MODELADO DE SISTEMAS SOFTWARE

Informe: Práctica 6

Héctor Luís
Mariño Fernández
alu0100595604

ÍNDICE

1. Objetivos

1. Modelar mediante diagramas UML los procesos clave del sistema de clasificación k-NN:
    - Actividades de carga, selección de vecinos, votación, generación de splits y evaluación.
    - Diagrama de estado para el preprocesado de datos.
2. Implementar en Java la generación reproducible de conjuntos de entrenamiento y prueba.
3. Organizar la práctica en un repositorio independiente (PRC-6).

---

2. Diagramas UML

- **Situados en el Repositorio publico de la practica:** https://github.com/HectorLMF/Modelado-PRC6
- **De actividad:** https://github.com/HectorLMF/Modelado-PRC6/tree/main/docs/diagrams/De%20Actividad
- **De Estado:** https://github.com/HectorLMF/Modelado-PRC6/tree/main/docs/diagrams/De%20estado

3. Implementación Java: `ExperimentManager`

Clase en el repositorio:
https://github.com/HectorLMF/Modelado-PRC6/blob/main/src/experiment/ExperimentManager.java

La clase `ExperimentManager` dispone de los siguientes **atributos** internos:

- `Dataset dataset`: Referencia al dataset original con instancias y metadatos.
- `List<Instance> trainSet`: Lista de instancias asignadas al conjunto de entrenamiento.
- `List<Instance> testSet`: Lista de instancias asignadas al conjunto de prueba.

Y los siguientes **métodos públicos** clave:

1. **`public ExperimentManager(Dataset dataset)`**
   - **Función**: Constructor que inicializa la referencia al dataset y crea listas vacías para `trainSet` y `testSet`.
2. **`public void splitDatasetRatio(float testRatio, boolean random, int seed)`**
   - **Parámetros**:
     - `testRatio`: porcentaje del dataset que se dedicará a prueba (valor entre 0 y 1).
     - `random`: si `true`, mezcla aleatoriamente las instancias antes de dividir.
     - `seed`: semilla para el generador aleatorio (asegura reproducibilidad).
   - **Funcionamiento**:
     - Duplica todas las instancias del dataset en una lista local.
     - Calcula el tamaño de la parte de prueba (`testSize = total * testRatio`).
     - Si `random` es `true`, aplica `Collections.shuffle` con un `Random(seed)`.
     - Asigna las primeras `(total - testSize)` instancias a `trainSet` y el resto a `testSet`.
3. **`public String[] saveSplit()`**
   - **Función**: Guarda `trainSet` y `testSet` en archivos CSV dentro de `datasets/` con nombre basado en timestamp.
   - **Retorno**: Un arreglo de dos rutas `[pathTrain, pathTest]`.
   - **Interno**: Invoca al método privado `saveDataset` para escribir cada lista.
4. **`private void saveDataset(List<Instance> list, String path)`**
   - **Parámetros**:
     - `list`: lista de instancias a guardar.
     - `path`: ruta y nombre del archivo de salida.
   - **Funcionamiento**:
     - Escribe la línea de cabeceras obteniendo los nombres de atributos desde `dataset.getAttributes()`.

        - Recorre cada `Instance` y escribe los valores separados por comas.
        - Maneja `IOException` imprimiendo un mensaje de error si ocurre.
5. **`public void runExperiment(KNNClassifier clf, String originalDatasetPath)`**
    - **Parámetros**:
        - `clf`: objeto `KNNClassifier` configurado previamente.
        - `originalDatasetPath`: ruta del archivo CSV original para referencia en el informe.
    - **Funcionamiento**:
        - Inicializa contadores para calcular precisión y una matriz de confusión de tamaño `[nClases][nClases]`.
        - Recorre `testSet`, clasifica cada instancia y actualiza contadores y matriz.
        - Calcula `accuracy = correct / total`.
        - Imprime por consola la matriz de confusión y la precisión.
        - Pregunta al usuario si desea guardar los datasets de train/test (`saveSplit`).
        - Llama a `generateExperimentReport` para crear un archivo de texto con todos los detalles.
6. **`private void generateExperimentReport(...)`**
    - **Función**: Construye y guarda en `experiments_output/` un informe con:
        - Fecha y hora.
        - Ruta del CSV original y de los splits (si se guardaron).
        - Parámetros del clasificador (`k`, métrica, ponderación, regla).
        - Precisión y matriz de confusión.
    - **Manejo de I/O**: Crea el directorio si no existe y captura excepciones de escritura.

Con esto, `ExperimentManager` ofrece una API sencilla para:

- Dividir datasets (con o sin aleatoriedad).
- Persistir conjuntos de entrenamiento/prueba.
- Ejecutar y documentar experimentos KNN.