Apéndice D: Manual Técnico

Práctica 1: Implementación del algoritmo de clasificación K-NN

1. Descripción general del sistema

El sistema desarrollado implementa el algoritmo de clasificación **K-Nearest Neighbors (K-NN)** aplicado al conjunto de datos Iris. El objetivo principal del programa es permitir la evaluación del modelo para diferentes valores de K y la clasificación de nuevos objetos sin etiqueta.

El sistema permite:

- Cargar archivos de entrenamiento, prueba y datos desconocidos.
- Ejecutar el algoritmo K-NN con un valor de K definido por el usuario.
- Calcular la exactitud del modelo (si se proporciona conjunto de prueba).
- Clasificar datos desconocidos y exportar los resultados en formato Excel.

2. Implementación en Python

2.1. Requerimientos de software

Para la ejecución del programa se requiere:

- **Python 3.x**
- Sistema operativo Windows, Linux o macOS
- Editor/IDE recomendado: Visual Studio Code, PyCharm o similar

2.2. Bibliotecas utilizadas

El programa utiliza bibliotecas estándar y externas de Python:

- **math**: operaciones matemáticas, principalmente raíz cuadrada.
- **tkinter**: construcción de la interfaz gráfica (GUI) y carga de archivos.
- **openpyxl**: generación de archivos Excel (`.xlsx`) con resultados.

2.3. Estructura de archivos de entrada

El programa trabaja con tres archivos principales en formato `.data` o `.txt`:

2.3.1. Archivo de entrenamiento

`DataTrained-iris.data`
Contiene registros con cuatro atributos numéricos y una clase al final. Formato general:

```
atributo1, atributo2, atributo3, atributo4, clase
```

2.3.2. Archivo de prueba

`TestData-iris.data`
Contiene registros con el mismo formato que el archivo de entrenamiento. Se utiliza para evaluar el modelo y calcular la exactitud.

2.3.3. Archivo de datos nuevos

`NewData-iris.data`
Contiene registros sin etiqueta de clase. Formato general:

```
atributo1, atributo2, atributo3, atributo4
```

2.4. Descripción funcional del programa

El funcionamiento del sistema se organiza en módulos lógicos:

2.4.1. Lectura y carga de datos

El sistema permite cargar los archivos mediante la interfaz gráfica. Los datos se leen línea por línea, separando atributos numéricos y clases (cuando existen). Los atributos se convierten a valores numéricos para permitir el cálculo de distancias.

2.4.2. Cálculo de distancias

Para clasificar un objeto, el sistema calcula su distancia respecto a todos los registros del conjunto de entrenamiento utilizando la distancia euclidiana.

2.4.3. Selección de vecinos y asignación de clase

Las distancias calculadas se ordenan de menor a mayor y se seleccionan los K registros más cercanos. La clase final se asigna mediante votación mayoritaria.

2.4.4. Evaluación del modelo

Si se proporciona un conjunto de prueba, el sistema compara la clase predicha con la clase real y calcula la exactitud del modelo para el valor de K ingresado.

2.4.5. Exportación de resultados

El sistema genera automáticamente un archivo Excel con los resultados de clasificación para los datos desconocidos. Este archivo incluye:

- El objeto (atributos)
- La clase asignada

2.5. Archivos de salida generados

Dependiendo de la ejecución, el sistema puede generar archivos como:

- `Set-Prueba-K357.xlsx`: tabla de resultados para objetos de prueba comparando $K = 3, 5, 7$.
- `New-Best-K.xlsx`: clasificación final de los datos desconocidos con el mejor valor de K.

2.6. Consideraciones técnicas

- El valor de K debe ser un entero positivo.
- Para evitar errores de clasificación, el valor de K debe ser menor o igual al número de objetos en el conjunto de entrenamiento.
- Los archivos deben respetar el formato de separación por comas y contener valores numéricos válidos.

3. Uso técnico de WEKA (Herramienta de minería de datos)

Para la validación y comparación de resultados se utilizó la herramienta de minería de datos **WEKA**, la cual proporciona implementaciones estandarizadas de diversos algoritmos de aprendizaje automático. En esta práctica se empleó el algoritmo K-NN mediante el clasificador `IBk`, con el objetivo de contrastar su desempeño frente a la implementación desarrollada en Python.

3.1. Requerimientos

Para el uso de WEKA fue necesario contar con:

- Java instalado en el sistema (JRE o JDK).
- WEKA en su versión de escritorio, instalada y funcional.

3.2. Configuración del clasificador

Dentro de WEKA se utilizó el módulo **Explorer**. En la sección *Classify* se seleccionó:

- **Classifier:** `lazy ->IBk`
- **Parámetro principal:** número de vecinos más cercanos (`k`)

Se realizaron ejecuciones independientes utilizando los valores:

$$K = 3,\ 5,\ 7$$

manteniendo como medida de distancia la distancia euclidiana, de forma equivalente a la implementación en Python.

3.3. Carga de datos

El conjunto de datos Iris fue cargado en WEKA desde un archivo compatible (por ejemplo, formato `.arff`). Se verificó que el atributo de clase estuviera correctamente definido como la última columna, garantizando la correcta interpretación de etiquetas durante la clasificación.

3.4. Método de evaluación

Para evaluar el desempeño del clasificador se utilizó un conjunto de prueba equivalente al archivo `TestData-iris.data` empleado en Python. WEKA generó automáticamente métricas como:

- Exactitud del modelo (*Correctly Classified Instances*)
- Matriz de confusión
- Métricas por clase (precisión, recall y medida F)

3.5. Salida de resultados

Los resultados fueron exportados en archivos de texto para su análisis y comparación. En particular:

- `Weka_Output.txt`: resultados del clasificador evaluado con el conjunto de prueba.
- `Weka_Output_NewData.txt`: resultados de la clasificación de objetos desconocidos.

Estos reportes incluyen métricas de desempeño y clases predichas, y fueron utilizados para realizar la comparación directa con los resultados obtenidos mediante la implementación en Python.