MODELADO DE SISTEMAS SOFTWARE

Informe: Práctica 5

Héctor Luís
Mariño Fernández
alu0100595604

ÍNDICE

1. Introducción al Proyecto

El presente documento describe el desarrollo de un **clasificador k-NN** con configuración avanzada, permitiendo a los usuarios seleccionar diversos parámetros como la **métrica de distancia**, el **pesado de vecinos**, la **regla de clasificación**, y la **normalización de datos**. Además, el sistema es capaz de **almacenar y replicar experimentos** mediante archivos generados dinámicamente.

Este clasificador permite analizar **datasets tabulares** en formato `.csv`, procesando sus atributos y ejecutando comparaciones con instancias existentes. Como resultado, se obtiene una **predicción de clase** basada en el comportamiento de los vecinos más cercanos según la configuración definida por el usuario.

2. Enlaces al Proyecto en GitHub

El código fuente del proyecto y documentación detallada se encuentran disponibles en GitHub:

- **Repositorio principal:** https://github.com/HectorLMF/Modelado---PRC-5

3. Diagrama de Clases y de Casos de uso

El proyecto se estructura en **diferentes paquetes**, organizados según la funcionalidad de cada módulo. A continuación, se presentan los **diagramas de clases y casos de uso en varios formatos**, que ilustra la relación entre los componentes principales del sistema, asi como su interaccion con el usuario..

Ubicación: (en el repositorio) `docs/diagrams`

4. Explicación de Clases, Atributos y Métodos

🔹 Paquete model (Gestión de Datos)

Define la estructura de los datos utilizados en el clasificador.

Attribute

Clase que representa un **atributo** del dataset, incluyendo su tipo y si es la etiqueta de clase.

- **Atributos:**
  - `name` (`String`): Nombre del atributo.
  - `quantitative` (`boolean`): Indica si es numérico (`true`) o categórico (`false`).
  - `isClass` (`boolean`): Indica si es el atributo de clase (`true`).
- **Métodos:**
  - `getName()`: Devuelve el nombre del atributo.
  - `isQuantitative()`: Devuelve `true` si el atributo es numérico.
  - `isClass()`: Indica si el atributo es la etiqueta de clase.

`Instance`

Clase que representa una **instancia de datos** con sus valores.

- **Atributos:**
  - `values` (`List<Object>`): Lista de valores asociados a la instancia.
- **Métodos:**
  - `getValue(int index)`: Obtiene el valor en el índice especificado.
  - `setValue(int index, Object value)`: Asigna un nuevo valor al índice.
  - `getValues()`: Devuelve todos los valores de la instancia.

`Dataset`

Clase que almacena y administra el **conjunto de datos**.

- **Atributos:**
  - `data` (`List<Instance>`): Lista de instancias.
  - `attributes` (`List<Attribute>`): Lista de atributos.
  - `classAttributeIndex` (`int`): Índice del atributo de clase.
  - `normalizationMode` (`NormalizationMode`): Modo de normalización.
- **Métodos:**
  - `addInstance(Instance instance)`: Añade una instancia.
  - `normalize(NormalizationMode mode)`: Normaliza los atributos predictivos.
  - `getAttributeStats()`: Retorna estadísticas de atributos.

🔹 Paquete classification (Reglas de Clasificación)

Define cómo se votan las clases de los vecinos.

`ClassificationRule`

Interfaz que define una regla de clasificación.

`MajorityVote`

Clasifica con **mayoría simple**: la clase con más votos gana.

`ThresholdVote`

Clasifica según un **umbral mínimo** de votos.

- **Atributos:**
  - `threshold` (`float`): Porcentaje mínimo requerido para aceptar una clase.
- **Métodos:**
  - `predictClass(Map<String, Float> votes)`: Predice la clase basándose en el umbral.

🔹 Paquete `distance` (Métricas de Distancia)

`DistanceMetric`

Interfaz para definir una **métrica de distancia**.

`EuclideanDistance`

Implementa la distancia **Euclidiana**.

`ManhattanDistance`

Implementa la distancia **Manhattan**.

`ChebyshevDistance`

Implementa la distancia **Chebyshev**.

🔹 Paquete `weighting` (Pesado de Vecinos)

`CaseWeightingStrategy`

Interfaz para definir estrategias de **pesado de vecinos**.

`EqualWeighting`

Todos los vecinos tienen el mismo peso.

`DistanceInverseWeighting`

El peso es **inversamente proporcional a la distancia**.

`RankBasedWeighting`

Asigna pesos decrecientes según el orden del vecino.

🔹 Paquete `knn` (Clasificador k-NN)

`KNNClassifier`

Implementa el algoritmo **k-NN** con configuración avanzada.

- **Atributos:**
  - `k`: Número de vecinos considerados.
  - `distanceMetric`: Métrica de distancia usada.
  - `classificationRule`: Algoritmo de votación.
  - `caseWeightingStrategy`: Estrategia de pesado de vecinos.
- **Métodos:**
  - `classify(Instance instance)`: Predice la clase de una instancia.
  - `setK(int k)`: Ajusta el valor de `k`.

🔹 Paquete `experiment` (Gestión de Experimentos)

`ExperimentManager`

Clase que gestiona la ejecución y almacenamiento de **experimentos**.

- **Métodos:**
  - `splitDatasetRatio(float testRatio, boolean random, int seed)`: Divide el dataset.
  - `saveSplit()`: Guarda los conjuntos en `datasets/`.
  - `runExperiment(KNNClassifier classifier, String datasetPath)`: Ejecuta el experimento.
  - `generateExperimentReport()`: Genera un informe en `experiments_output/`.

5. Funcionamiento del Método Main

El método `main` es el punto de entrada del sistema y se encarga de interactuar con el usuario mediante un menú. A continuación se describe el flujo de ejecución:

1. **Menú Interactivo:** Se presenta un menú principal con opciones como:
    - Cargar un dataset (con validación de ruta para asegurar que el archivo exista).
    - Mostrar la información del dataset (incluyendo número de instancias, atributos y estadísticas).
    - Configurar el clasificador k-NN (solicitando al usuario el valor de *k*, la métrica de distancia, la estrategia de ponderación y el algoritmo de votación).
    - Clasificar una nueva instancia (permitiendo al usuario introducir valores separados por comas).
    - Ejecutar un experimento (dividiendo el dataset, ejecutando la clasificación sobre un conjunto de prueba, generando un informe con la configuración y la matriz de confusión).
    - Salir del programa.
2. **Carga y Validación del Dataset:** Se solicita al usuario ingresar la ruta del archivo CSV. El menú valida la existencia y formato del archivo antes de cargarlo. Además, se solicita seleccionar un modo de normalización para preprocesar los datos.
3. **Configuración del Clasificador k-NN:** Se recogen parámetros clave del usuario:
    - El valor de *k* se ingresa y se valida.
    - Se elige la métrica de distancia entre opciones predefinidas (Euclidiana, Manhattan o Chebyshev).
    - Se selecciona la estrategia de ponderación (igual, inversa o basada en ranking).
    - Se elige el algoritmo de votación, que puede ser por mayoría o basado en un umbral. Con estos datos, se instancia el `KNNClassifier`.
4. **Clasificación de Nuevas Instancias:** El usuario ingresa una línea de valores (sin la etiqueta de clase) y el sistema procesa la entrada para crear una `Instance`. El clasificador predice la clase correspondiente y muestra el resultado.
5. **Ejecución de Experimentos:** Se permite dividir el dataset de forma configurable (porcentaje para el conjunto de pruebas, opción aleatoria y semilla) y se ejecuta el experimento a través de la clase `ExperimentManager`. Este genera un informe que se guarda automáticamente, conteniendo desde la ruta del dataset original hasta los parámetros de configuración y la matriz de confusión.
6. **Salida:** La opción de salir finaliza el bucle interactivo y cierra la aplicación.

6. Uso de Interfaces y Enumeraciones

El diseño del sistema hace un extenso uso de interfaces y enumeraciones para lograr modularidad y flexibilidad:

- **Interfaces:** Las interfaces como `DistanceMetric`, `ClassificationRule` y `CaseWeightingStrategy` definen contratos que deben cumplir las implementaciones concretas. Esto permite que el clasificador k-NN se configure dinámicamente en función de las elecciones del usuario, facilitando la extensión o modificación de funcionalidades sin alterar el código base.
- **Enumeraciones:** La enumeración `NormalizationMode` define los modos de normalización (RAW, MIN_MAX, Z_SCORE), asegurando que la normalización se realice de manera consistente y que sólo valores válidos sean usados en el sistema. Esto mejora la claridad y la robustez del código.