Trabajo Práctico 2: Árboles de Decisión, Random Forests y k-NN

Asignatura: Aprendizaje Automático, 2025, Q2

Fecha de presentación: clase del 07/10/25*

***mandar la presentación y el código 24 horas antes de la clase en la que se presentará el TP**

El objetivo de este TP es entrenar y analizar modelos supervisados como Árboles de Decisión, Random Forests y k-NN desde cero. Con ello se pretende comprender su funcionamiento interno, así como entender como evaluar su rendimiento, capacidad de generalización, y poder explorar técnicas para optimización, eficiencia e interpretabilidad.

Para este trabajo se utilizará el dataset Breast Cancer Wisconsin (Diagnostic), que contiene medidas numéricas extraídas de imágenes médicas para clasificar tumores como benignos o malignos. Es un caso realista y clínicamente relevante que permite trabajar con variables continuas, clasificación binaria y evaluación práctica de modelos supervisados. Puedes encontrar los datos con una explicación un poco más detallada aquí:

https://archive.ics.uci.edu/dataset/17/breast+cancer+wisconsin+diagnostic

1. Introducción teórica (sólo para grado)

Esta información deberá estar contenida en la presentación (brevemente):

1.1 ¿Qué es un árbol de decisión?

1.2 Construcción de árboles: algoritmo CART y Gini

1.3 Sobreajuste, generalización y técnicas de pre-poda

1.4 Random Forests, bootstraping

1.5 KNN que son y cómo funcionan

1.6 Cuáles son los hiperparámetros y modificaciones relevantes en un KNN

2. Implementación desde cero

Separa el aleatoriamente el 80% de valores para tu conjunto de entrenamiento y reserva el 20% restante (test) para el apartado 6 (¡¡No lo uses hasta entonces!!)

2.1 Implementación básica del árbol CART con criterio de Gini: Programar desde cero un árbol de decisión binario usando Gini como criterio de impureza. Implementar la lógica de splits, nodos y predicción final.

2.2 Validación cruzada (por ejemplo, 5-fold): Implementar validación cruzada para estimar el rendimiento del modelo en los datos de entrenamiento*. Usar 5-fold como configuración estándar.

Solamente en los datos de entrenamiento, si no has separado todavía entrenamiento y test, hazlo ahora (80-20).* ***Todo lo que ocurra desde aquí hasta el apartado 6 será solo en los datos de entrenamiento!*

2.3 Evaluación: accuracy en entrenamiento y validación, generalization gap: Utiliza el 5-fold cross-validation implementado en el apartado 2.2 para evaluar el modelo. Calcular el accuracy (% de valores bien clasificados) en train y validation* para cada uno de los folds. ¿Qué información puedes sacar de esos valores? ¿Hay generalization gap? ¿Podría estar produciéndose sobreajuste? ¿y subajuste?

**De ahora en adelante, cuando reportes validation (y train) accuracy, hazlo como la media ± desviación estandard de los 5 folds*

2.4 Pre-poda: max_depth, min_samples_split, min_samples_leaf: Probar distintos valores de estos parámetros para observar cómo afectan la complejidad y el rendimiento del modelo (por ejemplo un árbol con max_depth = 5, otro con min_samples_split = 10, y otro con min_samples_ leaf = 10). Comparar los resultados de éstos arboles con el del apartado 2.3 (que no tiene pre-poda).

3. Optimización del modelo

3.1 Curva de validación para un hiperparámetro: Construir una curva de validación* que muestre cómo varía el accuracy con respecto a un hiperparámetro (por ejemplo max_depth). ¿Viendo esa curva, que valor de max_depth usarías?

**Estamos calculando esta curva de validación dentro del k-fold cross-validation, eso quiere decir que calcularemos una curva para cada uno de los conjuntos de validación (5 en nuestro caso) y promediaremos esos 5 para obtener nuestra curva de validación. Esto será mucho más robusto que hacerlo solo en 1 de los folds.*

3.2 Búsqueda en rejilla para dos hiperparámetros con validación cruzada: Realizar una grid search sobre una malla de combinaciones de dos hiperparámetros (por ejemplo, max_depth y min_samples_leaf). Usar validación cruzada para evaluar cada combinación posible de valores dentro del rango que definas para cada hiperparámetro. Elegir la mejor combinación y compara el valor de accuracy en train y validation con el de los arboles entrenados en el apartado 2. ¿ha cambiado algo?

4. Extensión a Random Forests

4.1 Implementación con bootstraping: Implementar el procedimiento de bagging para construir múltiples árboles entrenados con diferentes subconjuntos del dataset obtenidos por muestreo con reemplazo (*bootstrap*).

4.2 Incorporación de selección aleatoria de features por nodo (feature bagging): Agregar selección aleatoria de variables en cada nodo para aumentar la diversidad de los árboles y reducir la correlación entre ellos.

4.3 Comparación con árbol individual: rendimiento, estabilidad y generalización: Comparar el rendimiento de Random Forest contra un árbol individual. Evaluar accuracy (media y d.e) entre folds.

5. k-NN (K-Nearest Neighbors)

5.1 Implementación de k-NN desde cero: Desarrollar un clasificador k-NN que calcule la distancia (euclidiana) a los vecinos más cercanos y decida la clase por mayoría. Probar varios valores de k.

5.2 Versión con pesos según la distancia: Extender el k-NN para que cada vecino vote con un peso inversamente proporcional a su distancia, dándole más importancia a los vecinos más cercanos.

5.3 Optimización de hiperparámetros de k-NN: Evaluar diferentes valores de k (por ejemplo, de 1 a 30) usando validación cruzada. Construir una curva de validación para observar cómo varía el rendimiento y elegir el valor óptimo.

6. Comparación de resultados

6.1 Comparación de todos los modelos: Árbol de decisión, Random Forest y k-NN (normal y weighted): En base a todas las pruebas hechas hasta ahora, selecciona el mejor candidato de cada uno de los "tipos" de clasificador utilizados: Un árbol de decisión, un Random Forest, un k-NN y un weighted k-NN (cada uno con los hiperparametros que consideres mejores para este problema).

Para cada uno de ellos, entrena en TODO el conjunto de entrenamiento y clasifica el conjunto de TEST*.

Crea un gráfico en el que, para cada uno de los cuatro, plotees accuracy de validación (la has calculado en los apartados anteriores) y accuracy de test.

**Si todo ha ido bien hasta aquí, debería ser la primera vez que lo usas. Si lo habías usado antes, intenta borrar de tu mente todo lo que has aprendido de tus hiperparámetros y vuelve al apartado 2.2*

6.2 Extra: Si durante este TP se te ha ocurrido alguna mejora/aspecto que probar en los datos, features, hiperparámetros o combinaciones de ellos, hazlo y añádelo a la comparación del 6.1.

7. Conclusiones finales

En base al grafico del apartado 6.1 (**¡o 6.2 si has decidido probar algo adicional!**), ¿que clasificador elegirías para implementarlo en una aplicación real? ¿Por qué? ¿Qué precisión esperarías que vaya a tener ese clasificador en esa aplicación?