Práctica 5
Árboles de decisión

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

Jesus Abraham Haros Madrid
A01252642@itesm.mx
Tecnológico de Monterrey
Ingeniería en Tecnologías Computacionales
Monterrey, N.L., México

ABSTRACT

En la actualidad, uno de los problemas más importantes para los científicos es la clasificación: el etiquetado de elementos, basándose en las características de estos, seleccionando de un conjunto de clases, una que mejor lo represente. Los árboles de decisión, el objeto de estudio en esta práctica, son clasificadores que predicen las clases de los elementos usando algoritmos simples, lo que facilita su uso e implementación.

1 INTRODUCCIÓN

Un árbol de decisión clasifica instancias de datos planteando una serie de preguntas sobre las características de estos elementos. Cada pregunta se representa con un nodo, y cada nodo apunta a un nodo hijo, que puede ser un nodo terminal (que presenta el resultado del árbol: una clase o etiqueta), u otro nodo de decisión. Las preguntas forman así una jerarquía de decisiones capturada en una estructura de árbol.

Para clasificar un elemento se sigue el camino desde el nodo superior o raíz, hasta un nodo terminal, dependiendo las características del nodo y las preguntas que cada hoja del camino presenten.

Una ventaja de los árboles de decisión es que muchas veces son más interpretables que otros clasificadores, como las redes neuronales y las máquinas de vectores de soporte[1], porque combinan preguntas sencillas sobre los datos de forma comprensible. Por desgracia, pequeños cambios en los datos de entrada pueden provocar a veces grandes cambios en el árbol construido. Los árboles de decisión son lo suficientemente flexibles como para manejar elementos con una mezcla de características de valor real y categóricas, así como elementos con algunas características ausentes.

2 CONCEPTOS PREVIOS

- Programación básica en Python
- Conocimiento de las librerías *scikit-learn*, *matplotlib* y *numpy*
- Conocimientos básicos de estadística

3 METODOLOGÍA

Esta fue la primera práctica donde la metodología para llevarla a cabo fue muy clara en la descripción de la actividad, por lo que se siguieron los pasos descritos en el documento de la práctica 5.

Se comenzó por instalar *graphviz* en las computadoras, seguido de corroborar que se tenía instalado *numpy*, *matplotlib* y *scikit-learn*. Una vez hecho esto se creó el archivo llamado `practice5.py` y se configuró para que al entrenar el modelo las visualizaciones fueran estéticamente vistosas, así como también establecer el directorio donde se guardarían.

Se comenzó por entrenar un árbol de decisión para el dataset Iris incluido en la librería de *scikit-learn*, una vez entrenado se dibujó utilizando *graphviz*. Después, se probó el árbol de decisión utilizando los conjuntos de entrenamiento y prueba, y se calculó la precisión del modelo.

Para poder hacer el script genérico se optó por utilizar el módulo *sys* y leer como argumento el dataset con el que se quiere trabajar.

3.1 Dataset Iris

Se utilizó el dataset Iris incluido en la librería de *scikit-learn*, para esta práctica se tomaron solo los dos últimos features (petal length y petal width) de los datos ya que el código proporcionado para graficar los resultados se establecían sólo esas dos columnas. Una vez seleccionadas las columnas se entrena el modelo y se grafica el árbol de decisión correspondiente. Por último se calcula su precisión y se grafica la predicción del modelo para poder visualizar las particiones generadas por cada split.

3.2 Dataset Wine

Muy similar a lo que se hizo con el dataset Iris, se importó este dataset de la librería *scikit-learn* pero en este caso tomamos todos los features. Se entrenó el modelo y se graficó el árbol de decisión para este dataset. Por último se calculó su precisión.

3.3 Dataset Breast Cancer

Al igual que lo que se hizo con el dataset Iris, se importó este dataset de la librería *scikit-learn* y se utilizaron todos los features. Se entrenó el modelo y se graficó el árbol de decisión para este dataset. Por último se calculó su precisión.

3.4 Modelo de árbol de decisión

Para la generación del modelo del árbol de decisión se utilizó la implementación de la librería *sklearn*, específicamente la clase *sklearn.tree.DecisionTreeClassifier*[2], que, aunque permite al usuario elegir qué algoritmo utilizar para medir la calidad de las preguntas dentro del árbol, utiliza por defecto el algoritmo de impureza de Gini. También decidimos limitar el número de hojas en los árboles a 6 para evitar caer en *overfitting* sobre los datos de entrenamiento.

El código que ejecutamos para realizar el análisis del dataset se encuentra en el apéndice A.

4 RESULTADOS

Los resultados obtenidos en está práctica fueron muy similares para los tres datasets, y gracias a las librerías mencionadas anteriormente se pudo generar una imagen con el árbol de decisión

entrenado a partir de los conjuntos de entrenamiento y prueba de cada dataset. A continuación se presentan los resultados para cada uno:

4.1 Dataset Iris

Se aprecia como el dato con mayor ganancia es ”petal width” y en base a eso se empieza a determinar la clasificación.

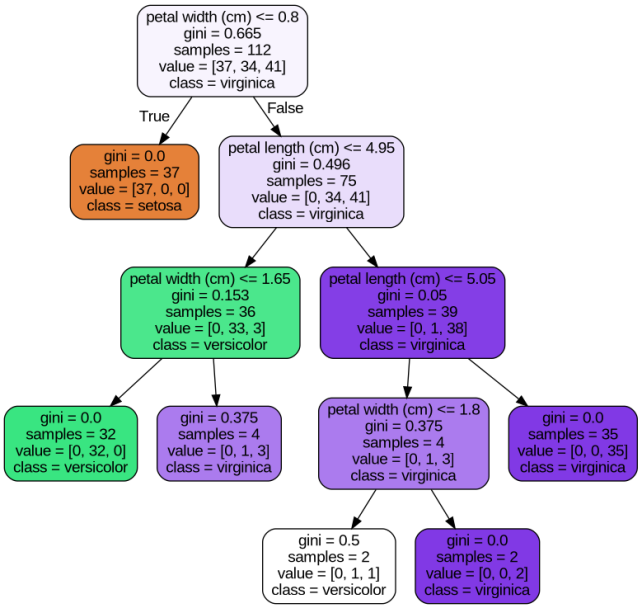

Figure 1: Modelo de árbol de decisión generado para el dataset Iris

Al graficar los resultados del modelo y ver donde se clasifican las instancias se puede ver que es muy preciso y que muy pocas instancias quedan fuera de la clase a la que pertenecen.

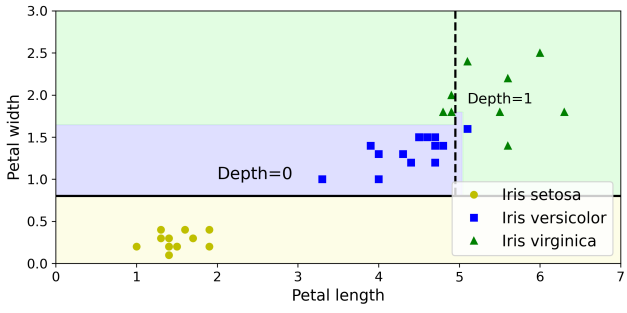

Figure 2: Resultados del modelo del árbol de decisión generado para el dataset Iris

Este árbol de decisión tuvo una precisión de 0.974 al ser comparado con el conjunto de datos de prueba.

4.2 Dataset Wine

Como en este dataset no se quitaron features, el árbol de decisión generado se nota más detallado, siendo color_intensity el atributo con mayor ganancia.

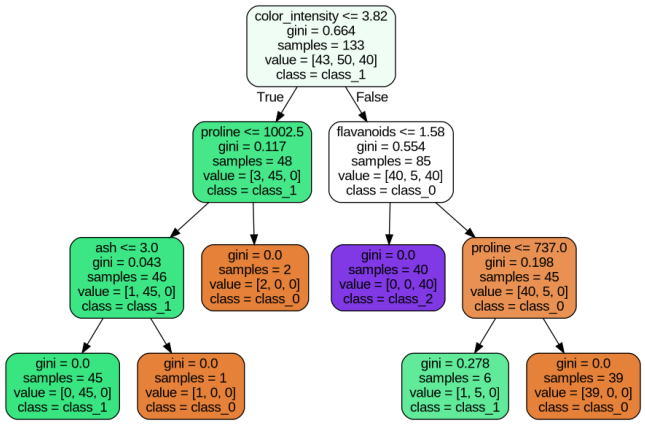

Figure 3: Modelo de árbol de decisión generado para el dataset wine

Este árbol de decisión tuvo una precisión de 0.933 al ser comparado con el conjunto de datos de prueba.

4.3 Dataset Breast Cancer

Este fue el dataset con mayor número de atributos, inicialmente el resultado fue un árbol muy grande, pero después de recortarlo al máximo de 6 nodos hoja obtuvimos el siguiente árbol, siendo ”mean concave points” el atributo con mayor ganancia.

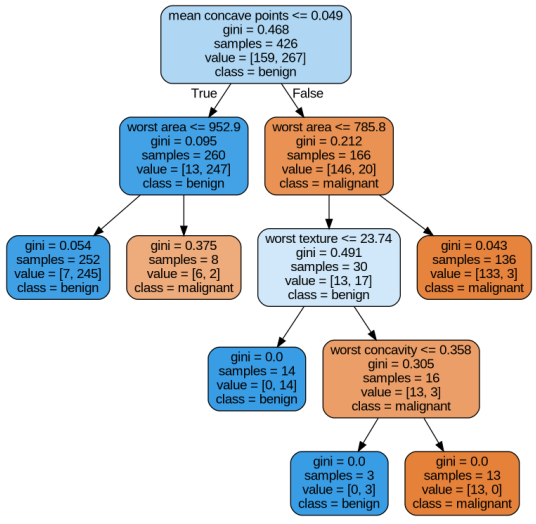

Figure 4: Modelo de árbol de decisión generado para el dataset breast_cancer

Este árbol de decisión tuvo una precisión de 0.958 al ser comparado con el conjunto de datos de prueba.

5 CONCLUSIONES Y REFLEXIONES

Los árboles de decisión son herramientas muy útiles para poder clasificar instancias y a la vez poder visualizar qué está pasando dentro del modelo. Claro que entre más features se tengan más difícil será visualizar el árbol y más extenso el trabajo de graficar resultados de prueba. La serie de preguntas de un árbol de decisión está ordenada con los atributos que tienen mayor ganancia donde cada nodo apunta a un nodo hijo con una nueva pregunta o un nodo terminal. Al utilizar el dataset Iris fue muy interesante ver cómo el modelo entrenado sí clasificaba correctamente la mayoría de los datos y poder observar el proceso que sigue para clasificarlos.

5.1 Refrexión de Abraham

Esta práctica fue sencilla de elaborar y refuerza lo que ya se había trabajado a mano dentro del examen parcial, me gustó poder visualizar el árbol de decisión del modelo entrenado y también poder comprobar si se clasificaban correctamente las instancias utilizando la segunda función para graficar. Me parece una técnica muy útil cuando se trata de clasificar datos ya que se puede mostrar el proceso a los clientes que lo necesiten. Por último, me hubiera gustado hacer las gráficas de comprobación para cada dataset pero se ve largo y hardcodeado de elaborar por lo que investigaré si existe alguna librería para hacerlo.

5.2 Reflexión de Mario

Por mi parte, considero que esta práctica me ayudó a comprender por qué se utilizan los árboles de decisión a pesar de que toman decisiones "codiciosas" y con son muy inestables cuando se seleccionan los datos utilizados para entrenar el árbol. Las características que yo pienso hacen del árbol de decisión un clasificador fácil de aprender y enseñar son que se puede representar gráficamente como lo hicimos en la sección resultados, además de que el proceso de clasificación de una instancia o elemento es una serie de preguntas simples.

REFERENCES

[1] Carl Kingsford and Steven L Salzberg. 2008. What are decision trees? *Nature biotechnology* 26, 9 (2008), 1011–1013.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

A CÓDIGO PARA LA GENERACIÓN DEL ÁRBOL DE DECISIÓN

```
import os
import sys
from graphviz import Source
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.datasets import load_iris, load_wine, load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_graphviz


def plot_decision_boundary(clf, X, y, axes=[0, 7, 0, 3]):
    x1s = np.linspace(axes[0], axes[1], 200)
    x2s = np.linspace(axes[2], axes[3], 200)
    x1, x2 = np.meshgrid(x1s, x2s)
    X_new = np.c_[x1.ravel(), x2.ravel()]
    y_pred = clf.predict(X_new).reshape(x1.shape)
    custom_cmap = ListedColormap(['#fafab0', '#9898ff', '#a0faa0'])
    plt.contourf(x1, x2, y_pred, alpha=0.3, cmap=custom_cmap)
    plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "yo", label="Iris setosa")
    plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], "bs", label="Iris versicolor")
    plt.plot(X[:, 0][y == 2], X[:, 1][y == 2], "g^", label="Iris virginica")
    plt.axis(axes)
    plt.xlabel("Petal length", fontsize=14)
    plt.ylabel("Petal width", fontsize=14)
    plt.legend(loc="lower right", fontsize=14)


def save_fig(fig_id, tight_layout=True, fig_extension="png", resolution=300):
    path = os.path.join(IMAGES_PATH, fig_id + "." + fig_extension)
    print("Saving figure", fig_id)
    if tight_layout:
        plt.tight_layout()
    plt.savefig(path, format=fig_extension, dpi=resolution)


mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# Setup folders
PROJECT_ROOT_DIR = "."
CHAPTER_ID = "decision_trees"
IMAGES_PATH = os.path.join(PROJECT_ROOT_DIR, "images", CHAPTER_ID)
os.makedirs(IMAGES_PATH, exist_ok=True)
```

```
# Dataset selection
if len(sys.argv) < 2:
    print("Include an argument from the list [iris, wine, breast_cancer]")
    exit(1)

if sys.argv[1] == 'iris':
    iris = load_iris()
    feauture_names = iris.feature_names[-2:]
    target_names = iris.target_names
    treeFilename = 'iris_tree'
    X = iris.data[:, -2:]
    y = iris.target
elif sys.argv[1] == 'wine':
    wine = load_wine()
    feauture_names = wine.feature_names
    target_names = wine.target_names
    treeFilename = 'wine_tree'
    X = wine.data
    y = wine.target
elif sys.argv[1] == 'breast_cancer':
    breast_cancer = load_breast_cancer()
    feauture_names = breast_cancer.feature_names
    target_names = breast_cancer.target_names
    treeFilename = 'breast_cancer_tree'
    X = breast_cancer.data
    y = breast_cancer.target

xTrain, xTest, yTrain, yTest = train_test_split(X, y, random_state=0)

# Create and train the decision tree model
tree_clf = DecisionTreeClassifier(random_state=0, max_leaf_nodes=6)
tree_clf.fit(xTrain, yTrain)

# Export an image of the tree
dot_src = export_graphviz(tree_clf, feature_names=feauture_names,
                          class_names=target_names, rounded=True,
                          filled=True)
image_filename = os.path.join(IMAGES_PATH, treeFilename)
Source(dot_src).render(image_filename, format="png", cleanup=True)

# Test the decision tree
accuracy = tree_clf.score(xTest, yTest)
print("Accuracy:", accuracy)


if sys.argv[1] == 'iris':
    plt.figure(figsize=(8, 4))
    plot_decision_boundary(tree_clf, xTest, yTest)
    plt.plot([0, 7], [0.8, 0.8], "k-", linewidth=2)
    plt.text(2.0, 1.0, "Depth=0", fontsize=15)
    plt.plot([4.95, 4.95], [0.8, 3], "k--", linewidth=2)
    plt.text(5.1, 1.9, "Depth=1", fontsize=13)
    save_fig("decision_tree_decision_boundaries_plot")
```