IIC2433 — Minería de Datos

Apuntes: Ensamblaje y Árboles

1. Árboles de decisión

Un árbol de decisión es otro modelo para clasificar[1]. Asumimos que en este curso todos sabemos usar árboles de decisión. La pregunta a responder es: ¿Cómo entrenar un árbol de decisión? Vamos a mostrar un posible algoritmo, basado en dos algoritmos, ID3 y C4.5, inventados por Ross Quinlan. Veamos un poco de notación.

1.1. Entropía

La entropía puede verse como una métrica de la incerteza sobre un dataset dado. Veamos primero un ejemplo. Imaginemos que tenemos dos tipos de bombones: de chocolate y de mazapán, y tenemos una bolsa con 10 de cada uno. Queremos calcular la entropía de mi bolsa de bombones.

La entropía, aplicada a clases de objetos sacados de un set S de datos, se define como

$$H(S) = -c(x)\log_2 c(x) - m(x)\log_2 m(x),$$

donde $c(x)$ es la proporción de bonbones de chocolate y $m(x)$ la de bombones de mazapan. En nuestro caso, $c(x) = m(x) = \frac{10}{20} = 0,5$. Resolvemos que $H(S) = 1$ en este caso: la entropia es máxima, por que tenemos absoluta incerteza, si sacamos un bombon al azar es igual de probable que sea de chocolate o de mazapán.

Ahora, si tenemos dos bombones de chocolate, y 18 de mazapán, $H(S) = -0{,}1 * -3{,}32 - 0{,}9 * -0{,}15$, lo que da algo así como 0,16: la entropía es mucho menor por que podemos asegurar con bastante certeza que el sacar un bombon al azar va a resultar en uno de mazapán. Puedes comprobar que si los 20 bombones son de mazapán, entonces $H(S) = 0$.

En términos generales, si tengo un dataset S, particionados en clases $c_1, \ldots, c_k$, cada una con $c_\ell(S)$ elementos, la entropía se define como

$$H(S) = \sum_{1 \leq \ell \leq k} -c_\ell(S)\log_2 c_\ell(S)$$

1.2. Ganancia de información

Supongamos que tenemos un set de datos $X = \bar{x}^1, \ldots, \bar{x}^m$, compuesto de m entidades sobre atributos $A_1, \ldots, A_n$. Cada una de esas entidades recibe un valor y_j, que representa

[1] En estricto rigor, pueden usarse para hacer regresiones también, no lo vamos a ver acá

una clase a clasificar. Hay k clases en total, $c_1, \ldots, c_k$, cada una con $c_\ell(S)$ elementos, luego y_j puede representar la clase con un número del 1 al k. La entropía del dataset es entonces

$$H(X) = \sum 1 \leq \ell \leq k \frac{c_\ell(S)}{|S|} \log_2 \frac{c_\ell(S)}{|S|}$$

Supongamos que en un árbol de decisión, en un nivel dado, tenemos actualmente T hojas, las que representan una partición de nuestro set de datos X. Cada hoja t tiene $|t|$ elementos. Definimos la entropía de ese nivel como

$$H(X\text{particionado en } T) = \sum t \in T \frac{|t|}{|S|} H(t)$$

Finalmente, dado un dataset X, la ganancia de información al particionar en T hojas es

$$IG(X, T) = H(X) - H(X\text{particionado en } T)$$

1.3. Clases en el dataset

Hasta ahora siempre hemos hablado de que nuestro dataset tiene *particiones*. ¿Pero de qué estamos hablando realmente?

Notar que cada atributo en X nos sirve para definir un número de clases:

- Si el atributo es categórico, podemos definir que ese atributo corta el dataset en C particiones, donde C es la cantidad distinta de categorías en ese atributo. Por ejemplo, si un atributo, usado para almacenar el género de una persona, tiene opciones M, F, NB, ND eso da lugar a 4 particiones, una por cada categoría, y luego particionamos X en $X = X_M \cup X_F \cup X_{ND} \cup X_{ND}$.
- Si el atributo es numérico, podemos tomar la media o la mediana de ese valor (si distribuye normal eso es lo mismo, pero no necesariamente los datos son normales siempre), y cortar X en dos: una partición para los valores por sobre la media o mediana, y otra para los valores bajo o igual a la media o mediana.
- Si el atributo no es ni categórico ni numérico (por ejemplo, strings), tenemos que transformarlo de alguna forma a algo categórico o numérico.

Así, podemos pensar en que cada atributo particiona a X. Si tenemos atributos $A_{i_1}, \ldots, A_{i_\ell}$ y cada atributo A particiona en C pedazos, entonces todos esos atributos juntos particionan a X en $C_{i_1} \cdot \cdots \cdot C_{i_\ell}$ pedazos distintas.

1.4. Entrenando un árbol de decision

Podemos pensar entonces en el siguiente algoritmo.

Requisito: Para nuestro dataset completo S, las posibles particiones de S según cada uno de los atributos $A_1, \ldots, A_n$. Llamamos P_i a esas particiones.

Input: Un dataset X, un dataset Y que asocia una de k clases $c_1, \ldots, c_k$ a cada entidad $\bar{x}^j$ en X. Llamamos y_j a esa clase.
Output: Un árbol de decisión para X.

1. Si todos los y_j en Y son iguales, entonces el árbol es una hoja, etiquetada con esa clase.
2. Elegir el atributo A_i de X que maximiza $IG(X, A)$ (recordemos que A particiona a S en P_i particiones).
3. Añadir un nodo en el árbol que haga el test si x pertenece a algún pedazo en P_i, conectado a una rama por cada $p \in P_i$
4. Para cada pedazo $p \in P_i$, sea X_p las entidades que pertenecen a la categoría p.
    - Si X_p es vacío, sea c la clase más numerosa en Y. La rama conecta a una hoja con la etiqueta c.
    - Si X_p no es vacío, sea Y_p los valores de la clase de cada entidad en X_p y sea $\pi_{-A_i} X_p$ el dataset X_p sin la columna A_i. La rama conecta con el árbol para $(\pi_{-A_i} X_p, Y_p)$.

2. Ensemble learning

Los árboles de decisión, por su simpleza, son unos buenos candidatos para introducir un concepto importante del aprendizaje de máquina: el aprendizaje mediante ensambles. La idea, en simple, es utilizar varios modelos en vez de uno. De esta forma, las predicciones de todos los modelos son agrupadas en una sola, y generalmente vamos a ver como muchos modelos (aunque quiza son mas malos individualmente) terminan aprendiendo mejor que uno.

2.1. Bagging y random forest

La idea detrás de un modelo que usa bagging (por Bootstrap AGGregation) es producir distintos pedazos del dataset, y luego combinar sus predicciones. Una forma básica de describirlo es con el siguiente pseudocodigo:
Para entrenar:

- Sea N el número de modelos a agregar. Para cada $i \leq N$:
    - Obtener un pedazo p del dataset eligiendo tuplas al azar
    - Construir el modelo M_i entrenando dobre esas p tuplas

Para clasificar:

- Dada una tupla $\bar{x}$, para predecir el y (donde clasificamos cada y a una de k clases):

- Si cada M_i entrega una probabilidad de pertenecer a cada clase, entonces la probabilidad total de pertenecer a cada clase es el promedio de las probabilidades de esa clase según cada M_i.
- Si cada M_i solo entrega la clase (sin probabilidad), entonces el modelo agregado predice la clase que obtiene la mayoría entre cada M_i, y si hay un empate, se determina la clase al azar.

Por supuesto, la técnica de *bagging* ha sido largamente estudiada, con muchas propuestas de mejoras o de caminos alternativos. Un ejemplo importante corresponde a **Random Forest**. Este predictor corresponde a un modelo entrenado de acuerdo a la guía de arriba, en donde cada M_i es un árbol de decisión, pero en donde además, al elegir el pedazo p del dataset para entrenar uno de los M_i, se elige un pedazo al azar también de atributos, y solo se entrena M_i con esos atributos.

2.2. Boosting

Boosting, en cambio, tiene que ver con unir clasificadores que se entrenan sobre los errores de otros clasificadores. Hay distintas formas de hacer boosting, pero la regla general es la siguiente.

- Se entrena primero un árbol, y se obtiene una medida del error de clasificación de ese árbol
- Usando esa medida del error, se entrena otro árbol orientado específicamente a tratar a ese error (por ejemplo, a clasificar donde se equivoca el árbol anterior)
- Ambos árboles son unificados usando alguna forma de agregación (por ejemplo, ponderando sus clasificaciones)
- Se repite este proceso una serie de veces, cada nueva vez sobre el error del árbol agregado.

Puedes buscar más sobre algoritmos famosos para hacer boosting que instancian esta regla general. Sugerimos mirar AdaBoost, y luego XGBoost y GBM o LightGBM. Estos últimos dos tienen una componente de error más compleja, por que integran además la dirección en donde se minimiza el error.