SMOTE)

Accuracy: 0.9523809523809523

---- 1NN ----

Hold-Out (70/30 Estratificado): (SMOTE)

Accuracy: 0.9682539682539683

Before Counter({2: 76, 1: 69, 3: 17, 5: 13, 6: 9, 7: 8})

After Counter({2: 76, 7: 76, 1: 69, 3: 17, 5: 13, 6: 9})

---- KNN ----

10-Fold Cross-Validation Estratificado: (SIN SMOTE)

Accuracy: 0.8262910798122066

---- 1NN ----

10-Fold Cross-Validation Estratificado: (SIN SMOTE)

Accuracy: 0.8967136150234741

---- KNN ----

10-Fold Cross-Validation Estratificado: (SMOTE)

Accuracy: 0.8450704225352113

---- 1NN ----

10-Fold Cross-Validation Estratificado: (SMOTE)

Accuracy: 0.892018779342723

En el KNN (sin SMOTE): La precisión alcanza un 93.65%, con una matriz de confusión que muestra un buen desempeño, aunque se observa una ligera confusión en algunas clases menores, especialmente en las clases 5 y 6.

1NN (sin SMOTE): La precisión es ligeramente superior a la de KNN, alcanzando 96.83%, con una matriz de confusión similar, pero con un mejor manejo de la clase 5, donde se cometen menos errores.

KNN (con SMOTE): La precisión incrementa a 95.24%, el aumento es pequeño contemplando que ya no hay desbalance de clases.

1NN (con SMOTE): Permanece igual que en el caso sin SMOTE, lo que indica que SMOTE no parece mejorar significativamente el rendimiento del modelo 1NN en este escenario.

SMOTE no tiene un impacto significativo, esto es provocado porque el modelo de clasificación ya es bueno, no requiere que las clases estén relativamente equilibradas.

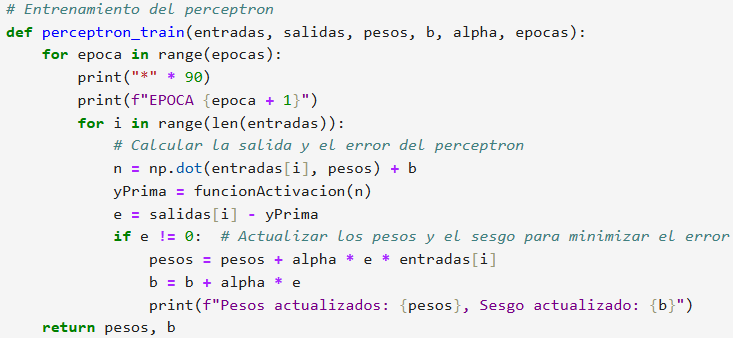
**Perceptrón Simple**

El perceptrón es una neurona artificial inspirada en el cerebro humano, el cual puede ser utilizado como un clasificador binario, es decir problemas en los que se busca clasificar elementos de dos categorías. El modelo del perceptrón consta de tres elementos, una capa de entrada en donde se envía la información a las neuronas de las capas siguientes, en la capa oculta se hacen los cálculos para ajustar los valores de los pesos y el sesgo que se encargan de hacer la clasificación de los datos, se hace una suma ponderada de la multiplicación de los pesos por los valores de los datos en la capa de entrada y se suma el valor del sesgo, en caso de que el error de la clasificación sea diferente de cero se tiene que actualizar los pesos y el sesgo para minimizar el error, en la capa de salida se aplica una función de activación para hacer la clasificación de los valores resultante de la capa oculta. Estos pasos para hacer la actualización de los pesos y el sesgo se hacen durante varias épocas para asegurarse de que el error se logre minimizar hasta hacer la clasificación de los datos correctamente.

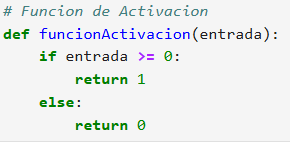
Para esta practica se hizo la clasificación del dataset iris plant con dos clases en lugar de tres, la clase iris setosa e iris virginica. Para calcular el rendimiento del perceptrón se realizó la validación con el metodo Hold-out 70/30, de igual manera se obtuvo su matriz de confusión y el valor del accuracy.

**Implementación:**

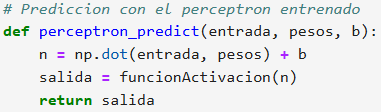
A continuación, se muestran las funciones para la implementación del perceptrón simple y los resultados de la clasificación del dataset iris plant con dos clases.



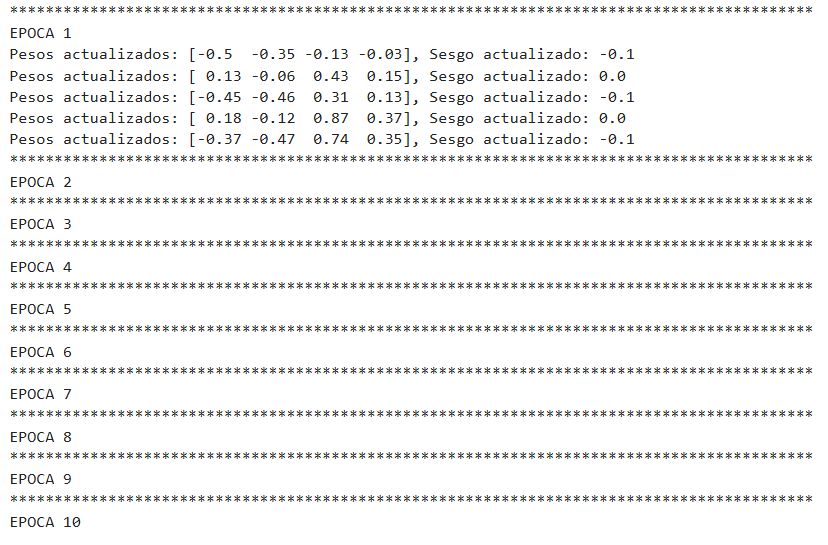
*Fig.4 Función para el entrenamiento del perceptrón*



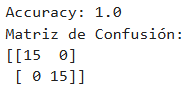
*Fig.5 Función de activación escalón*



*Fig.6 Función para hacer la predicción de los datos*



*Fig.7 Actualización de los pesos y sesgo durante 10 épocas*



*Fig.8 Matriz de confusión y accuracy resultante*

Enlaces a los repositorios

Bernardo: <https://github.com/BernardoD07/Machine-Learning-ESCOM.git>

Sergio: https://github.com/SergioCalderas/Trabajos-Machine-Learning.git