Práctica II: Árboles y Ensembles

Aprendizaje Automático II 2025-2026

Como entrega de esta práctica deberá subirse a la plataforma Moodle un archivo comprimido (zip) que contenga los archivos *.py* con las implementaciones solicitadas, así como un *jupyter notebook* (.ipynb) con la solución de los ejercicios. En cada uno de estos ficheros debe incluirse un comentario con el nombre y apellidos de los integrantes del grupo.

La entrega será **antes de las 10:00** del día **6 de noviembre para el grupo de los jueves**, y del día **7 de noviembre para el grupo de los viernes**. Ese mismo día, durante la clase de prácticas, se realizará una prueba práctica de evaluación relacionada con la entrega.

Objetivos

Los objetivos de esta práctica son los siguientes:

- Utilización de los árboles de clasificación de la librería *Sklearn*.
- Implementación propia de un árbol de regresión.
- Uso de esta implementación para resolver un problema de regresión.
- Análisis de la evolución del accuracy (y overfitting) conforme se aumenta la profundidad.
- Interpretación del resultado.
- Familiarización con los métodos de ensemble, bagging y boosting, más utilizados de la biblioteca *Sklearn*.

1. Clasificación mediante árboles de clasificación

El objetivo de esta sección es familiarizarse con los árboles de clasificación y su uso práctico con *Sklearn*. Para ello deberá resolver el problema de clasifición

multiclase covtype de *Sklearn*. Como guía siga los siguientes pasos y responda razonadamente a las cuestiones que se le planteen:

1. Cargue los datos del problema:

```
from sklearn.datasets import fetch_covtype
# Load the Covertype dataset
cov_type = fetch_covtype()
# Separate the features and the target variable
X = cov_type.data
y = cov_type.target
```

2. Separe los datos en training (50 %) y test (50 %).

3. Utilice la clase *DecisionTreeClassifier* de *Sklearn* para resolver el problema y dé un resultado de test usando la métrica que considere oportuna.

4. ¿Había *missing values* en nuestro problema? ¿Qué efecto tendría en los árboles no completar los *missing values*?

5. ¿Con qué parámetro podría controlarse la profundidad del árbol? ¿Qué ocurre si un árbol se inicializa sin dar un valor concreto a ese parámetro? ¿Crees que sería una buena práctica no darle ningún valor?

6. ¿Son sensibles los árboles al desequilibrio entre clases?

7. ¿Existe desequilibrio entre las clases de nuestro problema? ¿Con qué parámetro de la clase *DecisionTreeClassifier* podrías *corregirlo*?

8. ¿Cuál es la diferencia entre los parámetros *min_samples_split* y *min_samples_leaf*?

9. ¿Para qué sirve el parámetro *criterion*, qué valores puede tomar y qué significa o representa cada uno de esos valores?

10. El árbol entrenado, ¿realiza *splits* utilizando todas las variables disponibles?

11. ¿Se le ocurre alguna forma de crear un selector de atributos a partir de un árbol de clasificación?

2. Regresión mediante árboles de regresión

El objetivo en esta sección de la práctica es resolver un problema de regresión que consiste en predecir del precio de una casa en California y justificar los motivos de tal predicción. Para ello deberá implementar por sí mismo un árbol de regresión. A continuación le detallamos los pasos a seguir:

2.1. Obtención de los datos

Descargue los datos de *Sklearn*

```
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing()
X, y = data.data, data.target
```

Separe los datos en training (50 %) test (50 %) y realice el preprocesado de los datos que estime conveniente teniendo en cuenta que el modelo de aprendizaje con el que va a tratar de resolver el problema es un árbol de regresión. Finalmente responda a las siguientes preguntas:

1. ¿Podría trabajar un árbol de regresión con *missing values*?
2. ¿Es sensible un árbol de regresión a las magnitudes de los atributos?
3. ¿Es sensible un árbol de regresión a la distribución de las etiquetas?

2.2. Implementación de un árbol de regresión

El objetivo de esta sección es guiar la implementación de un modelo de regresión basado en árboles y que satisfaga todos los requisitos expuestos en el enunciado.

Basaremos el modelo en un árbol binario y desarrollaremos varios métodos con los que inicializar, construir y recorrer el árbol. Para ello le será de utilidad seguir los siguientes pasos:

1. Construya una clase *Nodo*, que deberá tener los siguientes atributos:
   - Hijo izquierdo, que será otro Nodo o None.
   - Hijo derecho, que será otro Nodo o None.
   - Índice del atributo a valorar, que será un entero positivo (o cero) menor que la dimensión del *dataset*.
   - Valor umbral, que es la frontera entre el hijo izquierdo y el derecho. Si i es el índice del atributo y θ es el umbral, entonces el hijo izquierdo será seleccionado si $x_i \leq \theta$; de lo contrario se seleccionará el derecho. Si es *None* estamos ante un nodo hoja.
   - Valor del nodo, que puede ser None o un Float. Solo tendrá valor de tratarse de una hoja.
2. Construya la clase *RegressionTree*, que debe contar con los siguientes atributos y métodos:

- Atributos:
  - Nodo raíz, que contendrá una instancia de la clase Nodo.
  - Profundidad máxima.
  - Mínimo de muestras para particionar un nodo.
- Métodos:
  - Método para el **cálculo del error**, `_region_error(y)`, que devuelve la varianza de y o cero en caso de que y sea un array vacío.
  - Método para **calcular el valor del mejor atributo f y el valor θ** con el que realizar la partición. Estos serán los argumentos que minimicen

$$f^*, \theta^* = \arg\min_{f,\theta} \left\{ \frac{|y_{izq}|}{n} regionError(y_{izq}) + \frac{|y_{der}|}{n} regionError(y_{der}) \right\}, \tag{1}$$

    donde $|.|$ representa el operador de cardinalidad de un conjunto y los conjuntos y_{izq} e y_{der} se calculan de la siguiente manera:
    a) Calculamos el conjunto de índices $I_{f\theta} = \{i : x_i \in X \wedge x_{if} \leq \theta\}$ donde x_{if} representa evaluar el atributo f-ésimo de la muestra i-ésima.
    b) $y_{izq} = \{y_i : i \in I\}$.
    c) $y_{der} = \{y_i : i \notin I\}$.
  - Función recursiva, `_construir_nivel(X,y,profundidad)`, que construya el árbol. Funciona de la siguiente manera:
    - Si la profundidad es cero devuelve un Nodo instanciado cuyo valor del nodo sea la media de las etiquetas.
    - Si la longitud de y es uno o menor que el *mínimo de muestras para particionar un nodo*, devuelve un Nodo instanciado cuyo valor sea la media de las etiquetas.
    - Si no, devolverá un Nodo tras el siguiente proceso:
      a) Se calcularán f^*, v^* óptimos.
      b) El hijo izquierdo de este nodo será

$$\_construir\_nivel(X_{izq}, y_{izq}, profundidad - 1).$$

      c) El hijo derecho de este nodo será

$$\_construir\_nivel(X_{der}, y_{der}, profundidad - 1).$$

  - Método `fit(X,y)` que llama al método de construcción de niveles para una matriz y sus etiquetas.

- Método `predict(X)` que predice las etiquetas de una matriz de muestras recorriendo el árbol pertinentemente.
- Método `decision_path(x)` que dado un vector, devuelve un string con las decisiones que le llevan a su predicción. La salida deberá ser un string que indicará el descendiente del que proviene, por ejemplo:

```
"""
1. Atributo HouseAge menor que 10
2. Atributo AveRooms menor que 3.1
"""
```

2.3. Nos enfrentamos al problema de California Housing

Gracias a la clase que ha implementado con anterioridad, devuelva el mejor error en test que le sea posible y responda razonadamente a las siguientes cuestiones:

1. ¿Cómo evoluciona el error de training y test conforme se aumenta la profundidad del árbol? Base su razonamiento en dos gráficas obtenidas con *matplotlib*.

2. ¿Sobre qué parámetros tendría sentido realizar *cross validation* en nuestro modelo?

3. Tome un par de muestras y utilizando el método `decision_path(x)` justifique la predicción dada. Como habrá comprobado, el método `decision_path(x)` devuelve una lista ordenada de argumentos, suponga que esa lista se desordena: ¿seguiría siendo válida la interpretación de la predicción?

4. Para un muestra idéntica a la del apartado anterior, si se entrenara un nuevo árbol, ¿la justificación de su predicción sería la misma?

5. Suponga que divide el conjunto de train en dos mitades, entrena dos árboles distintos, uno con cada mitad, y toma una muestra de test. ¿Sería la predicción de la muestra igual en cada árbol? ¿Y la interpretación obtenida con `decision_path(x)`? ¿Qué repercusión en la interpretación tendría esa situación?

6. Utilice la clase *DecisionTreeRegressor* de *Sklearn* para resolver de nuevo el problema. Compare el *score* de ambos modelos y, para alguna muestra, el resultado de `decision_path(x)` de ambos árboles.

7. En base a su implementación ¿cuál es el coste computacional teórico de entrenar un árbol? ¿Y cuál sería el coste en predicción? ¿Y el coste en memoria? Si tuviera que resolver un problema con un conjunto de datos de gran volumen, ¿qué preferiría usar, un árbol o un modelo de regresión lineal?

3. Ensembles

3.1. Ensembles basados en árboles

En esta sección tratará de resolver el problema de la sección segunda utilizando los siguientes métodos de ensemble que encontrará en la biblioteca de *Sklearn*:

- Bagging.
- Random forest.
- Gradient boosting.

Deberá resolver el problema con el máximo de accuracy en test, siguiendo prácticas adecuadas de aprendizaje automático. Tras ello, responda razonadamente a las siguientes preguntas:

1. ¿Cuáles son los hiperparámetros más relevantes de los métodos anteriores?
2. ¿Es cierto que el uso de ensembles mejora el accuracy de un estimador aislado?
3. ¿Cuantos más estimadores siempre es mejor?
4. ¿Se mantiene la interpretabilidad de los modelos anteriores?
5. ¿Son los modelos anteriores deterministas? Es decir, ¿cada train dará siempre las mismas predicciones?
6. Realice un experimento que consista en medir 10 veces el accuracy obtenido para cada uno de los tres modelos, muestre la evolución de la media y de la varianza frente al número de estimadores en dos gráficas distintas, analice y justifique los resultados.
7. ¿Qué métodos de los anteriores se podrían aplicar si en vez de árboles cada modelo fuera una regresión lineal?

3.2. Bagging para mejorar otros modelos

Los ensembles no solo son válidos para árboles sino que se pueden utilizar para otros modelos, en la sección actual se le propone que utilice un modelo de bagging para resolver el siguiente problema de clasificación de círculos concéntricos:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

# Paso 1: Generar datos de círculos concéntricos
X, y = make_circles(n_samples=1000, noise=0.1, factor=0.3, random_state=42)

# Visualizar los datos
plt.figure(figsize=(8, 6))
plt.scatter(X[y == 0][:, 0], X[y == 0][:, 1],
    color='blue', label='Clase 0', alpha=0.6)
plt.scatter(X[y == 1][:, 0], X[y == 1][:, 1],
    color='red', label='Clase 1', alpha=0.6)
plt.title('Datos de Círculos Concéntricos')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.legend()
plt.grid()
plt.show()


# Paso 2: Dividir el dataset en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.5, random_state=42)
```

Para resolver el problema siga la siguiente guía:

1. Resuelva primero el problema usando un solo estimador del clasificador LogisticRegression, que puede encontrar en *Sklearn*. Tras ello dé una métrica de error.

2. Tras ello realice un ensemble de bagging con boostraping y dé una métrica de error. Para resolver este apartado puede utilizar la clase BaggingClassifier de *Sklearn*.

3. Compare los errores del modelo de un estimador frente al ensemble y justifique qué modelo es mejor y qué es lo que está ocurriendo.