Capítulo 1

Introducción a ciencia de datos

1.1. -

1.2. -

1.3. Análisis predictivo

Un problema común en ciencia de datos es identificar el grupo (clase) al cual pertenece una observación, utilizando un conjunto de atributos de dicha observación. Por ejemplo, muchos clientes de email envían los mensajes recibidos a diferentes carpetas, tales como *entrada*, *promociones*, etc. En ocasiones los mensajes contienen *links* a sitios web con malware, en otras son mensajes enviados masivamente, es decir a gran cantidad de personas; resulta deseable que el cliente pueda enviar este tipo de mensajes a la carpeta de *spam* o ponerlos en cuarentena. Sin embargo, es raro que un mensaje entrante contenga todos los atributos para clasificarlo como peligroso, por tanto debe usarse una función predictiva.

Un algoritmo para clasificar mensajes de correo debe extraer datos de los mensajes, tales como la presencia de palabras específica o el número de letras mayúsculas utilizadas; con esta información se genera un vector de atributos predictores y se procesa con la función de predicción; esta función se encargará de devolver la clase a la que pertenece el mensaje recibido. Antes de utilizar una función para predecir, se debe revisar a detalle para buscar que tenga alta precisión; el tópico de la predicción y el aseguramiento de la precisión se conoce como *análisis predictivo*[1].

1.3.1. La tarea de predicción

El propósito de este tipo de algoritmos es obtener el valor de una variable objetivo a partir de un vector predictor; generalmente el objetivo es una variable categórica: una etiqueta que indica el grupo al que pertenece la observación. El analista no posee datos de la etiqueta pero, en cambio, tiene información codificada en los atributos de su vector.

[1]Este es el término usado en ciencia de datos; en estadística se conoce como aprendizaje estadístico y *machine learning* (aprendizaje automático) en computación.

1.3.2. Funciones de predicción $k-$vecinos

Una función de predicción de este tipo utiliza un conjunto de de observaciones en pares $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$ para los cuales se conocen todos los y_i valores objetivo; una observación objetivo es un par $(y_0, \mathbf{x}_0)$ para el que se desea calcular el valor de su variable objetivo y_0. La versión más sencilla de esta función de predicción para problemas de clasificación opera determinando las k observaciones más cercanas (similares) a $(y_0, \mathbf{x}_0)$, basada en las distancias entre $\mathbf{x}_0$ y $\mathbf{x}_i$; el valor de la predicción será la etiqueta más entre las observaciones más cercanas: sus $k-$vecinos. En el caso de variables cuantitativas, la predicción del algoritmo básico es la media de sus $k-$vecinos.

1.3.2.1. Notación

Una observación con valor objetivo y_0 desconocido, se denota como $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$, en donde el vector predictor $\mathbf{x}_0$ de longitud p ha sido observado y se utilizará para predecir el valor de y_0. La función de predicción $f(\cdot|D)$ se construye a partir del conjunto de datos $D = \{\mathbf{z}_1, \ldots, \mathbf{z}_n\}$; la notación condicional de $f(\cdot|D)$ enfatiza el papel del conjunto de entrenamiento D. La predicción de y_0 se denota como $\hat{y}_0 = f(\mathbf{x}_0|D)$. Por ejemplo, si el objetivo es un valor **cuantitativo**, se puede utilizar una regresión lineal como función predictiva: $f(\mathbf{x}_0|D) = \mathbf{x}_0^T \hat{\boldsymbol{\beta}}$.

En general, la función de predicción no tiene una forma simple y se trata de un algoritmo de varios pasos que toma como entrada $\mathbf{x}_0$ y devuelve la predicción y_0. Si la variable es **cualitativa**, se asume que la variable objetivo es una etiqueta que identifica la pertenencia a algún grupo; si el número de grupos es g, entonces por convención el conjunto de etiquetas es $\{0, 1, \ldots, g-1\}$.

Además, es recomendable contar con funciones indicadoras que identifiquen la pertenencia a los grupos; el indicador de pertencia al grupo j se define como:

$$I_j(y) = \begin{cases} 1, & \text{si } y = j \\ 0, & \text{si } y \neq j \end{cases},$$

donde y es una etiqueta de grupo.

1.3.2.2. Métricas de distancia

Es común utilizar las distancias euclideana y Manhattan para calcular distancias entre vectores predictores.

La **distancia euclideana** entre los vectores $p-$dimensionales $\mathbf{x}_0$ y $\mathbf{x}_i$ se obtiene como:

$$d_E(\mathbf{x}_i, \mathbf{x}_0) = \left[\sum_{j=1}^{p} (x_{i,j} - x_{0,j})^2\right]^{1/2}$$

Si las variables del predictor difieren mucho con respecto a la variabilidad de las variables del conjunto de entrenamiento, es muy recomendable escalar cada variable con la desviación estándar de la misma; sin este proceso, las diferencias serán muy grandes y tenderán a sesgar la predicción. El escalamiento puede calcularse al momento de obtener la distancia como:

$$d_S(\mathbf{x}_i, \mathbf{x}_0) = \left[\sum_{j=1}^{p} \left(\frac{x_{i,j} - x_{0,j}}{s_j}\right)^2\right]^{1/2},$$

donde s_j es la desviación estándar estimada de la $j-$ésima variable predictora; la varianza se calcula como:

$$s_j^2 = (n-g)^{-1} \sum_{k=1}^{g} \sum_{i=1}^{n} I_k(y_i)(x_{i,j} - \bar{x}_{j,k})^2,$$

donde $\bar{x}_{j,k}$ es la media del atributo j dentro del grupo k; la presencia de $I_k(y_i)$ asegura que en la sumatoria interna sólo se consideren las observaciones del grupo correspondiente.

La **distancia Manhattan** (*city-block*) entre $\mathbf{x}_0$ y $\mathbf{x}_i$ se calcula como:

$$d_C(\mathbf{x}_i, \mathbf{x}_0) = \sum_{j=1}^{p} |x_{i,j} - x_{0,j}|,$$

de igual forma, puede utilizarse escalamiento para evitar sesgos en el cálculo.

En realidad en la mayoría de las aplicaciones de este algoritmo, es más impotante elegir el tamaño k del vecindario, dado que las métricas suelen devolver ordenamientos similares para los vecindarios.

Si un predictor consiste de p variables **cualitativas**, entonces la distancia más usada es la de **Hamming** para comparar $\mathbf{x}_0$ y $\mathbf{x}_i$; esta métrica determina el número de atributos que son diferentes entre ambos vectores; se calcula como:

$$d_H(\mathbf{x}_i, \mathbf{x}_0) = p - \sum_{j=1}^{p} I_{x_{i,j}}(x_{0,j}),$$

nótese que $I_{x_{i,j}}(x_{0,j})$ es 1 siempre que $x_{i,j} = x_{0,j}$ y 0 en otro caso; entonces la suma cuenta el número de atributos distintos entre ambos vectores.

1.3.2.3. La función de predicción de los $k-$vecinos más cercanos

La predicción de y_0 con el algoritmo de $k-$vecinos se obtiene determinando un vecindario de las k observaciones de entrenamiento más cercanas a $\mathbf{x}_0$; después se calcula la proporción de los k vecinos que pertenecen al grupo j y se predice el valor de y_0 como el grupo con la mayor proporción entre los vecinos. Esta regla de predicción es equivalente a predecir que $\mathbf{z}_0$ pertenece al grupo más común entre los k vecinos más próximos.

Formalmente, la función de predicción que estima la probabilidad de pertenencia al grupo j se define como la proporción de los miembros del grupo j entre los k vecinos más cercanos:

$$\widehat{Pr}(y_0 = j|\mathbf{x}_0) = \frac{n_j}{k}, j = 1, \ldots, g,$$

donde n_j es el número de vecinos dentro de los k más próximos que pertenecen a j. Resulta útil tener una expresión para $\hat{Pr}(y_0 = j|\mathbf{x}_0)$ en términos de variables indicadoras:

$$\widehat{Pr}(y_0 = j|\mathbf{x}_0) = k^{-1} \sum_{i=1}^{k} I_j\left(y_{[i]}\right)$$

Los corchetes denotan el valor objetivo del $i-$ésimo vecino más cercano; es decir, el vecino más próximo $\mathbf{z}_{[1]}$ tiene como valor objetivo $y_{[1]}$. Las probabilidades de pertenencia estimadas se agrupan como un vector:

$$\underset{g\times 1}{\widehat{\mathbf{p}}_0} = \left[\widehat{Pr}(y_0 = 1|\mathbf{x}_0), \ldots, \widehat{Pr}(y_0 = g|\mathbf{x}_0)\right]^T$$

El último paso para calcular la predicción obtiene el valor más grande dentro de $\widehat{\mathbf{p}}_0$:

$$f\left(\mathbf{x}_0|D\right) = \arg\max\left(\widehat{\mathbf{p}}_0\right)$$

La función arg máx devuelve el índice del elemento más grande de un vector, en este caso, de $\widehat{\mathbf{p}}_0$; en el caso de existir empate entre varios valores, devolverá el primero de ellos. Sin embargo, es mejor idea romper los empates de otra forma, por ejemplo incrementando el valor de k en uno y recalculando $\widehat{\mathbf{p}}_0$.

Un algoritmo computacional eficiente para la predicción con k vecinos consiste de dos funciones principales:

♢ obtener el arreglo ordenado $\mathbf{y}^o = \left(y_{[1]}, \ldots, y_{[n]}\right)$ para las entradas $\mathbf{x}_1, \ldots, \mathbf{x}_n$

♢ calcular los elementos de $\widehat{\mathbf{p}}$ y utilizar los primeros k elementos de $\mathbf{y}^o$

Las funciones de predicción $k-$vecinos más cercanos son conceptualmente simples y rivalizan con otras más sofisticadas en cuanto a la precisión de sus predicciones; el problema principal es que este tipo de algoritmos pueden ser computacionalmente muy costosos.

1.3.2.4. $k-$vecinos más próximos ponderados exponencialmente

El algoritmo convencional puede mejorarse en precisión; la idea es utilizar todos los vecinos y no solamente k de ellos. A cada vecino se le asigna una medida de importancia (peso) para determinar $f\left(\mathbf{x}_0|D\right)$ de acuerdo a su distancia relativa con $\mathbf{x}_0$; de esta forma, el vecino más próximo recibe el mayor peso y los demás decaen a cero conforme nos movemos a vecinos más distantes. Además, esta forma del algoritmo evita el problema de los empates en la predicción.

Primero, se debe obtener la suma de los pesos para todos los n vecinos; los pesos son:

$$w_i = \begin{cases} 1/i, & \text{si } i \leq k \\ 0, & \text{si } i > k \end{cases},$$

para $i = 1, \ldots, n$; con esto el estimador alternativo de la probabilidad de pertenencia

$$\widehat{Pr}\left(y_0 = j|\mathbf{x}_0\right) = \sum_{i=1}^{k} w_i I_j\left(y_{[i]}\right)$$

Este estimador indica que los pesos son iguales a $1/k$ para los primeros k vecinos y se vuelven 0 para el resto; es difícil justificar esta caída tan abrupta en los valores; una forma más plausible es que el contenido de la información decaiga con cada vecino más lejano, es decir, que tengan una caída suavizada.

El algoritmo de *$k-$vecinos más próximos ponderados exponencialmente* estima la probabilidad de pertenencia utilizando un conjunto de pesos diferentes. En la función convencional, la influencia de los vecinos está determinada por el tamaño k del vecindario; en cambio, con el algoritmo de pesos ponderados, la influencia de los vecinos depende de una constante α, un valor entre 0 y 1 que sirve para afinar la predicción; los pesos se definen como:

$$w_i = \alpha\left(1-\alpha\right)^{i-1}, i = 1, \ldots, n,$$

para $0 < \alpha < 1$; la suma de los pesos será aproximadamente 1, dado que n es grande.

Tarea:

♢ Demuestra que para $0 < \alpha < 1$, se cumple que $1 = \sum_{i=1}^{\infty} \alpha\,(1-\alpha)^{i-1}$

*

La parte izquierda de la figura 1.1 muestra los pesos correspondientes al algoritmo convencional de $k-$vecinos para valores de 3, 5, 10, y 20; mientras que el derecho presenta los pesos similares en el algoritmo ponderado exponencialmente para diferentes valores de α. Es importante notar que los pesos en $0.33\bar{3}$, 0.2, 0.1 y 0.05 son iguales a los pesos correspondientes a los valores recíprocos de k.

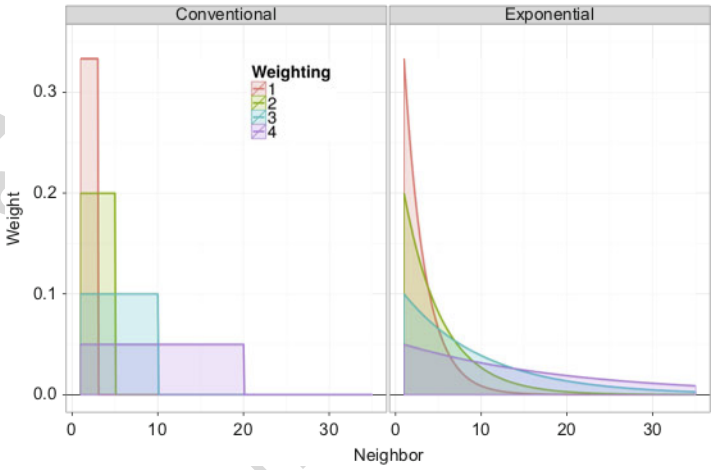


Figura 1.1: $k-$vecinos más próximos ponderados exponencialmente

La elección de α, generalmente es simple si se toma en cuenta que el vecino más próximo recibe el peso α; por ejemplo, $\alpha = 0.2$ implica que el vecino más cercano recibirá el mismo peso que corresponde a la función de predicción convencional de $5-$vecinos más cercanos.

1.3.2.5. Ejemplo

Construcción de un sistema de recomendación de *anime* basado en similitudes.

```
# bibliotecas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# datos
anime = pd.read_csv("https://bit.ly/2kiJkrW")
anime.head()
```

```
# verificando nulos
anime.isnull().sum()
```

Preprocesamiento

episodes

Muchos animes tienen un número desconocido de episodios e incluso algunos que siguen al momento, se especifican como *"unknown"*, en la fuente de los datos se especifica que algunos fueron llenados manualmente (*known_ animes*)

Para otros se toman decisiones fundamentadas:

♢ Animes agrupados en la categoría *Hentai* generalmente tienen sólo 1 episodio

♢ Animes agrupados en *OVA* (*Original Video Animation*) generalmente tienen 1 ó 2 capítulos (los más populares 2 ó 3); se decidió por en valor 1

♢ Animes agrupados como *Movies* se consideran de 1 episodio

♢ Para todos los demás que tienen valor *"unknown"*, se llenan con la mediana (2)

```
known_animes = {"Naruto Shippuuden":500,"One Piece":784,"Detective Conan":854,
                "Dragon Ball Super":86,"Crayon Shin chan":942,
                "Yu Gi Oh Arc V":148,"Shingeki no Kyojin Season 2":25,
                "Boku no Hero Academia 2nd Season":25,
                "Little Witch Academia TV":25}
#known_animes
for k,v in known_animes.items():
  anime.loc[anime["name"]==k,"episodes"] = v

anime.loc[(anime["genre"]=="Hentai") & (anime["episodes"]=="Unknown"),
                 "episodes"] = "1"
anime.loc[(anime["type"]=="OVA") & (anime["episodes"]=="Unknown"),
                 "episodes"] = "1"
anime.loc[(anime["type"] == "Movie") & (anime["episodes"] == "Unknown")] = "1"

anime["episodes"]=anime["episodes"].map(lambda x:np.nan if x=="Unknown" else x
anime["episodes"].fillna(anime["episodes"].median(),inplace = True)
```

type: se puede usar *get_ dummies*

```
pd.get_dummies(anime[["type"]]).head()
```

members: sólo se convierte a *float*
rating: los valores faltantes se sustituyen por la mediana
genre: se usan *dummies*

```
anime["members"] = anime["members"].astype(float)
anime["rating"] = anime["rating"].astype(float)
anime["rating"].fillna(anime["rating"].median(),inplace = True)
```

```
# Concatenando todas las características
anime_features = pd.concat([anime["genre"].str.get_dummies(sep=","),
                            pd.get_dummies(anime[["type"]]),
                            anime[["rating"]],anime[["members"]],
                            anime["episodes"]],axis=1)
# eliminar los símbolos 'raros' de los nombres
import re
anime["name"]=anime["name"].map(lambda name:re.sub('[^A-Za-z0-9]+'," ",name))
anime_features.head()
```

"*episodes*", "*members*" y "*rating*" tienen valores muy distintos a las variables categóricas para evitar sesgos se utiliza *MinMaxScaler*:

```
# Escalando
from sklearn.preprocessing import MinMaxScaler
min_max_scaler = MinMaxScaler()
anime_features = min_max_scaler.fit_transform(anime_features)
# resultado del escalamiento
np.round(anime_features,2)
```

```
# modelo KNN
from sklearn.neighbors import NearestNeighbors
```

```
# BallTree for fast generalized N-point problems
#https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.BallTree.
nbrs = NearestNeighbors(n_neighbors=6, algorithm='ball_tree')
nbrs.fit(anime_features)

distances, indices = nbrs.kneighbors(anime_features)
```

Ejemplos de consultas:

A veces los nombres de los animes en japones e inglés no son exactamente iguales, por eso se requieren algunas funciones auxiliares.

```
all_anime_names = list(anime.name.values)
# funciones auxiliares
def get_index_from_name(name):
    return anime[anime["name"]==name].index.tolist()[0]
def get_id_from_partial_name(partial):
    for name in all_anime_names:
        if partial in name:
            print(name,all_anime_names.index(name))
```

```
# uso de get_id_from_partial_name
get_id_from_partial_name("Naruto")
```

```
# busca animes similares, puede ser por id o por nombre
def print_similar_animes(query=None,id=None):
    if id:
        for id in indices[id][1:]:
            print(anime.iloc[id]["name"])
    if query:
        found_id = get_index_from_name(query)
        for id in indices[found_id][1:]:
            print(anime.iloc[id]["name"])
```

```
# uso de print_similar_animes
print_similar_animes(id=719)

print_similar_animes(query="Naruto")

print_similar_animes("Noragami")

print_similar_animes("Gintama")

print_similar_animes("Fairy Tail")
```

Original:
https://gist.github.com/Tahsin-Mayeesha/81dcdafc61b774768b64ba5201e31e0a

1.3.2.6. Regresión $k-$vecinos

Las funciones de predicción $k-$vecinos pueden adaptarse para ser usadas con variables objetivo cuantitativas: para predecir variables cuantitativas, las funciones $k-$vecinos son alternativas a funciones de predicción basadas en regresión.

El objetivo es predecir la variable objetivo y_0, usando un vector predictor $\mathbf{x}_0$ y una función entrenada con un conjunto de pares de variables objetivo y predictoras $D = \{(y_1, \mathbf{x}_1), \ldots, (y_n, \mathbf{x}_n)\}$. La predicción es un promedio ponderado del conjunto ordenado de objetivos $y_{[1]}, \ldots, y_{[n]}$ dado por:

$$\widehat{y} = \sum_{j=1}^{n} w_i y_{[i]}$$

donde el orden $y_{[1]}, \ldots, y_{[n]}$ se determina con las distancias desde $\mathbf{x}_0$ hasta $\mathbf{x}_1, \ldots, \mathbf{x}_n$; los w_i pueden ser tanto los pesos usados función $k-$vecinos convencional como los usados en la función $k-$vecinos exponencialmente ponderados.

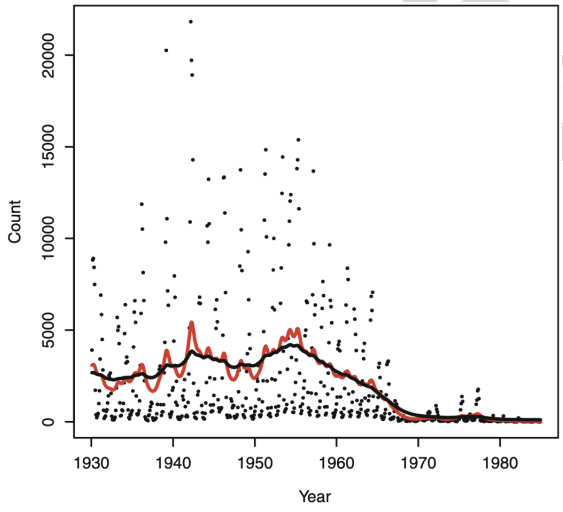


Figura 1.2: $k-$vecinos más próximos usado para regresión

La figura 1.2 muestra el número de casos reportados de **sarampión** por mes en California entre 1930 y 1985; adicionalmente, muestra el número de casos predicho por la función de regresión $k-$vecinos ponderados exponencialmente para dos valores de la constante de *suavizado* α: Las predicciones para el valor $\alpha = 0.05$ (rojo) es mucho menos suave que las predicciones obtenidas con $\alpha = 0.02$ (negro). Estas líneas, que muestran ciertas *tendencias* al eliminar variaciones de corto plazo, se conocen comúnmente como *suavizados*.

En la gráfica se puede observar que existió gran cantidad de variaciones en el número de casos entre 1930 y 1960; en 1963 se obtuvo una vacuna eficiente para el sarampión y los casos comenzaron a bajar hasta ser casi nulos para 1980. El suavizado rojo varios ciclos anuales en el número de casos mientras el suavizado negro captura la tendencia de largo plazo.

Programa:

- ♢ Implementar *desde cero* el algoritmo $k-$vecinos más próximos convencional con la medida de distancia euclideana y *city-block*

*****

1.3.3. Función de predicción: clasificador bayesiano multinomial

La función de predicción bayesiana (inocente, *naive*) es un algoritmo conceptual y computacionalmente simple; se dice que es *inocente* debido a que se presupone que las características son independientes una de la otra. En general su rendimiento no es el mejor cuando se trata de variables predictoras cuantitativas; se comporta aceptablemente bien cuando se tienen variables categóricas y es bastante bueno cuando se tienen variables categóricas con gran cantidad de categorías.

1.3.3.1. Introducción

Considerando problemas de predicción en los que las variables predictoras son categóricas; por ejemplo, los clientes que realizan sus compras en una tienda departamental se pueden clasificar en uno de múltiples grupos según sus hábitos de compra; los datos de entrenamiento pueden consistir de una enorme lista de productos adquiridos, cada uno puede ser identificado por una categoría, tales como perecederos, electrónica, *hardware*, etc. Fácilmente el número de grupos puede exceder los cientos, incluso miles; esto imposibilita el uso del algoritmo de $k-$vecinos; existen varios algoritmos alternativos para atacar este tipo de problemas, pero la *función de predicción multinomial bayesiana* (*inocente*) sobresale por la simplicidad de su algoritmo. Antes de los ejemplos de uso, se muestra el desarrollo de su fundamento matemático.

1.3.3.2. La función de predicción multinomial bayesiana

El problema se establece de la siguiente manera: la clase a la que pertenece la observación $\mathbf{z}_0 = (y_0, \mathbf{x}_0)$, es y_0 y $\mathbf{x}_0$ es su vector predictor; una función predictora $f(\cdot|D)$ se construye a partir de D, el conjunto de observaciones de entrenamiento. La predicción de y_0 es $\widehat{y}_0 = f(\mathbf{x}_0|D)$; la diferencia es que $\mathbf{x}_0$ es un vector de contadores: cada elemento de $\mathbf{x}_0$ almacena el número de veces en que un tipo particular (categoría) o nivel de una variable cualitativa fue observada.

Sea $x_{0,j}$ el número de veces en que un tipo t_j aparece en el *documento* F_0; por ejemplo, si un cliente compró tres productos en el departamento de electrónica, en su nota se puede contabilizar esa cantidad (la nota es el *documento* del que se extrae el dato). Entonces, $\mathbf{x}_0 = [x_{0,1}, \ldots, x_{0,n}]^T$ contiene las frecuencias de los tipos para el documento F_0 y existen n tipos diferentes dentro del conjunto de tipos T.

El *propietario* (clase) de F_0 se denota como y_0 y pertenece al conjunto de propietarios: $y_0 \in \{P_1, \ldots, P_m\}$; la probabilidad de que el tipo t_j se encuentre en F_0 está dada por:

$$\pi_{k,j} = \Pr(t_j \in F_0 | y_0 = P_k)$$

La probabilidad $\pi_{k,j}$ es específica para cada clase, pero no varía entre documentos del mismo propietario. Como t_j pertenece necesariamente a T, entonces $\sum_{j=1}^{n} \pi_{k,j} = 1$. La probabilidad de ocurrencia del tipo t_j puede variar entre diferentes propietarios; si las diferencias son substanciales, entonces se puede utilizar una función de predicción para discriminar entre los propietarios dado un vector de frecuencias $\mathbf{x}_0$.

Por ejemplo, si sólo existen dos tipos, dos propietarios y sus apariciones entre documentos son independientes, entonces la probabilidad de observar un vector particular, digamos $\mathbf{x}_0 = [27, 35]^T$, se puede calcular usando la distribución binomial:

$$\Pr(\mathbf{x}_0 | y_0 = P_k) = \frac{(27+35)!}{27! \times 35!} \pi_{k,1}^{27} \pi_{k,2}^{35}$$

Recordando que la variable aleatoria binomial calcula la probabilidad de observar x *éxitos* entre n *intentos*; la probabilidad de x éxitos y $n-x$ fallas se obtiene como:

$$\Pr(x) = \frac{n!}{x!\,(n-x)!}\pi^x\,(1-\pi)^{n-x},$$

para $x \in \{0, 1, \ldots, n\}$; la expresión anterior toma en cuenta el hecho que $\pi_2 = 1 - \pi_1$.

Los valores de cada $\pi_{k,i}$ son estimaciones basadas en conjuntos de datos previos y se sustituyen sus valores al momento del cálculo.

Con esto, la definición de la función de predicción es:

$$\widehat{y}_0 = f(\mathbf{x}_0|D) = \arg\max\{\Pr(\mathbf{x}_0|y_0 = P_1), \ldots, \Pr(\mathbf{x}_0|y_0 = P_m)\}$$

En general, el número de categorías es sustancialmente mayor que dos y el cálculo de la probabilidad requiere una extensión al de la distribución binomial. Al igual que en la distribución binomial, la versión multinomial supone que el número de ocurrencias de cierto tipo son eventos independientes, entonces la probabilidad de observar al vector de frecuencias $\mathbf{x}_0$ está dada por la función de probabilidad multinomial:

$$\Pr(\mathbf{x}_0|y_0 = P_k) = \frac{(\sum_{i=1}^n x_{0,i})!}{\prod_{i=1}^n x_{0,i}!}\pi_{k,1}^{x_{0,1}} \times \ldots \times \pi_{k,1}^{x_{0,n}}$$

El término que incluye factoriales se conoce como *coeficiente multinomial*, tal como el coeficiente binomial $n!/x!(n-x)!$. Al ser computacionalmente costoso, se busca evitar su cálculo y puede realizarse para determinar la función de predicción.

Tarea:

♢ Con $\mathbf{x}_0 = [27, 35]^T$, suponiendo que existen tres propietarios y que $\pi_{1,1} = 0.5$, $\pi_{1,2} = 0.3$, $\pi_{2,1} = 0.3$, $\pi_{2,2} = 0.4$, $\pi_{3,1} = 0.4$ y $\pi_{3,2} = 0.5$:

- Calcula $\Pr(\mathbf{x}_0|y_0 = P_k)$ para $k = 1, 2, 3$ y determina la predicción:

$$\widehat{y}_0 = \arg\max\{\Pr(\mathbf{x}_0|y_0 = P_1), \Pr(\mathbf{x}_0|y_0 = P_2), \Pr(\mathbf{x}_0|y_0 = P_3)\}$$

*

1.3.3.2.1. Probabilidad posterior La función predictora de Bayes puede mejorarse al tomar en cuenta información previa. Por ejemplo, al tratar de asignar un documento a algún propietario se puede tomar en cuenta cierto conocimiento histórico sobre documentos ya clasificados para obtener el valor de las *probabilidades previas*. Incluso en ausencia de más evidencias (p.e. el vector de frecuencias), al buscar determinar al propietario de algún documento, se debería elegir a aquel que tenga la mayor cantidad hasta el momento.

La fórmula de Bayes conjunta la información contenida en el vector de frecuencias y la información previa para determinar la probabilidad posterior de la propiedad de algún documento:

$$\Pr(y_0 = P_k|\mathbf{x}_0) = \frac{\Pr(\mathbf{x}_0|y_0 = P_k) \times \Pr(y_0 = P_k)}{\Pr(\mathbf{x}_0)}$$

La predicción de la propiedad será aquella con la mayor probabilidad subsecuente; la predicción se obtiene como:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \left\{\Pr\left(y_0 = P_k|\mathbf{x}_0\right)\right\} \\ &= \arg\max_k \left\{\frac{\Pr\left(\mathbf{x}_0|y_0 = P_k\right) \times \Pr\left(y_0 = P_k\right)}{\Pr\left(\mathbf{x}_0\right)}\right\}\end{aligned}$$

El denominador $\Pr(\mathbf{x}_0)$ escala las probabilidades subsecuentes por el mismo factor y no afecta el orden de las mismas; por tanto puede simplificarse a una expresión más práctica:

$$\widehat{y}_0 = \arg\max_k \left\{\Pr\left(\mathbf{x}_0|y_0 = P_k\right) \times \Pr\left(y_0 = P_k\right)\right\}$$

Como el coeficiente multinomial depende solamente del vector $\mathbf{x}_0$ y tiene el mismo valor para cualquier P_k, puede ser ignorado en el cálculo de $\Pr(\mathbf{x}_0|y_0 = P_k)$, dado que sólo nos interesa determinar cual de las probabilidades posteriores es mayor.

Las probabilidades multinomiales ($\pi_{k,j}$) son desconocidas, pero pueden estimarse a partir de la información previa, evitando el cálculo del coeficiente multinomial; así, la función de Bayes en términos de las probabilidades estimadas es:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \left\{\widehat{\Pr}\left(y_0 = P_k|\mathbf{x}_0\right)\right\} \\ &= \arg\max_k \left\{\widehat{\pi}_{k,1}^{x_{0,1}} \times \ldots \times \widehat{\pi}_{k,n}^{x_{0,n}} \times \widehat{\pi}_k\right\}\end{aligned},$$

donde $x_{0,j}$ es la frecuencia de ocurrencia del tipo t_j en el documento F_0; la probabilidad estimada $\widehat{\pi}_{k,j}$ es la frecuencia relativa de la ocurrencia de t_j entre todos los documentos pertenecientes a P_k; finalmente, $\widehat{\pi}_k = \widehat{\Pr}(y_0 = P_k)$ es la probabilidad previa estimada de que el documento pertenezca a P_k.

Una vez que se observa al vector $\mathbf{x}_0$, se utiliza la información que contiene para actualizar las probabilidades posteriores. Si no se cuenta con información previa, se deben utilizar probabilidades previas *no informativas*: $\pi_1 = \ldots = \pi_g = 1/g$, suponiendo que hay g propietarios.

Para utilizar la fórmula de Bayes, se necesita estimaciones de las probabilidades $\pi_{k,j}$, $j = 1, \ldots, n$ y $k = 1, \ldots, g$; para obtenerlas se usan las proporciones muestrales o probabilidades empíricas:

$$\widehat{\pi}_{k,j} = \frac{x_{k,j}}{n_k},$$

donde $x_{k,j}$ es el número de ocurrencias de t_j en los documentos pertenecientes a P_k y $n_k = \sum_j x_{k,j}$ es el total de todas la frecuencias de tipos para el propietario P_k. Se puede presentar un subdesbordamiento (*underflow*) si los exponentes de $\widehat{\pi}_{k,j}^{x_{0,j}}$ son grandes y las bases son pequeñas; para evitarlo se busca al propietario con la mayor probabilidad *logarítmica-posterior*:

$$\arg\max_k \left\{\widehat{\Pr}\left(y_0 = P_k|\mathbf{x}_0\right)\right\} = \arg\max_k \left\{\log\left[\widehat{\Pr}\left(y_0 = P_k|\mathbf{x}_0\right)\right]\right\}$$

Con esto, la versión final de la función de predicción bayesiana multinomial es:

$$\begin{aligned}\widehat{y}_0 &= \arg\max_k \left\{\log\left[\widehat{\Pr}\left(y_0 = P_k|\mathbf{x}_0\right)\right]\right\} \\ &= \arg\max_k \left\{\log\left(\widehat{\pi}_k\right) + \textstyle\sum_{j=1}^{n} x_{0,j} \log\left(\widehat{\pi}_{k,j}\right)\right\}\end{aligned}$$

Tarea: _____

♢ Para el mismo ejemplo, suponiendo las probabilidades previas $\pi_1 = 0.8$, $\pi_2 = \pi_3 = 0.1$, determina las probabilidades posteriores $\Pr(y_0 = P_k|\mathbf{x}_0)$ para $k = 1, 2, 3$ y determina la predicción $\widehat{y}_0$.

_____*

1.3.3.3. Ejemplo

Construcción de un detector de *spam* en textos cortos (SMS) en inglés:

Se han realizado estudios y se ha determinado que los mensajes de *spam* contienen palabras como *free, win, winner, cash* y *prize*; además suelen incluir letras mayúsculas y signos de admiración. Este es un problema de clasificación supervisada binaria, dado que los mensajes serán identificados como "*spam*" o "*no spam*" y los datos de entrenamiento están etiquetados, se encuentran en https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection.

Como el conjunto de datos es grande (5572 filas) y el modelo sólo acepta valores numéricos como entrada, es necesario hacer un preprocesamiento; utilizaremos el concepto de *Bag of Words* (*BoW* bolsa de palabras). BoW es un término usado para especificar problemas que tienen datos de texto en los que se requiere obtener la frecuencia de cada palabra en el texto tratando cada palabra como un elemento independiente sin importar el orden.

Para esto puede usarse la clase *CountVectorizer* de *sklearn*, esta clase realiza lo siguiente:

♢ Crea *tokens* individuales para cada palabra asignando un ID entero para cada *token*

♢ Cuenta las ocurrencias de cada *token*

♢ Convierte todas las palabras en minúsculas para no tratar la misma palabra como dos distintas si están escritas con alguna letra mayúscula

♢ Ignora los signos de puntuación para no tratar como distintas aquellas palabras que incluyan alguno de estos símbolos

♢ Un parámetro importante es *stop_words* que sirve para no tomar en cuenta las palabras más comúnes del idioma especificado; p.e. si se indica *english* se ignoraran entradas como "a*m*", "*and*", "*the*", etc.

Como ejemplo sencillo, se consideran cuatro documentos y se aplica *CountVectorizer*:

```
# textos a usar
documents = ['Hello, how are you!',
             'Win money, win from home.',
             'Call me now.',
             'Hello, Call hello you tomorrow?']
# objeto CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()
# información del objeto
print(count_vector)
# ajuste del modelo
count_vector.fit(documents)
names = count_vector.get_feature_names()
names
```

Ahora se crea una matriz en la que las filas corresponden con cada documento y las columnas con cada palabra; cada celda es la frecuencia de la palabra en ese documento, para esto se utiliza *transform*:

```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

```
array([[1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1],
       [0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 2, 0],
       [0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [0, 1, 0, 2, 0, 0, 0, 0, 0, 1, 0, 1]])
```

Para facilitar el entendimiento de los datos, se convierte el *array* en un *dataframe* con los nombres de las columnas:

```
import pandas as pd
frequency_matrix = pd.DataFrame(data=doc_array, columns=names)
frequency_matrix
```

Una vez entendido el uso de *BoW*, es posible continuar con el detector de *spam* en SMS.

```
# Importar pandas
import pandas as pd
df = pd.read_csv('https://bit.ly/2kCy3CN',
                 sep='\t',
                 names=['label','sms_message'])
# primeros 5 renglones
# La primera columna determina el tipo de mensaje: 'spam' o 'ham'
df.head()
```

Preprocesamiento

Como *sci-kit learn* sólo manipula valores numéricos, se convierten las etiquetas a valores binarios: $0 => "ham"; 1 => "spam"$

```
# Conversion
df.label = df.label.map({'ham':0, 'spam':1})
df.head()
```

Dividiendo los datos para entrenamiento y pruebas

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df.sms_message, df.label,
                                                    random_state=1)
print('Conjunto completo: {}'.format(df.shape[0]))
print('Conjunto de entrenamiento: {}'.format(X_train.shape[0]))
print('Conjunto de pruebas: {}'.format(X_test.shape[0]))
```

Aplicar BoW a los datos

- ♢ Primero se ajusta el objeto de tipo *CountVectorizer* con los datos de entrenamiento y devuelve una matriz
- ♢ Después se transforman los datos del conjunto de pruebas

```
# objeto CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()
# ajuste de datos de entrenamiento
training_data = count_vector.fit_transform(X_train)
# transformación de los datos de prueba
testing_data = count_vector.transform(X_test)
```

Clasificador bayesiano multinomial de Sci-Kit Learn

Se utiliza la clase *MultinomialNB* de *sklearn*; este modelo es adecuado para clasificación con características discretas, como el caso del *spam*:

```
from sklearn.naive_bayes import MultinomialNB
naive_bayes = MultinomialNB()
naive_bayes.fit(training_data, y_train)
```

Una vez entrenado el modelo, se pueden realizar las predicciones para contrastarlas con el conjunto de pruebas:

```
predictions = naive_bayes.predict(testing_data)
```

Evaluación del modelo

Existen varios mecanismos para realizarlo:

♢ *Accuracy* (exactitud): Determina la frecuencia con la que el modelo realiza una predicción correcta; matemáticamente es la razón entre el número de predicciones correctas entre el número de total de predicciones

$$accuracy = \frac{true\,positives + true\,negatives}{true\,positives + true\,negatives + false\,positives + false\,negatives}$$

♢ *Precision* (precisión): Es la proporción de mensajes clasificados como *spam* (*true positives*) y que realmente lo son (*all positives*)

$$precision = \frac{true\,positives}{true\,positives + false\,positives}$$

♢ *Recall* (sensitividad): Es la proporción de mensajes que son realmente *spam* y que fueron clasificados de esa forma

$$recall = \frac{true\,positives}{true\,positives + false\,negatives}$$

♢ *F1*: Combina las medidas de precisión (*precision*) y sensitividad (*recall*)

$$F1 = 2\frac{PRE \times REC}{PRE + REC}$$

Estas formas de medir modelos están incluídas dentro de *sklearn*:

```
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_sco
print(' Accuracy: ', format(accuracy_score(y_test, predictions)))
print('Precision: ', format(precision_score(y_test, predictions)))
print('   Recall: ', format(recall_score(y_test, predictions)))
print('       F1: ', format(f1_score(y_test, predictions)))
```

1.3.3.4. Ventajas y desventajas

Naive Bayes es un algoritmo de clasificación muy usado para procesamiento de lenguaje natural; por esto es importante tener en cuenta sus limitaciones y desventajas.

1.3.3.4.1. Ventajas

♢ **Sencillez y eficiancia**: es un algoritmo simple y computacionalmente eficiente, lo que lo hace sencillo de entender e implementar; además, requiere pocos datos de entrenamiento para estimar los parámetros necesarios para la clasificación.

♢ **Entrenamiento y predicción rápidos**: debido a su sencillez requiere poco tiempo de entrenamiento comparado con otros algoritmos más complejos; esto lo hace apropiado para aplicaciones de tiempo real.

♢ **Trabaja bien con datos categóricos**: este algoritmo supone que las características son categóricas o discretas; se puede utilizar con datos numéricas se se *discretizan* en intervalos.

♢ **Útil con datos de alta dimensionalidad**: se comporta bien cuando el número de caractarísticas es grande comparado con el número de muestras de entrenamiento; por esto es útil para la clasificación de textos y tareas de detección de *spam*.

♢ **Robusto ante características irrelevantes**: este algoritmo ignora las dependencias entre características, esto implica que aún si algunas columnas no son informativas o son redundantes, no afectarán significativamente la exactitud de la clasificación.

1.3.3.4.2. Desventajas

♢ **Fuerte dependencia de la suposición**: Si mayor debilidad es la fuerte suposición de independencia entre las características.

♢ **Nula interacción entre características**: El algoritmo no puede capturar relaciones entre las características; supone que el efecto de una característica en particular en la clasificación es independiente de la presencia o ausencia de otras características.

- ♢ **Poder de representación limitado**: Este algoritmo se comporta bien cuando las clases son fácilmente separables, pero puede presentar dificultados con límites de decisión complejas; por tal motivo su poder de representación es limitado en comparación con otros algoritmos, como SVM o redes neuronales.
- ♢ **Dependencia de la calidad de los datos**: es un algoritmo que depende en gran medida de la calidad de los datos de entrenamiento; si estos datos con ruidosos o incompletos, puede afectar negativamente en la exactitud del clasificador.

Programa: _____

- ♢ Implementar *desde cero* el algoritmo del clasificador Bayesiano multinomial

_____ *****

1.3.4. Árboles de decisión/regresión

Los árboles de decisión son un tipo importante de modelos predictivos de aprendizaje artificial (*machine learning*). Existen varios algoritmos para este tipo de árboles desde hace varias décadas y algunas variantes como *random forest* se consideran como técnicas muy poderosas en el área de minería de datos (*data mining*).

El término *Classification and Regression Trees* (CART) fue introducido por Leo Breiman (https://bit.ly/2RvIgwl) en 1984 para referirse a árboles de decisión y regresión que pueden usarse como modelos de clasificación o regresión. El algoritmo CART es el fundamento de otros importantes algoritmos tales como *bagged decision trees*, *random forest* y *boosted decision trees* (https://bit.ly/2NtIUXZ). La idea básica es dividir repetidamente los registros disponibles buscando maximizar la *homogeneidad* en los conjuntos obtenidos en dicha división.

Es un modelo jerárquico de aprendizaje que puede ser usado tanto para clasificación como para regresión: permiten explorar relaciones complejas entre entradas y salidas sin necesidad de hacer suposiciones sobre los datos. Los árboles de decisión pueden verse como una función f que realiza una estimación de un *mapeo* desde un espacio de entrada X hacia un espacio objetivo y: $y = f(X)$. Para el caso de la clasificación, el espacio objetivo y es numérico: $y \in \mathbb{R}$; para la regresión, los valores de y son categóricos: $y \in \{C_1, C_2, \ldots\}$. Se trata de un modelo de aprendizaje supervisado; por tanto, se tienen muestras con la salida esperada. La representación más común para CART es un árbol binario.

Un árbol de decisión divide el espacio X en regiones locales utilizando alguna medida de distancia y el objetivo es determinar particiones bien separadas y homogéneas. Las regiones se dividen de acuerdo a preguntas de prueba y, con esto, se obtiene una división $n-$aria, de forma que mientras se recorre el árbol, en cada nodo se realiza toman decisiones.

Ejemplo 1

La figura 1.3 se muestra un ejemplo sencillo. Los datos se encuentran en dos dimensiones $\{x_1, x_2\}$, también llamadas características, variables ó atributos. Se trata de una clasificación binaria con clases *Redonda* y *Cuadrada*. En el nodo raíz se pregunta si $x_1 > w_{10}$ que divide los datos en un nodo hijo (derecho) con instancias de la clase R y un nodo hijo (izquierdo) que tiene instancias tanto de R como de C. En el hijo izquierdo se realiza una segunda pregunta $x_2 > w_{20}$ para obtener finalmente instancias separadas de las clases R y C. Es importante notar que los resultados obtenidos realizan particiones disjuntas en las variables de entrada.

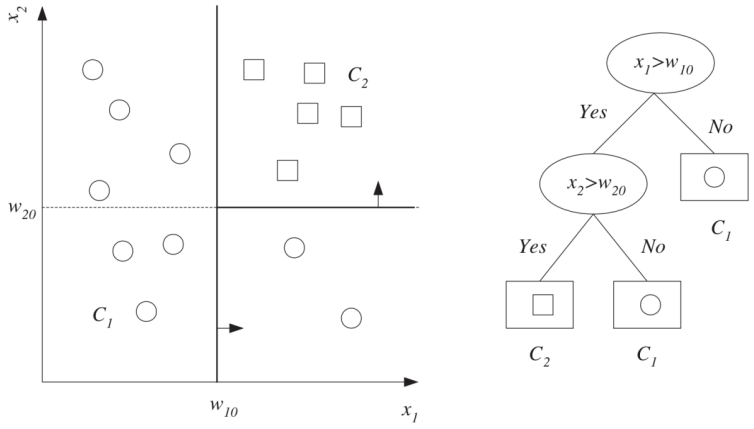


Figure 1.3: Ejemplo de árbol de decisión/regresión (derecha) y sus datos (izquierda)

Ejemplo 2

La figura 1.4 muestra otro ejemplo sencillo para determinar si se debe cargar paraguas o no, de acuerdo a la predicción del estado del tiempo.

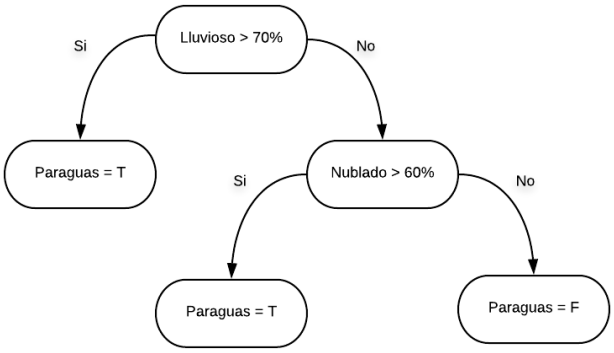


Figure 1.4: Ejemplo de árbol de decisión para llevar paraguas

El algoritmo 1.1 muestra la forma de obtener los valores para las particiones.

Algorithm 1.1 Obtención de particiones

elegir una variable x_i

- ♢ elegir algún valor s_i para x_i que divida los datos de entrenamiento en dos particiones (no necesariamente iguales)
- ♢ medir la *pureza* (homogeneidad) resultante en cada partición
- ♢ usar otro valor s_j para x_i buscando incrementar la pureza de las particiones hasta alcanzar el umbral de *pureza aceptable*

repetir el proceso de partición con una variable diferente (posiblemente alguna usada previamente)
cada valor obtenido se convierte en un nodo en el árbol

Para determinar la homogeneidad de las particiones, se tienen dos posibilidades:

1.3.4.1. Grado de impureza de Gini

El índice de impureza en un rectángulo A que contiene m clases, se calcula como:

$$I(A) = 1 - \sum_{i=1}^{m} p_i^2$$

Con p la proporción de casos en el rectángulo A que pertenecen a la clase i. Es importante notar que:

- ♢ $I(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.
- ♢ El valor máximo sucede cuando todas las clases se encuentran igualmente representadas (0.5 para el caso binario).

1.3.4.2. Grado de entropía

El grado de entropía en un área A que contiene m clases, se calcula como:

$$E(A) = \sum_{i=1}^{m} p_i \times log_2(p_i)$$

Con p la proporción de casos en A que pertenecen a la clase i. Es importante notar:

- ♢ $E(A) = 0$ cuando todos los casos pertenecen a la clase i; es decir, es totalmente homogénea.
- ♢ El valor máximo $log_2(m)$ sucede cuando todas las clases se encuentran igualmente representadas.

Problemas con CART

- ♢ Pueden ser poco robustos: un cambio pequeño en los datos de entrenamiento generan grandes cambios en la estructura del árbol.
- ♢ Obtener el mejor árbol de decisión es $NP - completo$; por tanto, en la práctica para su construcción se utilizan heurísticas basadas en óptimos locales.

♢ No es tan difícil obtener modelos sobreajustados que no generalizan bien las muestras; por esto es necesario establacer un límite en la obtención de particiones.

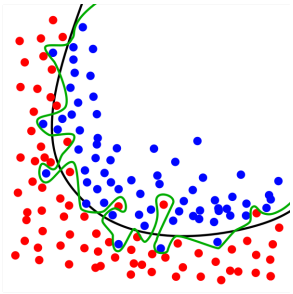


Figure 1.5: Ejemplo de sobreajuste

Ejemplo 3

Para el caso de variables categóricas el proceso es más simple. En la figura 1.6 se observa un ejemplo de datos categóricos para determinar si se debe jugar tennis o no, de acuerdo a la predicción del estado del tiempo.

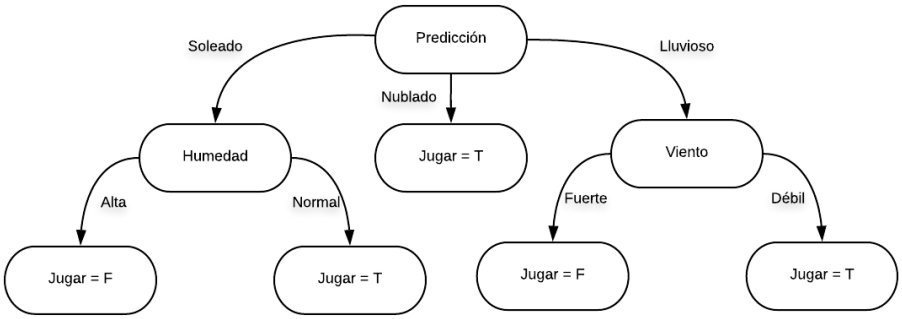


Figure 1.6: Ejemplo de árbol de decisión con variables categóricas

Extracción de reglas

Una vez constuidos, los árboles de decisión son altamente interpretables y es fácil entender la información que representan como reglas del tipo *if - then*: cada camino desde un nodo hasta una hoja es una conjunción de condiciones que deben ser satisfechas para llegar a ese nodo terminal. Para el último ejemplo, tenemos:

♢ **if** P=S **and** H=N **then** J=T

♢ **if** P=N **then** J=T

♢ **if** P=Ll **and** V=D **then** J=T

1.3.5. Función de predicción: *eXtreme Gradient Boosting* (*XGBoost*)

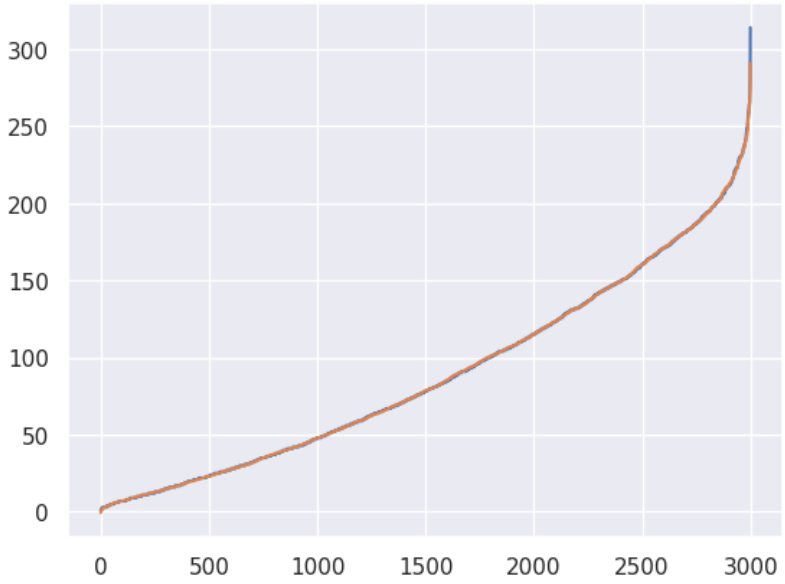
XGBoost es una biblioteca optimizada y distribuída de mejora del gradiente diseñada para ser altamente eficiente, flexible y portable. implementa algoritmos de aprendizaje automático bajo la idea del mejora del gradiente (*Gradient Boosting*). *XGBoost* provee mejora en la construcción de árboles paralelizada.

XGBoost es una de las implementaciones más populares y eficientes del algoritmo *Gradient Boosted Trees*, un método supervisado de aprendizaje basado en aproximación de funciones al optimizar las funciones de pérdida mientras aplica varias técnicas de regularización.

La propuesta original se encuentra en: https://arxiv.org/pdf/1603.02754.pdf.

Error medio absoluto:

```
mae = metrics.mean_absolute_error(y_test, y_hat)
print("Mean Absolute Error = ", mae)
```



Tarea:

♢ Dentro de los conjuntos de datos incluidos en SciKitLearn existe uno con dígitos escritos a mano

```
from sklearn.datasets import load_digits
digits = load_digits()
```

♢ Aplica los modelos *k-vecinos*, *bayesiano multinomial* y *XGBoost* y compara los resultados con las cuatro métricas presentadas

*

1.3.6. Análisis de asociaciones

Muchos negocios acumulan grandes cantidades de datos en sus actividades diarias, por ejemplo, en tiendas de autoservicio recolectan datos de compras diariamente en sus recibos de compra.

El **análisis de asociaciones** intenta extraer **reglas de asociación** de interés, ocultas en esta gran cantidad de datos.

TID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Figura 1.7: Ejemplo de transacciones de una tienda

Por ejemplo, de la tabla observada en la figura 1.7 se podría obtener la regla:

$$\{Diapers\} \longrightarrow \{Beer\}$$

Que sugiere una relación entre pañales y cerveza. Las tiendas pueden aprovechar este tipo de reglas para vender otros productos a sus clientes.

Figura 1.8: Recomendaciones (https://twitter.com/lisachwinter/status/494063906625961984)

1.3.6.1. Representación binaria

Para realizar el análisis de asociaciones, la tabla de la figura 1.7 se puede presentar en formato binario: cada fila representa una nota, y cada columna es un producto. En cada renglón, un 1

indica presencia del producto y un 0 denota su ausencia, como la mostrada en la figura 1.9. Es importante notar que esta tabla es una vista simplificada del los productos vendidos en una tienda, dado que no toma en cuenta aspectos como la cantidad o el precio de cada producto vendido.

TID	Bread	Milk	Diapers	Beer	Eggs	Cola
1	1	1	0	0	0	0
2	1	0	1	1	1	0
3	0	1	1	1	0	1
4	1	1	1	1	0	0
5	1	1	1	0	0	1

Figura 1.9: Tabla binaria de las transacciones en la figura 1.7

1.3.6.2. Conjunto de artículos (*itemset*) y conteo de soporte (*support count*)

Sea $I = \{i_1, i_2, \ldots, i_d\}$ el conjunto de todos los productos ofrecidos por una tienda y $T = \{t_1, t_2, \ldots, t_N\}$ el conjunto de todas las transacciones. Cada transacción t_i contiene un subconjunto de artículos elegidos de I; en el análisis de asociaciones, una colección de cero o más productos se conoce como *itemset* (*conjunto de artículos*).

Una transacción t_i contiene al *itemset* X si Xes un subconjunto de t_i, por ejemplo la segunda nota de la figura 1.7 contiene al conjunto $\{Bread, Diapers\}$ pero no a $\{Bread, Milk\}$. Una propiedad muy importante de un *itemset*, es su *conteo de soporte* (*support count*) que se refiere al número de transacciones que contienen un conjunto particular de productos; matemáticamente el conteo de soporte se establece como:

$$\sigma(X) = |\{t_i | X \subseteq t_i, t_i \in T\}|$$

donde el símbolo $|\cdot|$denota la cardinalidad de un conjunto. Para las notas de la figura 1.7, el conteo de soporte para $\{Beer, Diapers, Milk\}$ es dos porque solamente hay dos transacciones que contienen a los tres productos.

Un *itemset* se nombrará *frecuente* si $s(X)$ es mayor que un umbral definido por el usuario (hiperparámetro) llamado *min-sup*.

1.3.6.3. Reglas de asociación

Una regla de asociación es un implicación de la forma $X \longrightarrow Y$, en la que X y Y son *itemsets* disjuntos; la fortaleza de una regla de asociación se puede medir in términos de su *soporte* (*support*) y su *confianza* (*confidence*): el soporte determina con qué frecuencia una regla es aplicable a un determinado conjunto de datos, la confianza indica que tan frecuentemente los artículos de Y aparecen en las notas que contienen a X. La definición formal de estas métricas son:

$$\text{Soporte: } s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$\text{Confianza: } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Por ejemplo, para la regla $\{Milk, Diapers\} \longrightarrow \{Beer\}$, con ayuda de la tabla binaria de la figura 1.9 obtenemos:

♢ *Support*: filas en las que se encuentran $\{Milk, Diapers, Beer\} = 2$ y el total de filas $= 5$. Por lo tanto, esta medida tiene valor de $\frac{2}{5} = 0.4$.

♢ *Confidence*: filas que contienen $\{Milk, Diapers, Beer\} = 2$ y filas que incluyen $\{Milk, Diapers\} = 3$. Por lo tanto, esta medida tiene valor de $\frac{2}{3} = 0.67$.

El número total de reglas que pueden extraerse de un conjunto de transacciones es exponencial, con respecto al número de productos. Por este motivo, la tarea suele dividirse en:

1. Generación del conjunto de *productos frecuentes* (*Frequent Itemset*): encontrar los conjuntos que satisfacen *minsup*.
2. Generación de reglas: extraer las reglas con alta confianza, a partir de los conjuntos generados en el paso previo.

El problema se reduce a: dado un conjunto de transacciones, encontrar todas las reglas que tengan $support \geq minsup$ y $confidence \geq minconf$, donde, *minsup* y *minconf* son umbrales esperados (hiperparámetros).

1.3.6.4. Algoritmo *Apriori*

Aún con los valores mínimos de soporte y confianza, la generación de conjunto de productos frecuentes es una tarea computacionalmente muy compleja. Resulta necesario buscar alternativas eficientes, entre ellas:

♢ **Reducir el número de comparaciones**. En lugar de comparar cada conjunto con cada transacción, es posible reducir su número, usando alguna estructura de datos avanzada (gráfica).

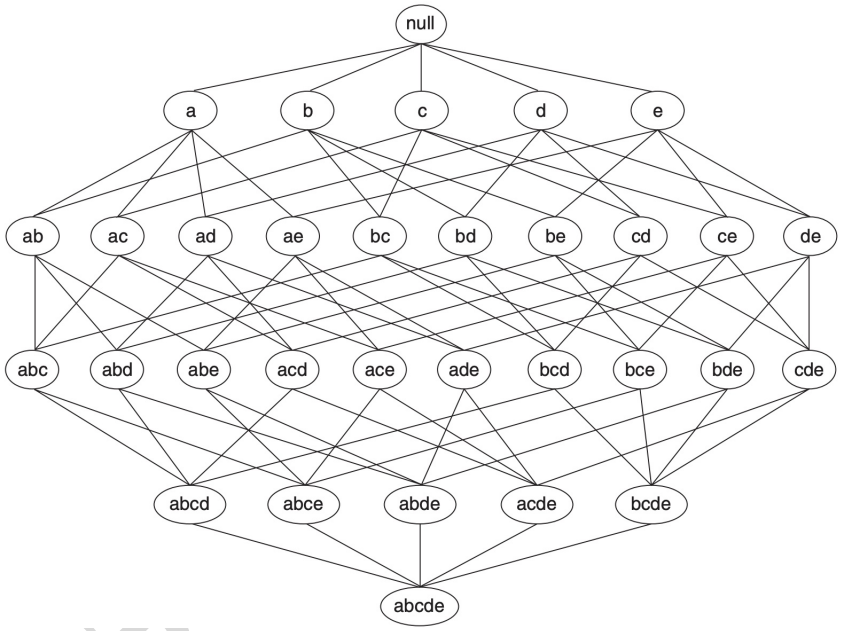


Figura 1.10: Gráfica compleeta generada para cinco artículos

♢ **Reducir el número de *itemsets* candidatos.** El ***principio Apriori*** es una forma efectiva de eliminar algunos conjuntos candidatos, sin la necesidad de contar sus elementos.

Entonces la reducción en la generación del conjunto de productos frecuentes se realiza de la siguiente manera:

1. Principio *Apriori*: si un *itemset* es frecuente, entonces todos sus subconjuntos también deben ser frecuentes.

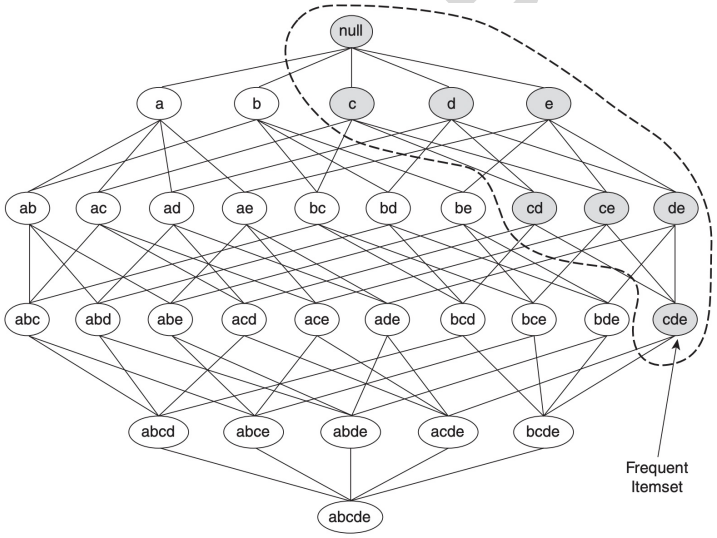


Figura 1.11: Ilustración del principio *Apriori*

2. *Poda basada en soporte*: si un *itemset* es infrecuente, entonces todos sus superconjuntos también deben ser infrecuentes.

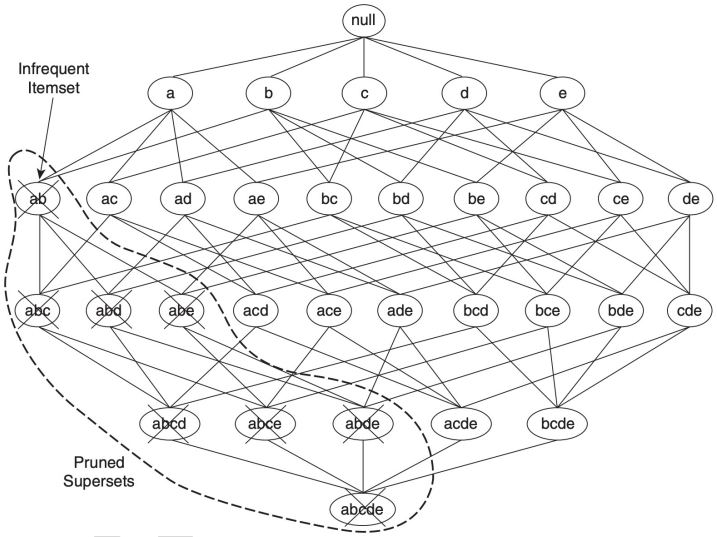


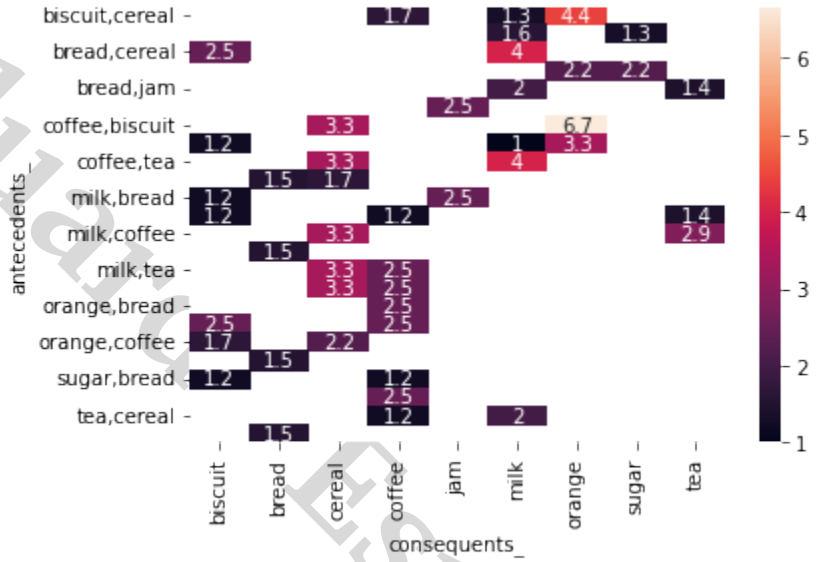
Figura 1.12: Ilustración de poda basada en soporte

Con estas consideraciones, el algoritmo para la generación de conjuntos frecuentes es el siguiente:

Algoritmo 1.2 Generación de conjuntos frecuentes con el principio *Apriori*

1. $k = 1$
2. $F_k = \{1 - itemsets \quad frecuentes\}$
3. Repetir
4. $\quad k = k + 1$
5. Generar *itemsets* candidatos: C_k a partir de F_{k-1}
6. Podar *itemsets* candidatos: C_k que no sean frecuentes en F_{k-1}
7. Conteo de soporte: contar el número de ocurrencias de cada candidato a partir de las filas del conjunto completo
8. Eliminación de candidatos: eliminar los candidatos de C_k que sean infrecuentes, dejando solamente a los frecuentes.
9. Hasta que $F_k = \emptyset$

```
# Generar un heatmap con anotaciones
import seaborn as sns
import matplotlib.pyplot as plt
sns.heatmap(pivot, annot = True)
plt.yticks(rotation=0)
plt.xticks(rotation=90)
plt.show()
```



Introduction to Data Mining (Second Edition).

Chapter 5. Association Analysis: Basic Concepts and Algorithms.

Tarea:

♢ Utilizando el conjunto de datos *Online Retail*:

`https://archive.ics.uci.edu/dataset/352/online+retail`

♢ Obtener un conjunto de artículos frecuentes así como reglas de asociación que se puedan extraer del mismo