Universidad Nacional de San Martín

ALGORITMOS 2

LICENCIATURA EN CIENCIA DE DATOS

Trabajo Práctico Integrador

Versión	Fecha	Descripción
1.0	31/03/2024	Versión inicial

INDICE

Introducción

El trabajo integrador de la materia reúne conceptos de los temas vistos durante la cursada. Se parte de un problema a resolver para realizar un análisis inicial, proponiendo un diseño parcial del mismo a partir de conceptos de ingeniería de software conocidos y finalmente implementando la solución en un lenguaje orientado a objetos. El desarrollo del trabajo se establece en etapas de entregas parciales para facilitar el seguimiento y corrección a lo largo del cuatrimestre.

El ciclo de vida del desarrollo de software se compone de varias fases, las cuales pueden variar según la metodología y modelo de procesos adoptado. En este trabajo se hace foco en la fase de **implementación**, apoyada previamente con **documentación mínima de análisis y diseño**. Se construirá un sistema desde sus inicios, produciendo así documentación que permita especificar suficientemente no sólo sus requerimientos sino también la solución propuesta a partir de diagramas, asimilando las decisiones y reflexiones que se desprenden del proceso.

Objetivo

El trabajo práctico integrador busca afianzar los conceptos vistos durante la materia a través del análisis, diseño e implementación en el lenguaje **Python** de un problema determinado. Cada una de las fases del ciclo de vida de desarrollo mencionadas requiere ciertos entregables que serán evaluados acorde a los criterios definidos en este documento.

Cronograma de entregas

Las fechas de entrega parcial se plantean para realizar el seguimiento activo del trabajo, por lo cual no son obligatorias, pero impactan en el aspecto de organización del grupo. La entrega final comprende a todos los entregables incluidos en este trabajo práctico integrador, el cual debe presentarse y defenderse en clase.

Fecha	Tipo de entrega	Entregables
16/04	Seguimiento	Documentación de Análisis
10/05	Seguimiento	Documentación de Diseño
07/06 y 14/06	Final	Todos

Formato de entrega

Se debe presentar el trabajo completo en **formato digital**. No existe una estructura predefinida requerida, es posible apoyarse en diferentes estándares o herramientas que vimos durante la materia. Para cada entregable, se plantean algunas sugerencias que pueden seguir.

Es requisito que se incluyan los contenidos mínimos solicitados para cada *entregable*.

Criterios de evaluación

Utilizaremos una evaluación basada en rúbricas donde se describen los distintos aspectos a considerar. La evaluación final del trabajo se descompone en dos partes:

a) Evaluación grupal sobre la elaboración de los entregables.

Aspectos \ Nivel	1 - Regular	2 - Bueno	3 - Muy Bueno	4 - Excelente
Análisis del problema (no se evalúa en este cuatrimestre)	No se interpreta correctamente el problema o no se presenta documentación vinculada al proceso de análisis.	Se presenta documentación suficiente de análisis acorde a lo solicitado en los entregables.	La documentación de análisis está completa y argumentada, relacionando temas vistos en clase.	El desarrollo de la documentación evidencia lectura de la bibliografía y emplea investigación adicional para mejorar el análisis.

Diseño de la solución basado en POO	No se presenta documentación del diseño acorde al análisis del problema, o está incompleta.	La documentación de diseño representa una solución básica en POO al problema dado.	La documentación de diseño está completa, incorpora conceptos de diseño, y favorece la extensibilidad y mantenibilidad.	Se hace uso de patrones de diseño, incluye argumentación y reflexión sobre principios, y contempla diversos atributos de calidad.
Implementación de la solución	El código implementado no satisface el problema completo, no compila, o arroja errores inesperados.	El código implementado resuelve el problema y no presenta errores en tiempo de compilación y ejecución.	El código implementado incorpora buenas prácticas vistas en clase y está alineado al diseño propuesto.	El desarrollo evidencia conocimiento profundo sobre el lenguaje, contempla el atributo de eficiencia, o incluye pruebas unitarias.
Organización y Presentación	Se entrega fuera de término o la documentación no tiene formato consistente.	Los entregables tienen un formato consistente y se prepara presentación para la defensa.	Se utilizan herramientas para facilitar la colaboración, versionado y organización de la documentación y código.	Se utilizan herramientas para gestionar el ciclo de vida del trabajo.

b) Evaluación individual sobre la participación durante las prácticas y en la defensa del trabajo en la fecha de presentación.

Aspectos \ Nivel	1 - Regular	2 - Bueno	3 - Muy Bueno	4 - Excelente
Participación en el grupo	No se evidencia participación dentro del grupo durante la elaboración del trabajo.	Se plantean ideas o se escucha con atención a los demás miembros del equipo.	Se participa activamente en las discusiones y en la resolución del trabajo.	Contribuye de forma proactiva en la resolución del trabajo y fomenta la colaboración en el grupo.
Defensa del trabajo	No asiste a la fecha de defensa del trabajo o no participa durante la presentación.	Expone con claridad su rol y aporte realizado al trabajo final.	Responde adecuadamente con seguridad las preguntas realizadas durante la defensa.	Relaciona conceptos vistos durante la materia, reflexionando y argumentando decisiones de diseño.

Entregables del trabajo

En esta sección se describe todo lo que se requiere entregar para evaluar el trabajo grupal, comprendiendo documentación solicitada, código de implementación y material referido a la presentación. En cada caso se describen contenidos mínimos y en ciertos casos algunos sugeridos.

Documentación de Análisis

Se solicita disponer de una especificación parcial, ya que no es el foco de evaluación en esta materia. Deberá estar **corregida y consensuada** entre el grupo y docentes.

Contenidos mínimos requeridos:

- Objetivo y Alcance
- Descripción de alto nivel del sistema
- Requerimientos funcionales más relevantes
- Requerimientos no funcionales

Documentación de Diseño

El sistema debe ser diseñado bajo el **paradigma orientado a objetos**, contemplando los principios y conceptos vistos en la materia. Se deben tener presente los siguientes atributos de calidad:

- **Extensibilidad**: Es necesario que el modelo pueda extenderse fácilmente para incorporar nueva funcionalidad según sea necesario.
- **Mantenibilidad**: Hacer uso de los principios de POO para construir un producto que sea de fácil mantenimiento, por ejemplo evitando la duplicación de código y desacoplando los subsistemas cuando sea posible.
- **Gestión de errores**: El sistema debe capturar todos los errores posibles en tiempo de ejecución y clasificarlos según corresponda con una jerarquía de excepciones.

En primer lugar, este documento debe incluir una **vista general estática** a partir de un **diagrama de clases** que permitirá identificar todas las entidades del sistema con sus relaciones correspondientes, aunque no es necesario incorporar todos los atributos y métodos de las mismas.

En segundo lugar, se diseñarán **vistas adicionales** que deberán surgir de al menos un módulo, componente o proceso crítico para el sistema, decisión que tendrá que estar justificada. A excepción de una vista estática general, el análisis se suele realizar a bajo nivel para describir la estructura y comportamiento de lo considerado complejo de entender dentro del sistema.

Consideraciones importantes al momento de diseñar vistas:

- Las vistas deben ser consistentes entre ellas
- Una vista se compone de diagrama y descripción asociada
- La descripción de una vista puede incluir:

    - Detalles sobre los elementos y conexiones relevantes
    - Estilos arquitectónicos o patrones utilizados
    - Justificación de decisiones de diseño involucradas (si aplica)

Contenidos mínimos requeridos:

- Vista estática del sistema (diagrama de clases general)

Contenidos opcionales:

- Diagrama de clases de un módulo o subsistema relevante
- Diagrama de secuencia y su diagrama de clases asociado
- Diagrama de estados y su diagrama de clases asociado

Código fuente

Entrega de la implementación en código fuente en lenguaje Python 3. Es deseable el uso de anotaciones de tipos (type hints) y la entrega de la documentación de uso asociada, es posible utilizar herramientas de generación automática como Sphinx o EpyDoc. También es muy valorada la incorporación de pruebas unitarias, pudiendo utilizar herramientas como pytest.

Contenidos mínimos requeridos:

- El código cumple con la funcionalidad requerida
- La ejecución no arroja excepciones inesperadas

Presentación

Si bien es en sí no un entregable, la defensa del trabajo se realizará durante la presentación del mismo, en la cual pueden generar contenido multimedia de apoyo. En esta instancia se debe demostrar el funcionamiento del producto construido con algún caso de uso.
Pueden incluirse dentro de la entrega todo contenido generado para la presentación, como así también documentación o artefactos relacionados para mostrar el funcionamiento.

Contenidos recomendados:

- Presentación de integrantes y responsabilidades dentro del trabajo
- Breve descripción del análisis del problema
- Detalle de la solución diseñada
- Problemas encontrados durante el desarrollo
- Investigación asociada para resolver los problemas
- Posibles mejoras pendientes
- Estrategias para incorporar nueva funcionalidad
- Demostración de la funcionalidad implementada con un caso de uso
- Conclusiones

Enunciado del problema

Se desea construir una librería que permita construir modelos predictivos a través de una versión simplificada del ensamble RandomForest [1]. El objetivo es entrenar un modelo con algún conjunto de datos etiquetados (aprendizaje supervisado) de forma que luego permita predecir la variable objetivo. En principio nos enfocaremos en **problemas de clasificación**, pudiendo extenderla a problemas de regresión eventualmente.

El árbol de decisión

Los árboles de decisión son modelos de aprendizaje automático que se utilizan para clasificación o regresión. Existen varios algoritmos que lo implementan, siendo el más simple el ID3 [2]. Básicamente, es una estructura de árbol donde cada nodo interno representa una *pregunta* sobre una característica (atributo o feature), cada rama representa el resultado de la pregunta, y cada nodo hoja representa una clasificación o valor de predicción.

Algoritmo ID3

El entrenamiento de este modelo consiste en construir la estructura del árbol de la siguiente forma:

1. Selección del atributo: ID3 utiliza la entropía y la ganancia de información para seleccionar el atributo que mejor clasifica los datos en cada nivel del árbol. La entropía es una medida de la impureza en un conjunto de datos, mientras que la ganancia de información mide la reducción en la entropía que resulta de dividir el conjunto de datos en función de un atributo específico.

2. División del conjunto de datos: Una vez seleccionado el atributo, el conjunto de datos se divide en subconjuntos basados en los valores de ese atributo.

3. Construcción del árbol: Se repiten los pasos 1 y 2 recursivamente para cada subconjunto creado en la etapa anterior. Este proceso continúa hasta que se cumple algún criterio de parada y se determina la clase de la hoja con la clase más popular del subconjunto de datos de la misma.

4. Podado del árbol: En algunas implementaciones, se realiza un proceso de poda para evitar el sobreajuste (overfitting) del modelo. La poda implica eliminar subárboles completos que no contribuyen significativamente a la precisión del modelo.

Una vez construido el árbol de decisión, se utiliza para predecir la clasificación de una muestra siguiendo el camino desde la raíz hasta una hoja.

1. Iniciar en la raíz: Comenzamos en el nodo raíz del árbol de decisión.

2. Evaluar la pregunta del nodo: En cada nodo, se evalúa la pregunta asociada al atributo del nodo. Por ejemplo, si el nodo pregunta "¿Es la temperatura mayor que 30 grados?".

3. Seguir la rama apropiada: Seguimos la rama correspondiente a la respuesta de la pregunta para nuestra muestra. Si la respuesta es sí, seguimos la rama izquierda; si es no, seguimos la rama derecha.

4. Repetir hasta llegar a una hoja: Repetimos los pasos 2 y 3 hasta llegar a un nodo hoja. La clase asignada a esa hoja es la predicción para nuestra muestra.

Entropía y Ganancia de información

Una forma de medir el nivel de impureza de una variable es a través de calcular la cantidad de información que tiene utilizando la entropía de Shannon. Asumiendo que tenemos un conjunto de datos clasificados con *n* clases, podemos calcular la **entropía** del conjunto *D* así:

$$Entropia(D) = -\sum_{i=1}^{n} P(C_i) \cdot \log_2(P(C_i))$$

donde $P(C_i)$ es la proporción de observaciones de la clase C_i dentro del conjunto de datos *D*.

Esto nos indica un nivel de impureza de los datos en dicho conjunto respecto a la variable objetivo que toma el valor de una clase. Dado que en cada nodo del árbol de decisión necesitamos separar la muestra S de forma que nos provea la mejor separación posible para clasificar las observaciones, necesitamos averiguar qué atributo nos permite reducir ese nivel de impureza inicial con el que partimos. A esto se lo denomina **ganancia de información**.

La ganancia de información para un atributo se calcula como la diferencia entre la entropía del conjunto de datos antes de la división y la entropía ponderada del conjunto de datos después de la división:

$$IG(A) = Entropia(D) - \sum_{v \in valores(A)} \frac{|D_v|}{|D|} \cdot Entropia(D_v)$$

- *D* es el conjunto de observaciones
- *A* es un atributo
- *valores(A)* es el conjunto de valores posibles del atributo *A*
- D_v es el conjunto de observaciones donde el atributo *A* tiene valor *v*

De esta forma podemos obtener la ganancia de información que ofrece cada atributo y así seleccionar aquel que ofrezca el **mayor valor**.

Criterios de parada

Durante la construcción del árbol podemos detenerla ante distintas situaciones:

- **Nodo puro**: Un nodo se considera puro si todos los ejemplos en ese nodo pertenecen a la misma clase. Cuando se alcanza un nodo puro, no hay necesidad de continuar dividiendo el conjunto de datos en ese nodo, ya que no habría ninguna ganancia adicional de información.

- **Profundidad máxima del árbol**: Se especifica una altura máxima para el árbol de decisión. Cuando se alcanza esta profundidad máxima, se detiene la construcción del árbol. Limitar la profundidad del árbol ayuda a prevenir el sobreajuste y a controlar la complejidad del modelo.

- **Mínimas observaciones por nodo**: Se establece un umbral mínimo para el número de observaciones requeridas en un nodo para continuar dividiéndolo. Si la cantidad es menor que este umbral, se detiene la división y se declara el nodo como una hoja.

- **Ganancia de Información mínima**: Se define un valor mínimo para la ganancia de información que debe obtenerse al dividir un nodo en función de un atributo. Si la ganancia de información obtenida es menor que este valor mínimo, se detiene la división y se declara el nodo como una hoja.

- **Mínimas observaciones por hoja**: Se especifica un número mínimo de observaciones que deben estar presentes en una hoja del árbol. Si la cantidad es menor que este umbral, se detiene la división y se declara al nodo como una hoja.

Podado de árbol

El objetivo de este paso es recortar el árbol entrenado para reducir su complejidad y así evitar el overfitting. Veamos dos métodos.

Reduced Error Pruning

Consiste en eliminar subárboles completos que no contribuyen significativamente a la precisión del modelo. El proceso es el siguiente:

1. Se comienza desde las hojas del árbol entrenado original y se evalúa el efecto de eliminar cada nodo.
2. Para evaluar el efecto de eliminar un nodo, se realiza lo siguiente:
    a. Se calcula el error de clasificación del subárbol sin el nodo que se está considerando podar en un conjunto de validación.
    b. Se compara este error con el error de clasificación del árbol original que contiene el nodo.
    c. Si el error de clasificación del subárbol sin el nodo es menor o igual al error del árbol original, se poda el nodo. Esto significa que la eliminación del nodo no empeora significativamente la precisión del árbol en el conjunto de validación.
3. Después de podar un nodo, se repite el proceso de evaluación del efecto de podar cada nodo interno hasta que no se puedan podar más nodos sin aumentar el error de predicción.

Rule Post Pruning

La poda posterior basada en reglas es una técnica que se centra en identificar y simplificar reglas redundantes o poco útiles generadas por el árbol de decisión. Una regla es una rama del árbol entrenado.

1. Extracción de reglas: A partir del árbol de decisión construido, se extraen todas las reglas posibles que representan las trayectorias desde la raíz hasta cada hoja del árbol (ramas). Cada regla está compuesta por una serie de condiciones basadas en los atributos y sus valores, seguidas de una clase de salida (la predicción).

2. Evaluación de reglas: Se evalúan las reglas extraídas utilizando un conjunto de validación independiente. Se mide la precisión de cada regla en el conjunto de validación y se identifican aquellas reglas que tienen un bajo rendimiento o que son redundantes.

3. Simplificación de reglas: Se simplifican las reglas redundantes o poco útiles para mejorar la generalización del modelo y reducir la complejidad. Esto puede incluir la fusión de reglas similares o la eliminación de condiciones que no contribuyen significativamente a la precisión del modelo en el conjunto de validación.

4. Conjunto final de reglas: Después de simplificar las reglas, se selecciona el conjunto final de reglas que proporciona el mejor rendimiento en el conjunto de validación y se ordenan por poder de predicción para producir las estimaciones.

Algoritmo C4.5

El C4.5 [3] es una extensión del algoritmo ID3 y ambos algoritmos comparten el objetivo de construir árboles de decisión para la clasificación. Introduce varias mejoras con respecto a ID3.

- **Atributos continuos**: Mientras que ID3 sólo puede manejar atributos categóricos, C4.5 puede manejar tanto atributos categóricos como continuos. Para los atributos continuos, se busca el umbral que maximiza la ganancia de información, dividiendo así el atributo en un conjunto de valores discretos.

    1. Ordenamiento de valores: Para un atributo continuo se ordenan todos los valores únicos en orden ascendente.
    2. Selección de umbrales: Se consideran los puntos medios entre cada par de valores consecutivos como posibles umbrales para dividir el atributo. Estos puntos medios actúan como candidatos para separar los valores del atributo en dos grupos. Sólo seleccionar aquellos puntos medios donde la variable objetivo para esa observación cambia entre su previo y siguiente.
    3. Cálculo de la ganancia de información: Para cada posible umbral, se calcula la ganancia de información resultante de dividir el conjunto de datos en dos grupos basados en ese umbral (valor <= umbral y valor > umbral). La ganancia de información se calcula utilizando la entropía de los grupos resultantes.

4. Selección del mejor umbral: El umbral que produce la mayor ganancia de información se selecciona como el umbral óptimo para dividir el atributo continuo en una versión discreta.
5. División del conjunto de datos: Si la ganancia de información del atributo con el umbral óptimo es mayor que el resto de los atributos, se selecciona esta discretización como nodo divisor del árbol y continúa el proceso de construcción recursivo.

- **Missing values**: C4.5 puede manejar valores faltantes en los datos de entrenamiento. Utiliza estrategias como la imputación de valores faltantes o la ponderación de ejemplos para tratar con datos incompletos.
  Algunas estrategias posibles:
  - Asignar el valor más común entre los datos de entrenamiento
  - Asignar el valor más común entre los datos de entrenamiento que tienen la misma clasificación
  - Asignar una probabilidad basada en frecuencias observadas en valores de *A* en el nodo actual

- **Criterio de división**: En lugar de usar solo la ganancia de información como criterio de división, C4.5 utiliza una métrica llamada **Gain Ratio**, que corrige la tendencia de la ganancia de información a favorecer atributos con muchos valores posibles. Penaliza la división de atributos con muchos valores distintos, lo que ayuda a evitar el sobreajuste.

  Se introduce un nuevo concepto llamado **split info** para calcular el gain ratio, que mide la dispersión de los valores del atributo *A* en el conjunto de datos. Se calcula como la entropía del conjunto de datos respecto al atributo *A*.

  $$SplitInfo(A) = -\sum_{v \in valores(A)} \frac{|D_v|}{|D|} \cdot \log_2\left(\frac{|D_v|}{|D|}\right)$$

  Luego el **gain ratio** se calcula utilizando el mismo **information gain** que utiliza ID3.

  $$GR(A) = \frac{IG(A)}{SplitInfo(A)}$$

  El atributo con el **mayor gain ratio** se selecciona como el mejor atributo para dividir el conjunto de datos.

- **Podado del árbol**: C4.5 incorpora un proceso de podado por reglas después de la construcción inicial del árbol. El podado ayuda a evitar el sobreajuste al eliminar las ramas del árbol que no contribuyen significativamente a la precisión del modelo en un conjunto de datos de validación independiente.

- **Costos asimétricos**: Puede manejar costos asimétricos asociados con diferentes tipos de errores de clasificación. Esto es útil en problemas donde los errores de clasificación tienen diferentes consecuencias, como en problemas médicos o financieros.

- **Datos Ponderados**: Permite el uso de datos ponderados para tratar conjuntos de datos desbalanceados, donde algunas clases están representadas en mayor proporción que otras. Esto garantiza que el modelo se ajuste de manera más equilibrada a todas las clases.

Random Forest

La idea simplificada del ensamble Random Forest se apoya en una técnica denominada Bagging (Bootstrap Aggregating) [5] donde se entrenan varios modelos a partir de muestras aleatorias con repetición del conjunto de entrenamiento. Esos modelos luego se combinan para producir una predicción a través del promedio de sus predicciones, si se trata de una regresión, o de la clase más popular (votación), si se trata de una clasificación.

Construcción del bosque

Random Forest construye un conjunto de árboles de decisión durante el proceso de entrenamiento. Cada árbol se construye utilizando un subconjunto aleatorio de muestras del conjunto de datos de entrenamiento y un subconjunto aleatorio de características (atributos).

Bootstrapping (Muestreo con Reemplazo)
Antes de construir cada árbol, se realiza un proceso llamado bootstrapping, que implica muestrear aleatoriamente con reemplazo del conjunto de datos de entrenamiento. Esto significa que algunas muestras pueden aparecer múltiples veces en el conjunto de datos de entrenamiento para un árbol particular, mientras que otras pueden no aparecer en absoluto.

Selección aleatoria de características
Además del muestreo de datos, también se realiza la selección aleatoria de características para cada árbol. En cada división de un árbol, se elige un subconjunto aleatorio de características para considerar como candidatas para la división. Una cantidad de atributos a seleccionar que suele funcionar bien es $\sqrt{A}$, donde *A* es el conjunto de todos los atributos.

Combinación de predicciones
Una vez que se han construido todos los árboles en el bosque, se realizan las predicciones combinando las predicciones de cada árbol individual. Para problemas de clasificación, se utiliza la votación mayoritaria entre los árboles para determinar la clase final de una instancia. Para problemas de regresión, se promedian las predicciones de todos los árboles para obtener el valor final de predicción.

Hiperparámetros

La inclusión de hiperparámetros para los learners que se solicitan **es opcional**. A continuación se proveen algunos que podrían incorporar para configurarlos:

- Selección de algoritmo de árbol: ID3, C4.5
- Criterios de parada:

    - profundidad máxima del árbol
    - número mínimo de observaciones por nodo
    - ganancia de información mínima
    - número mínimo de observaciones por hoja
- Imputación de faltantes
- Cantidad de estimadores del ensamble (árboles del bosque)

Se recomienda utilizar una interfaz de uso similar a la propuesta por la librería scikit-learn. También pueden optar por desarrollar su estimador para que sea compatible con el paquete scikit-learn.

Referencias

[1] Breiman, L. (2001). Random forests. Machine learning, 45, 5-32. https://link.springer.com/article/10.1023/A:1010933404324

[2] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1, 81-106. https://link.springer.com/article/10.1007/BF00116251

[3] Quinlan, J. R. (2014). C4. 5: programs for machine learning. Elsevier. https://link.springer.com/article/10.1007/BF00993309

[4] Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). Cart. Classification and regression trees.

[5] Breiman, L. (1996). Bagging predictors. Machine learning, 24, 123-140. https://link.springer.com/article/10.1023/A:1018054314350

[6] sklearn.ensemble.RandomForestClassifier https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7] Developing scikit-learn estimators https://scikit-learn.org/stable/developers/develop.html