UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
FACULTAD DE INGENIERÍA ELÉCTRICA, ELECTRÓNICA, INFORMÁTICA Y MECÁNICA
ESCUELA PROFESIONAL DE INGENIERÍA INFORMÁTICA Y DE SISTEMAS

Árbol de Decisión en PySpark

Asignatura: Minería de datos

Docente: Carlos Fernando Montoya Cubas

Integrantes:

- Juan Carlos Condori Lopez 170429
- Melissa Briggitte Espejo Franco 171258
- Alexander Junior Monzon Montalvo 171914
- Edgar Daniel Ramos Alvarez 171570
- Claudia Luz Rojas Soto 171805
- Jorge Andre Salcedo Hurtado 171572

Cusco - Perú
2021

ÍNDICE

Introducción

Muchas áreas como la ingeniería, medicina, biología, etc. tienen problemas de clasificación o regresión que necesitan ser resueltos, es así que la aplicación de técnicas de aprendizaje automático supervisado se han convertido popularmente usadas para resolverlos. Muchas de estas técnicas utilizan distintos métodos y algunas son difíciles de interpretar.

Es así, que la elección de los árboles de decisión para el presente trabajo es debido a su efectividad e interpretación en diversas situaciones. De manera más precisa, un árbol de decisión es un modelo predictivo que divide recursivamente el espacio de la covariable en subespacios, de modo que cada subespacio constituye una base para una función de predicción diferente. Los árboles de decisión se pueden utilizar para diversas tareas de aprendizaje, incluida la clasificación, la regresión y el análisis de supervivencia. Debido a sus beneficios únicos, los árboles de decisión se han convertido en uno de los enfoques más poderosos y populares en la ciencia de datos.

Una de sus variaciones más conocidas es el algoritmo de árboles de clasificación y regresión (CART - Classification and Regression Tree). El algoritmo CART es un tipo de algoritmo de clasificación que se requiere para construir un árbol de decisión sobre la base del índice de impureza de Gini. Es un algoritmo básico de aprendizaje automático y proporciona una amplia variedad de casos de uso.

En el presente trabajo, presentamos la implementación del algoritmo CART usando pySpark, con el objetivo de proponer alternativa que mejore la eficiencia de este algoritmo para una gran cantidad de datos, tarea que nos ayuda a realizar la librería en mención.

1. Aprendizaje Automático

Es un campo de las ciencias de la computación que, de acuerdo a Arthur Samuel en 1959, le da a las computadoras la habilidad de aprender sin ser explícitamente programadas.

Es la idea de que existen algoritmos que pueden darte hallazgos o conclusiones relevantes obtenidas de un conjunto de datos, sin que el ser humano tenga que escribir instrucciones o códigos para esto.

El propósito del aprendizaje automático es que las personas y las máquinas trabajen de la mano, al éstas ser capaces de aprender como un humano lo haría. Precisamente esto es lo que hacen los algoritmos, permiten que las máquinas ejecuten tareas, tanto generales como específicas.

Si bien al principio sus funciones eran básicas y se limitaban a filtrar emails, hoy en día puede hacer cosas tan complejas como predicciones de tráfico en intersecciones muy transitadas, detectar cáncer, mapear sitios para generar proyectos de construcción en tiempo real, e incluso, definir la compatibilidad entre dos personas.

¿Cómo funciona?

El principal objetivo de todo aprendiz (learner) es desarrollar la capacidad de generalizar y asociar. Cuando traducimos esto a una máquina o computadora, significa que éstas deberían poder desempeñarse con precisión y exactitud, tanto en tareas familiares, como en actividades nuevas o imprevistas.

¿Y cómo es posible esto?

Haciendo que repliquen las facultades cognitivas del ser humano, formando modelos que “generalicen” la información que se les presenta para realizar sus predicciones. Y el ingrediente clave en toda esta cuestión son los datos.

En realidad, el origen y el formato de los datos no es tan relevante, dado que el aprendizaje automático es capaz de asimilar una amplia gama de éstos, lo que se conoce como ***big data***, pero éste no los percibe como datos, sino como una enorme lista de ejemplos prácticos.

Podríamos decir que sus algoritmos se dividen principalmente en tres grandes categorías: supervised learning (aprendizaje supervisado), unsupervised learning (aprendizaje no

supervisado) y reinforcement learning (aprendizaje por refuerzo). A continuación, se detalla las diferencias entre estas.

Tipos de aprendizaje automático

Aprendizaje supervisado

Depende de datos previamente etiquetados, como podría ser el que una computadora logré distinguir imágenes de coches, de las de aviones. Para esto, lo normal es que estas etiquetas o rótulos sean colocadas por seres humanos para asegurar la efectividad y calidad de los datos.

En otras palabras, son problemas que ya hemos resuelto, pero que seguirán surgiendo en un futuro. La idea es que las computadoras aprendan de una multitud de ejemplos, y a partir de ahí puedan hacer el resto de cálculos necesarios para que nosotros no tengamos que volver a ingresar ninguna información.

Ejemplos: reconocimiento de voz, detección de spam, reconocimiento de escritura, entre otros.

Aprendizaje no supervisado

En esta categoría lo que sucede es que al algoritmo se le despoja de cualquier etiqueta, de modo que no cuenta con ninguna indicación previa. En cambio, se le provee de una enorme cantidad de datos con las características propias de un objeto (aspectos o partes que conforman a un avión o a un coche, por ej.), para que pueda determinar qué es, a partir de la información recopilada.

Ejemplos: detectar morfología en oraciones, clasificar información, etc.

Aprendizaje reforzado

En este caso particular, la base del aprendizaje es el refuerzo. La máquina es capaz de aprender con base a pruebas y errores en un número de diversas situaciones.

Aunque conoce los resultados desde el principio, no sabe cuáles son las mejores decisiones para llegar a obtenerlos. Lo que sucede es que el algoritmo progresivamente va asociando los patrones de éxito, para repetirlos una y otra vez hasta perfeccionarlos y volverse infalible.

Ejemplos: navegación de un vehículo en automático, toma de decisiones, etc.

Aplicaciones

Dado que el aprendizaje automático es un sistema basado en el procesamiento y análisis de datos que son traducidos a hallazgos, se puede aplicar a cualquier campo que cuente con bases de datos lo suficientemente grandes. De momento, algunos de sus usos más populares y desarrollados son:

- Clasificación de secuencias de DNA
- Predicciones económicas y fluctuaciones en el mercado bursátil
- Mapeos y modelados 3D
- Detección de fraudes
- Diagnósticos médicos
- Buscadores en Internet
- Sistemas de reconocimiento de voz
- Optimización e implementación de campañas digitales publicitarias

2. Árboles de Decisión

Definición

Los árboles de decisión son una técnica de aprendizaje automático supervisado muy utilizada en muchas áreas. Como su nombre indica, esta técnica de aprendizaje automático toma una serie de decisiones en forma de árbol. Los nodos intermedios (las ramas) representan soluciones. Los nodos finales (las hojas) nos dan la predicción que vamos buscando.

Los algoritmos de aprendizaje basados en árboles se consideran uno de los mejores y más utilizados métodos de aprendizaje supervisado. Los métodos basados en árboles potencian los modelos predictivos con alta precisión, estabilidad y facilidad de interpretación.

A diferencia de los modelos lineales, mapean bastante bien las relaciones no lineales. Son adaptables para resolver cualquier tipo de problema (clasificación o regresión).

Clasificación

a) Árboles de decisión para regresión

Definición

Los árboles de regresión son el subtipo de árboles de predicción que se aplica cuando la variable respuesta es continua. En términos generales, en el entrenamiento de un árbol de regresión, las observaciones se van distribuyendo por bifurcaciones (nodos) generando la estructura del árbol hasta alcanzar un nodo terminal. Cuando se quiere predecir una nueva observación, se recorre el árbol acorde al valor de sus predictores hasta alcanzar uno de los nodos terminales. La predicción del árbol es la media de la variable respuesta de las observaciones de entrenamiento que están en ese mismo nodo terminal.

Entrenamiento del árbol

El proceso de entrenamiento para este tipo de árboles se divide en dos etapas:

- División sucesiva del espacio de los predictores generando regiones no solapantes (nodos terminales). R1, R2, R3, … , Rj. Aunque, desde el punto de vista teórico las regiones

podrían tener cualquier forma, si se limitan a regiones rectangulares (de múltiples dimensiones), se simplifica en gran medida el proceso de construcción y se facilita la interpretación.

- Predicción de la variable respuesta en cada región.

A pesar de la sencillez con la que se puede resumir el proceso de construcción de un árbol, es necesario establecer una metodología que permita crear las regiones R1 , R2 , R3 , ..., Rj , o lo que es equivalente, decidir donde se introducen las divisiones: en que predictores y en qué valores de los mismos. Es en este punto donde se diferencian los algoritmos de árboles de regresión y clasificación.

En los árboles de regresión, el criterio empleado con más frecuencia para identificar las divisiones es el Residual Sum of Squares (RSS). El objetivo es encontrar las J regiones (R_1,...,R_j) que minimizan el Residual Sum of Squares (RSS) total.

b) Árboles de decisión para clasificación

Definición

Los árboles de regresión son el subtipo de árboles de predicción que se aplica cuando la variable respuesta es categórica.

Construcción

Para construir un árbol de clasificación, se emplea el mismo método recursive binary splitting descrito en los árboles de regresión. Sin embargo, como la variable respuesta es cualitativa, no se puede emplear el RSS como criterio de selección de las divisiones óptimas. Existen varias alternativas, todas ellas con el objetivo de encontrar nodos lo más puros/homogéneos posible. Las más empleadas son:

- **Classification Error Rate:** Se define como la proporción de observaciones que no pertenecen a la clase mayoritaria del nodo.

$$E_m = 1 - max_k(\hat{p}_{mk})$$

donde p_{mk} representa la proporción de observaciones del nodo m que pertenecen a la clase k A pesar de la sencillez de esta medida, no es suficientemente sensible para crear buenos árboles, por lo que, en la práctica, no suele emplearse.

- **Gini Index:** cuantifica la varianza total en el conjunto de las K clases del nodo m ,es decir , mide la pureza del nodo.

$$G_m = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Cuando $\widehat{p}_{mk}$ es cercano a 0 o a 1 (el nodo contiene mayoritariamente observaciones de una sola clase), el término $\widehat{p}_{mk}(1 - \widehat{p}_{mk})$ es muy pequeño. Como consecuencia, cuanto mayor sea la pureza del nodo, menor el valor del índice Gini G.

$$D = -\sum_{k=1}^{K} \hat{p}_{mk}\ log(\hat{p}_{mk})$$

- **Information Gain: Cross Entropy:** La entropía es una forma de cuantificar el desorden de un sistema. En el caso de los nodos, el desorden se corresponde con la impureza. Si un nodo es puro, contiene únicamente observaciones de una clase, su entropía es cero. Por el contrario, si la frecuencia de cada clase es la misma, el valor de la entropía alcanza el valor máximo de 1.
- **Chi-Square:** Esta aproximación consiste en identificar si existe una diferencia significativa entre los nodos hijos y el nodo parental, es decir, si hay evidencias de que la división consigue una mejora. Para ello, se aplica un test estadístico chi-square goodness of fit empleando como distribución esperada H_0 la frecuencia de cada clase en el nodo parental. Cuanto mayor el estadístico X^2 mayor la evidencia estadística de que existe una diferencia.

Para el proceso de construcción del árbol, acorde al libro Introduction to Statistical Learning, Gini index y cross-entropy son más adecuados que el ***classification error rate*** debido a su mayor sensibilidad a la homogeneidad de los nodos. Para el proceso de pruning (descrito en la siguiente sección) los tres son adecuados, aunque, si el objetivo es conseguir la máxima precisión en las predicciones, mejor emplear el ***classification error rate.***

Predicción del árbol

Tras la creación de un árbol, las observaciones de entrenamiento quedan agrupadas en los nodos terminales. Para predecir una nueva observación, se recorre el árbol en función del valor de sus predictores hasta llegar a uno de los nodos terminales. En el caso de clasificación, suele emplearse la moda de la variable respuesta como valor de predicción, es decir, la clase más frecuente del nodo. Además, puede acompañarse con el porcentaje de cada clase en el nodo terminal, lo que aporta información sobre la confianza de la predicción.

Algoritmo CART (Classification And Regression Tree)

Para la implementación del árbol de decisión usando pySpark se usó el algoritmo CART, el cual es un tipo de algoritmo de clasificación que se requiere para construir un árbol de decisión sobre la base del índice de impureza de Gini.

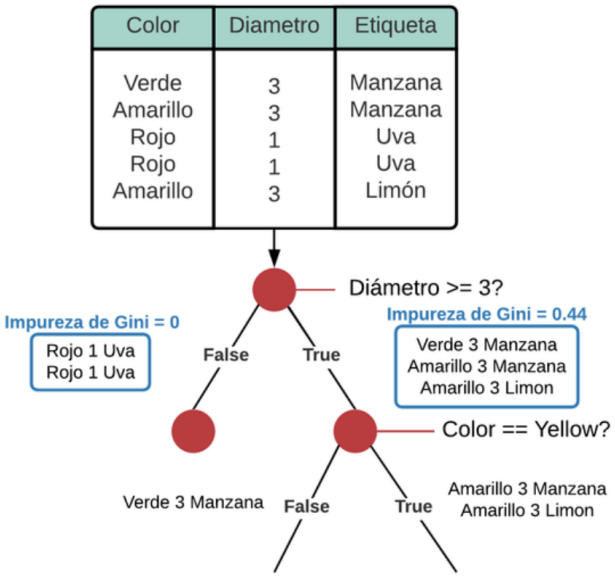
CART es una palabra que se refiere a los siguientes tipos de árboles de decisión (que también se vieron anteriormente):

- **Árboles de clasificación:** Cuando la variable objetivo es continua, el árbol se utiliza para encontrar la "clase" en la que es más probable que caiga la variable objetivo.
- **Árboles de regresión:** Se utilizan para predecir el valor de una variable continua.

Árboles de clasificación y regresión

CART (Classification and Regression Trees) inicializa la raíz del árbol con el dataset de entrenamiento completo como se ve en la fig.. Luego busca dividir los datos del dataset con una pregunta. Los nodos siguientes reciben solamente los datos que corresponden a la respuesta.

Se reitera este proceso hasta desenredar totalmente los datos (cada hoja del árbol debería tener idealmente un solo tipo de etiqueta).

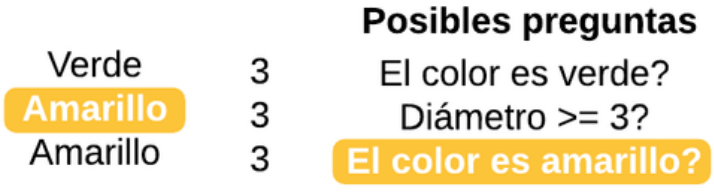


Para construir un árbol eficiente, el punto importante es identificar qué preguntas formular y cuándo. Por lo tanto, necesitamos cuantificar en qué medida una pregunta permite desenredar las etiquetas. Para hacer eso se utiliza 2 métricas:

- El coeficiente de 'Gini impurity': mide que tan desenredados están los labels de un nodo.
- El coeficiente de 'Information Gain': mide cuánto una pregunta permite bajar el 'Gini impurity'.

¿Qué preguntas formular?

Para saber qué preguntas formular, cada nodo itera sobre las características de los datos a su disposición y define una lista de preguntas posibles.

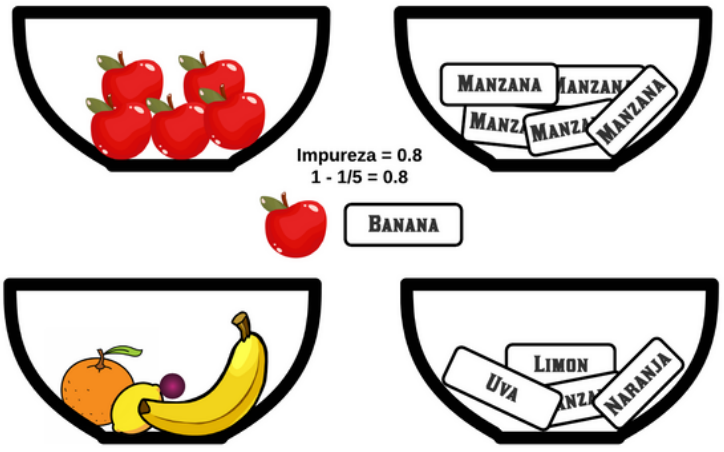


Partición

Una vez una pregunta elegida, se divide los datos en dos según la respuesta a la pregunta.

Coeficiente de Gini

El coeficiente de Gini impurity representa la probabilidad de ser incorrecto si asigna aleatoriamente una etiqueta a un ejemplo del mismo conjunto. Por ejemplo, en los dos ejemplos siguientes: ¿Cuál es la probabilidad de equivocarse si asignamos una etiqueta del recipiente B a un dato del recipiente A?



¿Ganancia de información usando Gini?

Para un nodo S en el conjunto de datos con c clases objetivo, el índice Gini se define como:

$$Gini(S) = 1 - \sum_{i=1}^{c} (p_i)^2$$

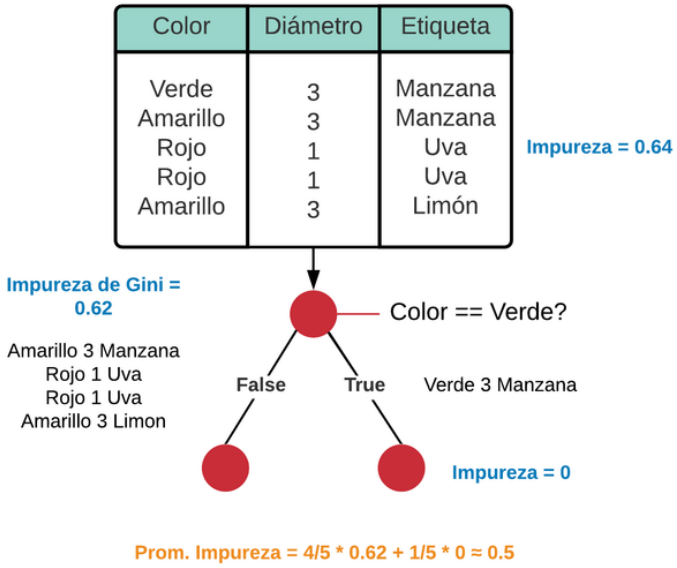
Teniendo a p_i como la proporción de S que pertenece a la clase i. El cálculo del índice de Gini de una división con el atributo A se define como:

$$Gini(A_{split}) = \sum_{i=1}^{c} p_i Gini(F_i)$$

Donde p_i es la proporción de la clase i después de la división. F representa las características. El cálculo de la ganancia de información sobre el atributo A después de la división se define como:

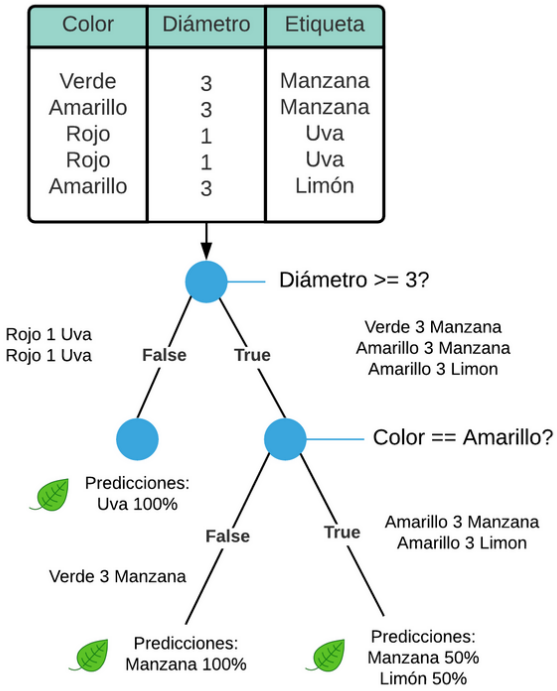
$$Gain\ (S,\ A) = Gini(S) - Gini(A_{split})$$

Fuente: *Mohammad A. (2018) A Comprehensive Study of Decision Trees to Classify DNA Sequences. Universidad de Nuevo México.*



Encontrar mejor división

Se encuentra la mejor pregunta iterando sobre cada función/valor y calculando la ganancia de información, anteriormente explicada.



3. SPARK

Spark es una solución de big data que ha demostrado ser más fácil y rápida que Hadoop MapReduce. Spark es un software de código abierto desarrollado por UC Berkeley RAD lab en 2009.

En la era de los big data, los profesionales necesitan más que nunca herramientas rápidas y confiables para procesar la transmisión de datos. Las herramientas anteriores como MapReduce eran favoritas, pero eran lentas. Para superar este problema, Spark ofrece una solución rápida y de uso general. La principal diferencia entre Spark y MapReduce es que Spark ejecuta cálculos en memoria durante el último en el disco duro. Permite el acceso de alta velocidad y el procesamiento de datos, reduciendo tiempos de horas a minutos.

pySpark

Spark es el nombre del motor para realizar la computación en clúster, mientras que PySpark es la biblioteca de Python para usar Spark.

4. Implementación del Árbol de Decisión en pySpark

- **MÓDULO VALORES ÚNICOS**

```python
def unique_vals(rows, col):
    """Encuentre los valores únicos para una columna en un conjunto de datos.
    Args:
        rows (list of lists): Dataset
        col (int): Indica la columna con la que se está trabajando
    Returns:
        list: Valores únicos la columna evaluada
    """
    # Creamos la rdd
    sparkRow=sc.parallelize(rows,os.cpu_count())
    # Mapeamos el rdd para la columna evaluada
    sparkColumn=sparkRow.map(lambda x:x[col])
    # Reducimos extrayendo los valores únicos de la columna
    unique_vals=sparkColumn.map(lambda x:(x,1)).reduceByKey(lambda x,y:x+y).map(lambda x:x[0])
    # Devolvemos nuestra lista de valores únicos
    return sorted(unique_vals.collect())
```

- **CONTADOR DE CLASES**

```python
import functools
import itertools
def class_counts(rows):
    """Cuenta el número de cada tipo de ejemplo en un conjunto de datos.
    Args:
        Rows(list of list): Dataset
    Returns:
        Dict(devuelve la labels de las clases y la cantidad de repeticiones en el dataset)
    """
    #Mapeamos la columna final que contiene las clases del dataset
    LastColumn=map(lambda x:x[-1],rows)
    #Agregamos un valor de 1 a cada valor para la reduccion
    Clases=map(lambda x:(x,1),LastColumn)
    list_key=[]
    list_values=[]
    #Reducimos nuestros datos clase
    for key, group in itertools.groupby(list(Clases), lambda x: x[0]):
        list_key.append(key)#guardamos nuestra clase
        list_values.append(sum(list(map(lambda x:x[1],list(group)))))#guardamos las repeticiones de las
clases
    dict_from_list = dict(zip(list_key,list_values))#Creamos nuestro diccionario
    return dict_from_list
```

- **VERIFICADOR NUMÉRICO**

```python
def is_numeric(value):
    """Prueba si un valor es numérico.
    Args:
        value (any): Cualquier valor
    Returns:
        boolean: retorna True si es un valor numérico
    """
    return isinstance(value, int) or isinstance(value, float)
```

- **CLASE PREGUNTA**

```python
class Question:
    """
    Atributos:
        column(int): indica el numero de columna
        value(str):indica el valor de la columna
    """
    def __init__(self, column, value):
        self.column = column
        self.value = value

    def match(self, example):
        """Compara el valor de característica en un ejemplo al valor de característica
            almacenado en la pregunta.
            Args:
                example(list):una fila que contiene una muestra del dataset
            Returns:
                boolean(bool):indica si el valor es mayor o igual si es numerico
                              o igual si es categorico
        """
        # Compara el valor de la función en un ejemplo con el
        # valor de característica en esta pregunta.
        val = example[self.column]
        if is_numeric(val):
            return val >= self.value
        else:
            return val == self.value

    def __repr__(self):
        """
        Método auxiliar para imprimir la pregunta en un formato legible.
            Returns(str): ">=" en caso sea numerico
                          "sentencia" en caso sea categorico
        """
        condition = "=="
        if is_numeric(self.value):
            condition = ">="
        return "Es %s %s %s?" % (
            header[self.column], condition, str(self.value))
```

- **PARTICIÓN DE DATOS**

```python
def partition(rows, question):
    """Particiona un conjunto de datos.
    Para cada fila del conjunto de datos, se comprueba si coincide
    con la pregunta. Si entonces, se agrega a 'filas verdaderas',
    de lo contrario, se agrega a 'filas falsas'.

    Args:
        rows (list of lists): Dataset
        question (object): Validador
    Returns:
        list: Valores únicos la columna evaluada
    """
    verdaderas = list(filter(lambda x: question.match(x) == True,rows))
    falsas = list(filter(lambda x: question.match(x) == False,rows))
    return verdaderas, falsas
```

- **GINI**

```python
import numpy as np
def gini(filas):
    """Calcula la impureza de Gini para una lista de filas.
    Args:
        filas (list of lists): Dataset.
    Returns:
        float: Valor de la impureza de Gini obtenido.

    """
    cantidadXClase = class_counts(filas)
    nroFilas = float(len(filas))
    impurezaGini = 1
    gini = map((lambda lbl: np.power(cantidadXClase[lbl]/nroFilas,2)),cantidadXClase)
    res = sum(list(gini))
    impurezaGini-=res
    return impurezaGini
```

- **GANANCIA DE INFORMACIÓN**

```python
def info_gain(left, right, current_uncertainty):
    """Ganancia de información.
     La incertidumbre del nodo inicial, menos la impureza ponderada de
     dos nodos secundarios.
    Args:
        left: .
        right: .
        current_uncertainty: .
    Returns:
        float: Valor de la impureza de Gini obtenido.
    """
    p = float(len(left)) / (len(left) + len(right))
    return current_uncertainty - p * gini(left) - (1 - p) * gini(right)
```

- **CLASE NODO DE DECISIÓN**

```python
class Decision_Node:
    """Un nodo de decisión hace una pregunta.

     Esto contiene una referencia a la pregunta y a los dos nodos secundarios.
    """

    def __init__(self,
                 question,
                 true_branch,
                 false_branch):
        self.question = question
        self.true_branch = true_branch
        self.false_branch = false_branch
```

- **ENCONTRAR LA MEJOR DIVISIÓN**

```
def find_best_split(rows):
    """Encuentra la mejor pregunta para hacer iterando sobre cada función/valor y
calculando
    la ganancia de información.

    Args:
        rows (list): Dataset
    Returns:
        best_gain: Mejor ganancia de información
        best_question: Mejor pregunta con mejor ganancia
    """
    # Recuperamos la incertidumbre dini de los datos
    current_uncertainty = gini(rows)
    # Recuperamos el numero de columnas de la data
    n_features = len(rows[0]) - 1

    result = []
    # Recorremos cada columna de la data
    for col in range(n_features):

        def best_question_and_gain(val, col = col, rows = rows):
            """Encuentra la mejor pregunta calculando la ganancia de información.

            Args:
                val (): Valor a evaluar
                col (int): Valor de la columna
                rows (int): Valor de la fila
            Returns:
                col: Columna en la cual se trabajo
                val: Valor el cual se evaluo
                gain: Ganancia obtenida
            """

            # Creamos un clase Question con los valores de (col) y (val)
            question = Question(col, val)
            # Recuperamos los arreglos que cumplan o no la pregunta
            true_rows, false_rows = partition(rows, question)

            # Inicializamos la variable de la ganancia y la actualizaremos
            gain = 0
            if len(true_rows) != 0 and len(false_rows) != 0:
                gain = info_gain(true_rows, false_rows, current_uncertainty)

            return (col, val, gain)

        # Recorremos los valores de la data
        values = unique_vals(rows,col)
        # Convertimos la data en un RDD
        palavrasRDD = sc.parallelize(values, os.cpu_count())
        # Recorremos nuestro RDD con un map, y hallamos su ganancia y question con la
funcion
        pluralRDD = palavrasRDD.map(best_question_and_gain)
        # Recuperamos el resultado en un array de tuplas
        result += pluralRDD.collect()

    # Hallamos la maxima ganancia del resultado
    max_value = sc.parallelize(result).sortBy(lambda x: x[2]).collect()
    index = len(max_value) - 1
    # Recuperamos la mejor ganancia
    best_gain = max_value[index][2]
    # Recuperamos los valores de la question, y creamos su clase
    best_question = Question(max_value[index][0],max_value[index][1])


    return best_gain, best_question
```

- **CLASE HOJA**

```
class Leaf:
    """Un nodo Hoja clasifica los datos.

     Esto contiene un diccionario de clase (por ejemplo, "Apple") -> número de veces
     aparece en las filas de los datos de entrenamiento que llegan a esta hoja.
    """

    def __init__(self, rows):
        self.predictions = class_counts(rows)
```

- **MÉTODO PARA IMPRESIÓN DE HOJA**

```
def print_leaf(counts):
    """Una forma más agradable de imprimir las predicciones en una hoja."""
    total = sum(counts.values()) * 1.0
    probs = {}
    for lbl in counts.keys():
        probs[lbl] = str(int(counts[lbl] / total * 100)) + "%"
    return probs
```

- **CONSTRUCCIÓN DEL ÁRBOL**

```python
def build_tree(rows):
    """Construye el árbol.

     Reglas de recursividad:
     1) Creer que funciona.
     2) Comience por verificar
     para el caso base (no se obtiene más información).
     3) Prepárate para
     rastros de pila gigante.
    """
    # Intente particionar el conjunto de datos en cada uno de los atributos únicos,
    # calcular la ganancia de información, y devuelve la pregunta que produce la mayor
ganancia.

    gain, question = find_best_split(rows)

    # Caso base: no se obtiene más información
    # Ya que no podemos hacer más preguntas,
    # devolveremos una hoja.
    if gain == 0:
        return Leaf(rows)

    # Si llegamos aquí, hemos encontrado una característica / valor útil
    # para particionar.
    true_rows, false_rows = partition(rows, question)

    # Construye recursivamente la rama verdadera.
    true_branch = build_tree(true_rows)

    # Construye recursivamente la rama falsa.
    false_branch = build_tree(false_rows)

    # Devolver un nodo Pregunta.
    # Esto registra la mejor característica/valor para preguntar en este punto,
    # así como las ramas a seguir
    # dependiendo de la respuesta.
    return Decision_Node(question, true_branch, false_branch)
```

- **IMPRESIÓN DEL ÁRBOL**

```python
def print_tree(node, spacing=""):
    """La función de impresión de árboles más elegante del mundo."""

    # Caso base: hemos llegado a una hoja
    if isinstance(node, Leaf):
        print (spacing + "Prediccion", node.predictions)
        return

    # Imprimir la pregunta en este nodo
    print (spacing + str(node.question))

    # Llame a esta función recursivamente en la rama verdadera
    print (spacing + '--> Verdadero:')
    print_tree(node.true_branch, spacing + "  ")

    # Llame a esta función recursivamente en la rama falsa
    print (spacing + '--> Falso:')
    print_tree(node.false_branch, spacing + "  ")
```

- **CLASIFICACIÓN**

```python
def classify(row, node):
    # Caso base: hemos llegado a una hoja
    if isinstance(node, Leaf):
        return node.predictions

    # Decide si seguir la rama verdadera o la rama falsa.
    # Comparar la característica/valor almacenado en el nodo,
    # al ejemplo que estamos considerando.
    if node.question.match(row):
        return classify(row, node.true_branch)
    else:
        return classify(row, node.false_branch)
```

Bibliografía

- An Introduction to Statistical Learning: with Applications in R (Springer Texts in Statistics)
- API design for aprendizaje automático software: experiences from the scikit-learn project, Buitinck et al., 2013.
- Árboles de decisión con Python: regresión y clasificación by Joaquín Amat Rodrigo, available under a Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/py07_arboles_decision_python.html
- Introduction to aprendizaje automático with Python: A Guide for Data Scientists
- Lewis, Roger. (2000). An Introduction to Classification and Regression Tree (CART) Analysis.
- Mohammad A. (2018) A Comprehensive Study of Decision Trees to Classify DNA Sequences. Universidad de Nuevo México.
- Rokach, Lior. (2016). Decision Forest: Twenty Years of Research. Information Fusion. 27. 10.1016/j.inffus.2015.06.005.
- Roche, A. (2009.). *Árboles de decisión y series de tiempo.* Tesis de maestría. Universidad de la República (Uruguay). Facultad de Ingeniería.

Webgrafía

- https://platzi.com/tutoriales/2459-machine-learning/12028-clase-tipos-de-modelos-de-machine-learning/
- https://www.sciencedirect.com/topics/computer-science/decision-tree
- https://www.analyticssteps.com/blogs/classification-and-regression-tree-cart-algorithm
- https://spark.apache.org/docs/latest/api/python/