Árboles de decisión

¿Has jugado alguna vez al juego de las 20 preguntas? Si no, el juego consiste en que una persona piensa en algún objeto y los demás jugadores tratan de averiguarlo. Cada jugador tiene un máximo de 20 preguntas y recibe solamente respuesta de si/no. El árbol de decisión funciona igual que este juego: le proporcionas unos datos y te proporciona una respuesta.

Los árboles de decisión son uno de los métodos de clasificación más usados. No hace falta saber demasiado de aprendizaje automático para entender cómo funciona. El árbol de decisión está formado por bloques de decisión y las hojas (en las que se alcanza alguna conclusión). Las flechas que indican true/false se llaman ramas.

A diferencia del KNN los árboles de decisión son fácilmente entendibles por los humanos.

El algoritmo que se construirá en este tema tomará un conjunto de datos y construiremos el árbol de decisión. El árbol transformará el conjunto de datos en conocimiento/información.

Con esto, podremos coger un conjunto de datos desconocidos y extraer un conjunto de reglas. Los arboles de decisión son usados en sistemas expertos y los resultados obtenidos son comparables a las de un experto que tiene décadas de experiencia en el campo.

Construcción del árbol

Pros: computacionalmente bajo, resultados fáciles de entender para los humanos, valores null OK, puede tratar con atributos irrelevantes.

Contras: propenso al sobreajuste

Funciona con: valores numéricos, nominales.

Para construir un árbol de decisión lo primero que hay que hacer es tomar una decisión sobre el conjunto de datos: ¿Qué atributo usaremos para clasificar los datos? Para solucionar esto, iremos probando todos los atributos y midiendo cual nos proporciona los mejores resultados.

Después de haber separado los datos partiremos el conjunto de datos en subconjuntos, los cuales atravesarán hacia abajo el árbol de decisión. Si los datos en las ramas son de la misma clase, entonces habremos clasificado apropiadamente y no necesitaremos

continuar. Si por el contrario no lo es, necesitaremos repetir el proceso en el subconjunto (el método para separar se hace igual que en el conjunto original de datos), así hasta que logremos tener todos los datos clasificados.

Como se puede observar, la función createBranch() tiene un carácter recursivo.

1. **Obtención de datos:** cualquier método.
2. **Preparación:** este algoritmo de construcción de árbol funciona solo con valores nominales, por lo que cualquier valor continuo necesitará ser cuantizado.
3. **Análisis:** Cualquier método. Deberemos inspeccionar visualmente el árbol tras su construcción.
4. **Entrenamiento:** Construir una estructura de árbol de datos ¿?
5. **Test:** calcular el porcentaje de error con el árbol entrenado.
6. **Uso:** puede ser usado en cualquier tare de aprendizaje supervisado. A veces los árboles son usados para entender mejor los datos.

Algunos árboles de decisión hacen una separación binaria de los datos, pero nosotros no haremos esto. Si separamos por un atributo y el atributo tiene cuatro posibles valores, entonces partiremos los datos en 4 caminos y crearemos 4 ramas separadas. Seguiremos el algoritmo ID3, que nos dice como separar los datos y cuando parar de hacerlo. También vamos separar por uno y solo un atributo. Si nuestro conjunto de datos tiene 20 atributos ¿cómo elegimos uno para usar primero?

Ganancia de información (Information gain)

Elegimos separar nuestro conjunto de datos de manera que convierte nuestro conjunto desorganizado de datos en otro más organizado. Hay varias maneras de realizar esto y cada una tiene sus ventajas y sus desventajas.

Una manera es la de medir la información: usando la teoría de la información, se puede medir la información antes y después de la separación de los datos. La teoría de la información es una rama de la ciencia que concierne a la cuantificación de la información.

El cambio en la información antes y después de la separación es conocido como “information gain”. Cuando sabes cómo calcular el “information gain” puedes separar los datos con todas las características para ver cuál te da mayor “information gain”. El Split con la mayor “information gain” es la mejor opción.

Antes de que podamos medir el mejor Split y comenzar a separar nuestros datos, necesitamos saber cómo calcular el "information gain". La medida de información de un conjunto de datos es conocido como **Shannon Entropy** o solamente **Entropy** (el nombre viene del padre de la teoría de la información).

Si los términos de information gain y entropy te parecen confusos no te preocupes. La entropía se define como el valor esperado de la información. Lo primero que necesitamos es definir la información. Si estamos clasificando algo que puede tomar múltiples valores la ecuación que se usa es:

$$l(x_i) = log_2 p(x_i)$$

Donde $p(x_i)$ representa la probabilidad de este esta clase.

Para calcular la entropía, necesitamos el valor esperado de toda la información de todos los posibles valores de nuestra clase. La fórmula que se usa para la entropía es:

$$H = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

Donde n es el número de clases.

Vamos a ver como calcularlo en Python, para lo cual primero crearemos un archivo llamado trees.py, el cual nos calculará la entropía para un conjunto de datos dados.

(Código pag 42) Python notebook (Machine Learning in Action).

Vamos a separar los datos de manera que nos de la mayor "information gain". No sabremos cómo hacerlo a no ser que separemos el conjunto de datos y midamos la "information gain".

Splitting the dataset

En la anterior sección vimos cómo medir la cantidad de desorden en un conjunto de datos. Para nuestro algoritmo de clasificación funcione, necesitamos medir la entropía, separar el conjunto de datos, medir la entropía sobre los conjuntos de datos separados y ver si separar los datos fue lo correcto o no. Haremos estos para todas las características para determinar la mejor característica para separar los datos. Lo que queremos hacer es pintar una línea para separar una clase de otra. ¿debería hacerlo en el eje X o en el eje Y? Eso es lo que estamos intentando averiguar en esta sección.

(Código pág. 43) Python notebook (Machine Learning in Action).

El código tiene tres parámetros: el conjunto de datos que separaremos, la característica por la que separaremos y el valor de la característica para devolver. La mayoría de las veces no tenemos que preocuparnos por la reserva de memoria. Python pasa las listas por referencia, por lo que, si modificas una lista en una función, la lista será modificada en todas partes.

(Código pág. 44) Python notebook (Machine Learning in Action).

Ahora vamos a combinar la Entropía de Shannon y el splitDataSet() para ciclar a través del conjunto y decidir cuál es la característica mejor para separar los datos en conjuntos.

(Código pág. 44) Python notebook (Machine Learning in Action).

CONSTRUYENDO EL ÁRBOL RECURSIVAMENTE

Ahora que ya tenemos todos los componentes necesarios para crear un algoritmo que cree el árbol de decisión a partir de un conjunto de datos dado. Esto se hace de la siguiente manera:

Empezamos con nuestro conjunto de datos inicial y lo partimos por el mejor atributo (viste en la sección anterior). Estos no son arboles binarios, por lo que puedes más de dos ramas en el árbol. Una vez separado el conjunto de datos original, los datos atravesarán el árbol por las ramas a otro nodo, el cual volverá a separar los datos etc...

La recursión parará cuando se den las siguientes condiciones: nos quedamos sin atributos por los que separar o que todos los datos de la rama sean de la misma clase. Si todos los datos tienen la misma clase entonces crearemos un nodo hoja.

(Código pág. 47) Python notebook (Machine Learning in Action).

Graficando los árboles con Matplotlib en Python

El árbol creado en la sección anterior vamos a visualizarlo. Usaremos Matplotlib para crear un árbol que se entienda con solo mirarlo. Una de las ventajas de los árboles es que los humanos podemos entenderlos fácilmente. Python no incluye una buena herramienta para pintar los árboles, por lo que haremos la nuestra propia.

Anotaciones matplotlib

Matplotlib tiene una herramienta muy buena llamada anotaciones, que puede añadir texto en el plot. Las anotaciones son usadas para explicar alguna parte de los datos.

(Código pág. 50) Python notebook (Machine Learning in Action).

Construyendo un árbol de anotaciones

Necesitamos una estrategia para pintar este árbol. Tenemos las coordenadas X e Y. ¿Donde colocamos los nodos? Necesitamos saber cuántos nodos hoja tenemos para poder pintar adecuadamente, así como lo niveles de profundidad del árbol

(Código pág. 51) Python notebook (Machine Learning in Action).

(Código pág. 52) Python notebook (Machine Learning in Action).

(Código pág. 53) Python notebook (Machine Learning in Action).

(Código pág. 55) Python notebook (Machine Learning in Action).

Testing y almacenamiento del clasificador

Ya hemos conseguido construir el árbol y plotearlo pero no hemos realizado ninguna clasificación. En esta sección construiremos un clasificador que usa nuestro árbol y luego veremos como afecta al almacenamiento del clasificador. Haremos un ejemplo práctico para predecir qué tipo de lentes de contacto debería usar una persona.

Test: usando el árbol de clasificación

¿Queremos utilizar el árbol que hemos construido a partir de nuestros datos de entrenamiento, pero como lo hacemos? Necesitamos el árbol y el vector con las etiquetas que usamos en la creación del árbol.

(Código pág. 56) Python notebook (Machine Learning in Action).

Persistiendo el árbol de decisión

Construir el árbol es lo más costoso, toma unos pocos segundos con conjuntos de datos pequeños, pero con conjuntos de datos grandes puede tomar un tiempo. A la hora de clasificar elementos con un árbol se hace rápidamente. Sería una pérdida de tiempo construir el árbol cada vez que quieras realizar una clasificación. Para quitarnos este problema, usamos un módulo de Python llamado pickle, que serializa objetos, lo que permite almacenar los objetos para usarlos más tarde.

(Código pág. 57) Python notebook (Machine Learning in Action).

Ahora ya tenemos el árbol almacenado, por lo que no habrá que construirlo cada vez que queramos clasificador algo con él.

Me he quedado pág. 58.