FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES

Descarte de ideas no factibles

Idea no factible	Justificación
1.b. Técnica CRISP-DM (Cross Industry Standard Process for Data Mining).	Debido a la naturaleza del proyecto, esta metodología para el desarrollo de proyectos excede los alcances del mismo. Exigiendo un mayor esfuerzo para obtener resultados que se pueden alcanzar con otra metodología más simple. Es por esto, que se descarta la idea de utilizar la técnica CRISP-DM.
2.a. Modelo de Identificación.	No va acorde a los objetivos que busca lograr la solución, ya que sólo evidenciar la existencia de objetos, eventos o actividades en un conjunto de datos no es suficiente para generar una solución al problema planteado.
3.a Uso de grafos para ver qué producto está asociado con cuál en el entorno de ventas.	Esta idea de presentación se descarta debido a su complejidad al momento de implementar la visualización gráfica de un grafo. Además, no es una alternativa que sea muy clara para representar varias asociaciones.

Después de realizar el descarte de ideas no factibles, se tienen las siguientes alternativas:

1. Metodologías para el desarrollo de proyectos en Minería de Datos:

    a. Técnica Proceso de Generación de Conocimiento o KDD (*Knowledge Discovery in DataBases*).

2. Modelos (y algoritmos respectivos) para la Minería de Datos:
    a. Agrupación
        i. Algoritmo de Clústeres K-Means.
        ii. Algoritmo de Clústeres K-Medoids.
        iii. Autómatas finitos.

    b. Asociación
        i. Algoritmo de Fuerza Bruta.
        ii. Algoritmo Apriori
        iii. Algoritmo Partition

    c. Predicción
        i. Algoritmo de Bayes naive.
        ii. Algoritmo de Regresión logística.

3. Presentación (gráfica) de resultados:

    a. Uso de histogramas para mostrar la frecuencia de los itemsets.
    b. Uso de tablas para mostrar las asociaciones encontradas.

Diseños preliminares

Técnica Proceso de Generación de Conocimiento o KDD (*Knowledge Discovery in DataBases*).

Este proceso surte cuatro pasos para la generación de conocimiento. Estas etapas pueden ser recursivas, es decir, que se retorna a ellas una y otra vez (proceso iterativo) a medida que se obtienen resultados preliminares que requieren replantear las variables iniciales.

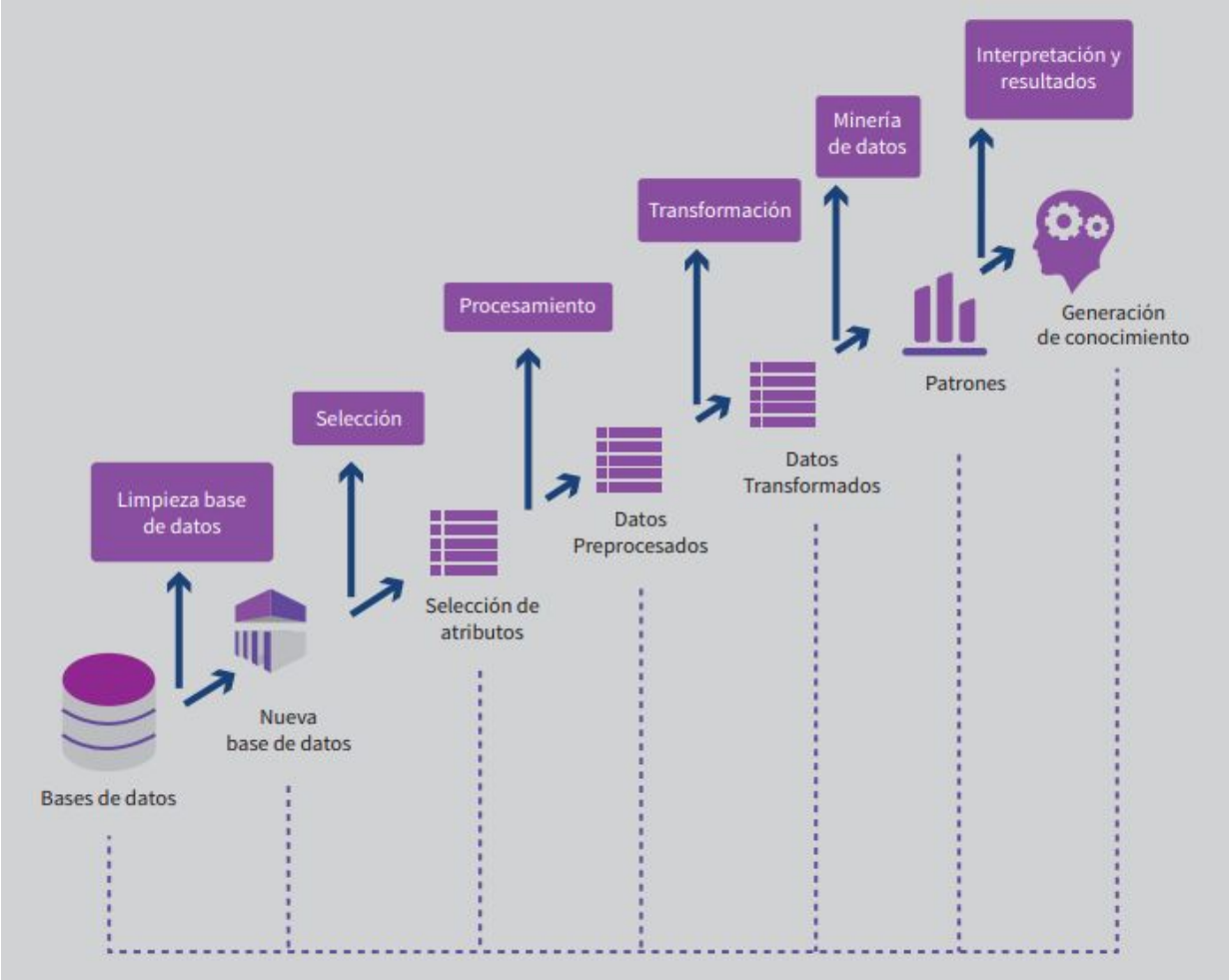


Figura . Proceso de KDD

Las etapas del proceso son:

1. Selección de los datos
2. Pre procesamiento de datos
3. Selección de características
4. Minería de Datos
5. Interpretación y Resultados

Etapas	Descripción

Selección de los datos		Consiste en la recolección y preparación de los datos. En este proceso se comprende la problemática asociada a la base de datos y se establecen objetivos. Y se identifican las variables que serán consideradas para la construcción del modelo de minería de datos
Pre-procesamiento de datos	Integración de datos	Se analiza si la base de datos requiere incluir o integrar información o variables que reposan en otras bases de datos, y que será relevante para el modelo de minería de datos
	Reconocimiento y limpieza	Se depura el conjunto de datos respecto a valores atípicos, faltantes y erróneos (eliminación de ruido e inconsistencias).
Selección de características	Exploración y limpieza de datos	Aplicando técnicas de análisis exploratorio de datos se busca identificar la distribución de los datos, simetría, pruebas de normalidad y correlaciones existentes entre los datos. En esta etapa es útil el análisis descriptivo del conjunto de datos (clustering y segmentación, escalamiento, reglas de asociación y dependencia, reducción de la dimensión), identificación de datos nulos, ruido y outliers, así como el uso de matrices de correlación (si las variables son numéricas), diagramas (barras, histogramas, caja y bigotes), entre otras técnicas adecuadas de muestreo
	Transformación	Se estandariza o normaliza la información (colocarla en los mismos términos de formato y forma).
	Reducción de datos	Se disminuye el tamaño de los datos mediante la eliminación de características redundantes.
Minería de Datos		Se puede definir como un proceso no trivial de identificación válida, novedosa, potencialmente útil y entendible de patrones comprensibles que se encuentran ocultos en los datos, que a su vez, facilita la toma de decisiones y emplea técnicas de aprendizaje supervisado y no-supervisado.
Interpretación y Resultados		Se analizan los resultados de los patrones obtenidos en la fase de MD, mediante técnicas de visualización y de representación, con el fin de generar conocimiento que aporte mayor valor

	a los datos. En esta fase se evalúan los resultados con los expertos y, si es necesario, se retorna a las fases anteriores para una nueva iteración

Tabla . Proceso KDD

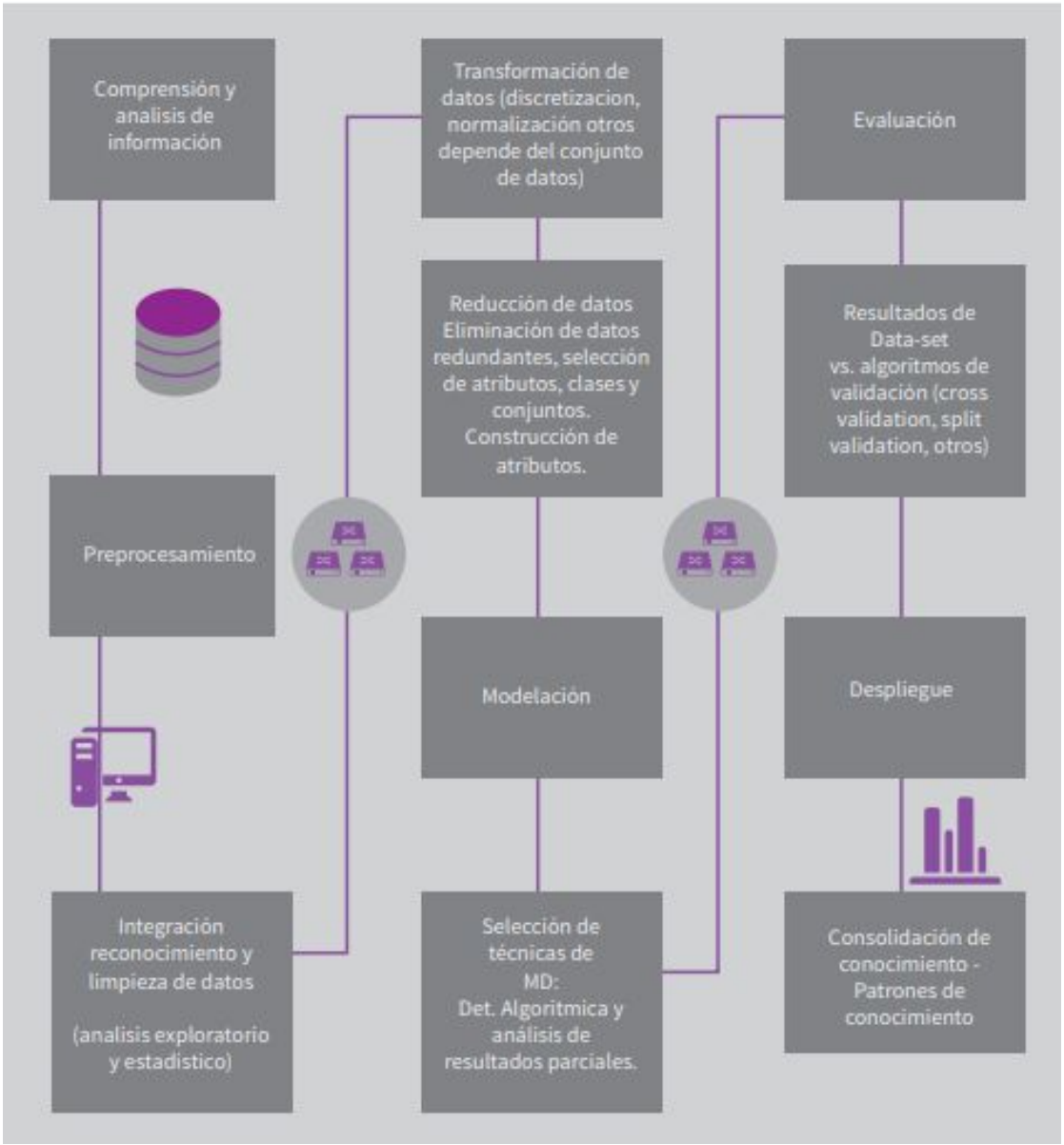


Imagen. Proceso KDD

Técnica de Agrupación utilizando el Algoritmo de Clústeres K-Means.	Técnica de Agrupación utilizando el Algoritmo de Clústeres K-Medoids.
El análisis de conglomerados o Clustering, es una técnica que permite analizar y examinar datos que no se encuentran etiquetados, formando conjuntos de grupos a partir de su similitud.	
Pseudocódigo	

Algorithm 1: K-Means Algorithm

Input: $E = \{e_1, e_2, \ldots, e_n\}$ (set of entities to be clustered)
k (number of clusters)
$MaxIters$ (limit of iterations)
Output: $C = \{c_1, c_2, \ldots, c_k\}$ (set of cluster centroids)
$L = \{l(e) \mid e = 1, 2, \ldots, n\}$ (set of cluster labels of E)

foreach $c_i \in C$ **do**
  $c_i \leftarrow e_j \in E$ (e.g. random selection)
end
foreach $e_i \in E$ **do**
  $l(e_i) \leftarrow argminDistance(e_i, c_j) j \in \{1 \ldots k\}$
end

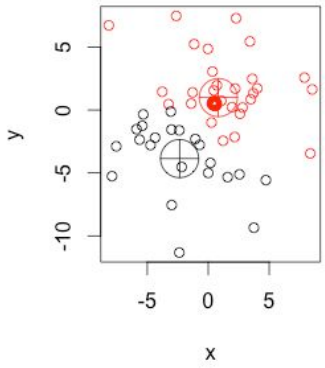
$changed \leftarrow false$;
$iter \leftarrow 0$;
repeat
  foreach $c_i \in C$ **do**
    $UpdateCluster(c_i)$;
  end
  foreach $e_i \in E$ **do**
    $minDist \leftarrow argminDistance(e_i, c_j)\ j \in \{1 \ldots k\}$;
    if $minDist \neq l(e_i)$ **then**
      $l(e_i) \leftarrow minDist$;
      $changed \leftarrow true$;
    end
  end
  $iter++$;
until $changed = true$ *and* $iter \leq MaxIters$;

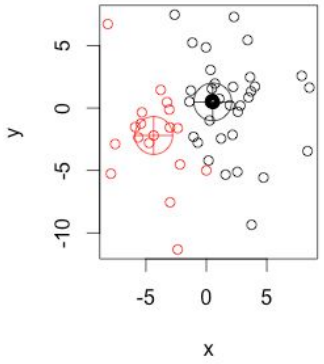
Algorithm 1 The k-Medoid Algorithm

Input: k : The total number of clusters to generate.
Input: D : The collection of input documents.
Input: T : Total number of iterations allowed.
Output: P : The array of k clusters of the partition.

1: Function k-Medoidsk, D
2: $iterCount \leftarrow 0$
3: **repeat**
4: $\{m_1, m_2, \ldots, m_k\} \leftarrow$ InitializeMedoids(k, D)
5: $P \leftarrow [\{m_1\}, \{m_2\}, \ldots, \{m_k\}]$
6: **repeat**
7: $\tilde{D} \leftarrow \{d_i : d_i \in D \wedge d_i \notin \{m_1, m_2, \ldots, m_k\}\}$
8: **for each:** $d_i \subset \tilde{D}$ **do**
9: **for** $j = 1$ *to* k **do**
10: $sim_{i,j} \leftarrow$ CosineSimilarityd_i, m_j
11: end for
12: $imax \leftarrow \text{argmax}_{1 \leq j \leq k}(sim_{i,j})$
13: $P[imax] \leftarrow P[imax] \cup \{d_i\}$
14: **end for**
15: $oldMedoids \leftarrow \{m_1, m_2, \ldots, m_k\}$
16: $[m_1, m_2, \ldots, m_k) \leftarrow$ UpdateMedoids P
17: $iterCount \leftarrow iterCount + 1$
18: **until** $(oldMedoids = \{m_1, m_2, \ldots, m_k\}) \vee (iterCount = T$
19: $NoTinyClusters \leftarrow$ CheckSizeOfClusters (C)
20: **until** $(NoTinyClusters = True) \vee (iterCount = T)$
21: **return** P
22: **end function**

Clústeres K-Means.	Clústeres K-Medoids.
Ambos algoritmos intentan minimizar la distancia entre los puntos del mismo grupo y un punto particular que es el centro de ese grupo.	





Similitudes

- La variación dentro del clúster disminuye con cada iteración del algoritmo.
- El algoritmo siempre converge, depende en los centros iniciales. Para cualquiera de los algoritmos, uno debe ejecutarlo varias veces con diferentes inicios
- La agrupación final depende de los centros iniciales del clúster. Diferentes comienzos dan como resultado diferentes agrupaciones finales.

Diferencias

- Devuelve centros que son promedios de puntos de datos.

- Generalmente devuelve un mayor valor de

$$\sum_{k=1}^{K} \sum_{C(i)=k} \left\| X_i - c_k \right\|_2^2$$

- Computacionalmente más difícil (debido al paso 2: calcular el medoid es más difícil que calcular el promedio)
- Tiene la propiedad (potencialmente importante) de que los centros se encuentran entre los puntos de datos
- Se basa en el cálculo de centroides (o medoides) minimizando la distancia absoluta entre los puntos y el centroide seleccionado, en lugar de minimizar la distancia cuadrada.

Técnica de Asociación utilizando el Algoritmo de Fuerza Bruta.

Es una técnica de búsqueda también llamada búsqueda exhaustiva que consiste en observar todos los posibles candidatos a una solución.

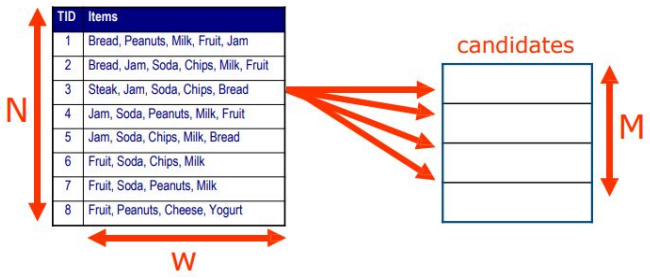
Pseudocódigo

Algorithm 1: COMBINATIONS(c, S)

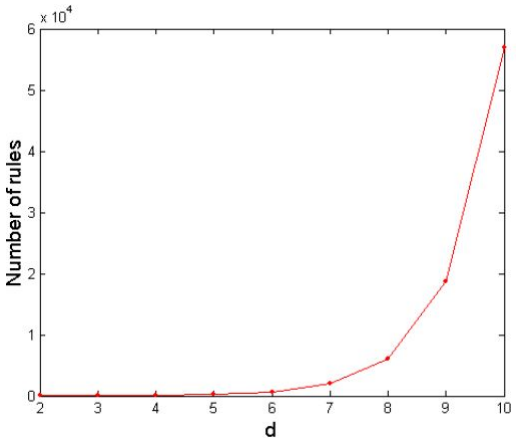
Input: c, S
Output: All subsets of size c of S

```
root ← ∅
s ← |S|
COMBINATIONSREC(c, 0, root, s)
icombinations ←EXTRACTFROMTREE(root)
combinations ←MAPPOSITIONSTOELEMENTS(icombinations)
return combinations
```

Complejidad

Complejidad temporal	O(NMw) , M = 2^n 
Complejidad computacional	Dado un número únido d de item. • Número total de itemsets: 2^d • Número total de posibles asociaciones:

$$\sum_{k=1}^{d-1} \left[\binom{d}{k} \times \sum_{j=1}^{d-k} \binom{d-k}{j} \right]$$



Estrategias para mejorar la generación de itemsets frecuentes

- Reducir el número de candidatos (M)
  - Búsqueda completa: 2^n .
  - Utilizar técnicas de poda para reducir M.
- Reducir el número transacciones (N)
  - A medida que aumenta el tamaño de los itemsets, reducir N.
- Reducir el número de comparaciones (NM)
  - Usar estructuras de datos eficientes para almacenar los candidatos o transacciones.

Técnica de Asociación utilizando el Algoritmo Apriori.

Pseudocódigo

Algorithm 6.1 Frequent itemset generation of the *Apriori* algorithm.

1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$. {Find all frequent 1-itemsets}
3: **repeat**
4: $k = k + 1$.
5: $C_k = \text{apriori-gen}(F_{k-1})$. {Generate candidate itemsets}
6: **for** each transaction $t \in T$ **do**
7: $C_t = \text{subset}(C_k, t)$. {Identify all candidates that belong to t}
8: **for** each candidate itemset $c \in C_t$ **do**
9: $\sigma(c) = \sigma(c) + 1$. {Increment support count}
10: **end for**
11: **end for**
12: $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$. {Extract the frequent k-itemsets}
13: **until** $F_k = \emptyset$
14: Result $= \bigcup F_k$.

Algorithm 6.2 Rule generation of the *Apriori* algorithm.

1: **for** each frequent k-itemset f_k, $k \geq 2$ **do**
2: $H_1 = \{i \mid i \in f_k\}$ {1-item consequents of the rule.}
3: **call** `ap-genrules`$(f_k, H_1.)$
4: **end for**

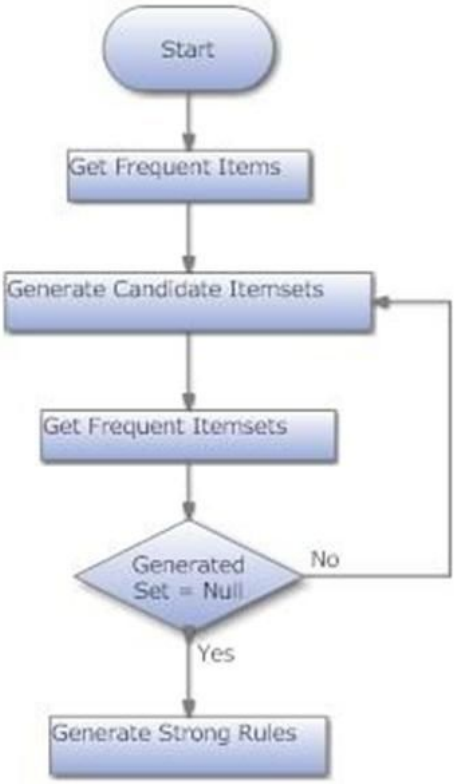
Algorithm 6.3 Procedure `ap-genrules`(f_k, H_m).

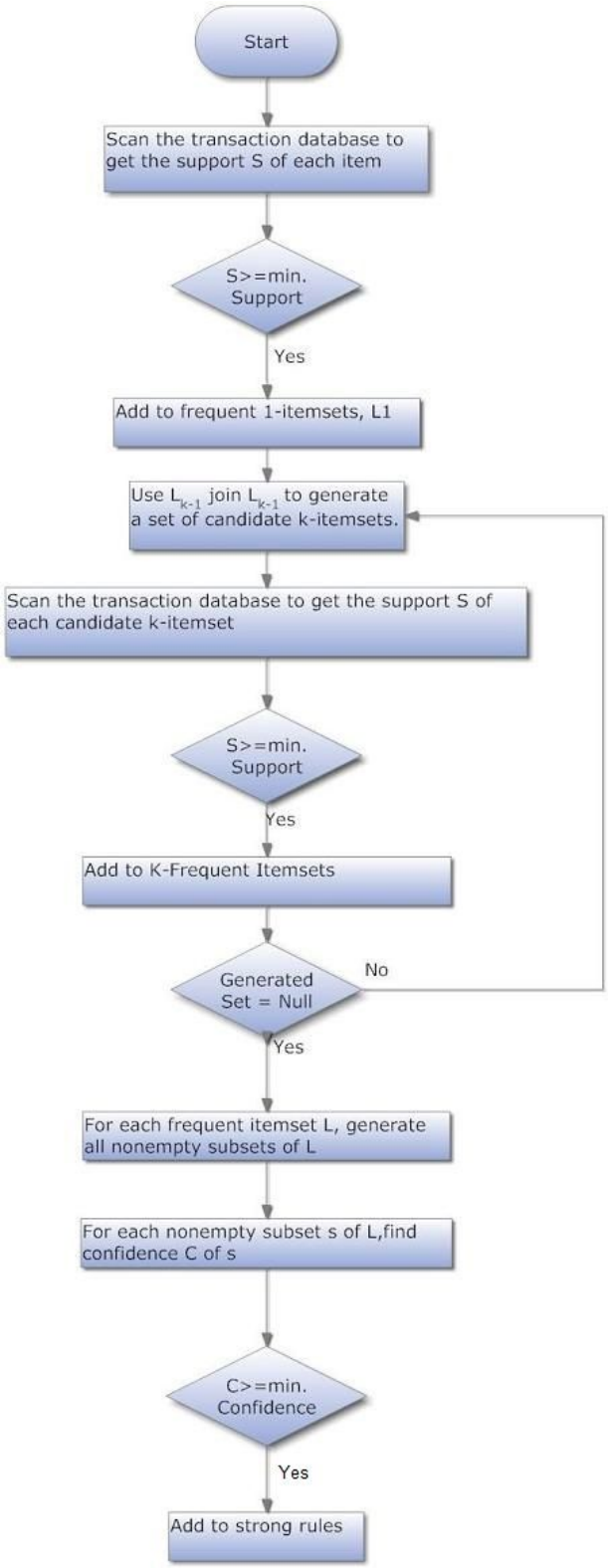
1: $k = |f_k|$ {size of frequent itemset.}
2: $m = |H_m|$ {size of rule consequent.}
3: **if** $k > m + 1$ **then**
4: $H_{m+1} = \text{apriori-gen}(H_m)$.
5: **for** each $h_{m+1} \in H_{m+1}$ **do**
6: $conf = \sigma(f_k)/\sigma(f_k - h_{m+1})$.
7: **if** $conf \geq minconf$ **then**
8: **output** the rule $(f_k - h_{m+1}) \longrightarrow h_{m+1}$.
9: **else**
10: **delete** h_{m+1} from H_{m+1}.
11: **end if**
12: **end for**
13: **call** ap-genrules$(f_k, H_{m+1}.)$
14: **end if**

Algorithm 6.4 Support counting using closed frequent itemsets.

```
1: Let C denote the set of closed frequent itemsets
2: Let k_max denote the maximum size of closed frequent itemsets
3: F_{k_max} = {f | f ∈ C, |f| = k_max}      {Find all frequent itemsets of size k_max.}
4: for k = k_max − 1 downto 1 do
5:    F_k = {f | f ⊂ F_{k+1}, |f| = k}      {Find all frequent itemsets of size k.}
6:    for each f ∈ F_k do
7:       if f ∉ C then
8:          f.support = max{f'.support | f' ∈ F_{k+1}, f ⊂ f'}
9:       end if
10:   end for
11: end for
```

Diagramas de flujo


Start
Get Frequent Items
Generate Candidate Itemsets
Get Frequent Itemsets
Generated Set = Null
No
Yes
Generate Strong Rules


Start
Scan the transaction database to get the support S of each item
S>=min. Support
Yes
Add to frequent 1-itemsets, L1
Use L_{k-1} join L_{k-1} to generate a set of candidate k-itemsets.
Scan the transaction database to get the support S of each candidate k-itemset
S>=min. Support
Yes
Add to K-Frequent Itemsets
Generated Set = Null
No
Yes
For each frequent itemset L, generate all nonempty subsets of L
For each nonempty subset s of L,find confidence C of s
C>=min. Confidence
Yes
Add to strong rules

Técnica de Asociación utilizando el Algoritmo Partition.

Pseudocódigo

```
Initial Partitions ( examples, final_attributes)
m ← Number (final_attributes)
for i ← 1 ... m do
  e[ ] ← { sorted examples of the attribute i}
  partitions[ i ] ← Average (e[ ])
  pointer ← Position (e[ ], partitions[ i ] )
  k ← pointer
  while e_k.class ≠ 1   // seeking next positive
     if OR[ i ] > 1 then
          k ← k + 1    // positive association
     else
          k ← k – 1    // negative association
     end-if
  end-while
  if pointer ≠ k then
     if OR[ i ] > 1 then
          partitions[ i ] ← ( e_k + e_k–1 ) / 2
          // positive association
     else
          partitions[ i ] ← ( e_k + e_k+1 ) / 2
          // negative association
     end-if
  end-if
end-for
```

Características de Implementación

- Este algoritmo recorre la base de datos sólo dos veces. La primera vez, cada partición es minada independientemente para encontrar todos los conjuntos de ítems frecuentes en la partición y luego se mezclan éstos para generar el conjunto de los conjuntos de ítems candidatos. Muchos de éstos pueden ser falsos positivos, pero ninguno falso negativo.
- En la segunda pasada se cuenta la ocurrencia de cada candidato, aquellos cuyo soporte es mayor que el mínimo soporte especificado se retienen como conjuntos frecuentes.
- Emplea el mecanismo de intersección entre conjuntos para determinar el soporte de dichos conjuntos, en este caso cada ítem en una partición mantiene la lista de los identificadores de las transacciones que contienen a dicho ítem.

Técnica de Predicción utilizando el Algoritmo de Bayes Naive.

Naive Bayes es una técnica de clasificación y predicción que construye modelos que predicen la probabilidad de posibles resultados. Naive Bayes utiliza datos históricos para encontrar asociaciones y relaciones y hacer predicciones.

Pseudocódigo

Algorithm 3.1: Fitting a naive Bayes classifier to binary features

$N_c = 0$, $N_{jc} = 0$;
for $i = 1 : N$ **do**
  $c = y_i$ // Class label of i'th example;
  $N_c := N_c + 1$;
  for $j = 1 : D$ **do**
    if $x_{ij} = 1$ **then**
      $N_{jc} := N_{jc} + 1$
$\hat{\pi}_c = \frac{N_c}{N}$, $\hat{\theta}_{jc} = \frac{N_{jc}}{N_c}$

Algorithm 3.2: Predicting with a naive bayes classifier for binary features

for $i = 1 : N$ **do**
  for $c = 1 : C$ **do**
    $L_{ic} = \log \hat{\pi}_c$;
    for $j = 1 : D$ **do**
      if $x_{ij} = 1$ **then** $L_{ic} := L_{ic} + \log \hat{\theta}_{jc}$ **else** $L_{ic} := L_{ic} + \log(1 - \hat{\theta}_{jc})$
  $p_{ic} = \exp(L_{ic} - \text{logsumexp}(L_{i,:}))$;
  $\hat{y}_i = \text{argmax}_c \, p_{ic}$;

Ventajas

- Muy simple, fácil de implementar y rápido.
- Si el supuesto de independencia condicional NB se mantiene, entonces convergerá más rápido que los modelos discriminatorios como la regresión logística.
- Incluso si la suposición de NB no se cumple, funciona bien en la práctica.
- Necesita menos datos de entrenamiento.
- Altamente escalable. Se escala linealmente con el número de predictores y puntos de datos.
- Se puede utilizar para problemas de clasificación binarios y de múltiples clases de vidrio.
- Puede hacer predicciones probabilísticas.
- Maneja datos continuos y discretos.
- No es sensible a las características irrelevantes.

Técnica de Predicción utilizando el Algoritmo de Regresión logística	
Ventajas	**Desventajas**
● Es una técnica muy utilizada porque es muy eficiente. ● No requiere demasiados recursos informáticos. ● Es muy fácil de interpretar. ● No requiere que las funciones de entrada se amplíen. ● No requiere ningún ajuste. ● Es fácil de regularizar. ● Produce probabilidades pronosticadas bien calibradas. ● Funciona mejor cuando elimina atributos que no están relacionados con la variable de salida, así como atributos que son muy similares (correlacionados) entre sí. ● Es muy eficiente de entrenar. ● También es una buena línea de base que puede usar para medir el rendimiento de otros Algoritmos más complejos.	● No puede resolver problemas no lineales ya que su superficie de decisión es lineal. ● Cuenta con una alta dependencia de una correcta presentación de sus datos. Esto significa que la regresión logística no es una herramienta útil a menos que ya haya identificado todas las variables independientes importantes. ● Dado que su resultado es discreto, sólo puede predecir un resultado categórico.

Técnica de Predicción utilizando el Algoritmo de Regresión logística
Un autómata finito es un modelo computacional que realiza un proceso de forma automática para producir una salida desde cierta entrada. Para implementar un análisis de datos usando este método, la idea será crear un autómata finito que compute automáticamente un proceso que siempre lleve a analizar unos datos de entrada.
Definición
Un autómata finito es una 5-tupla que se compone de la siguiente manera: (Q, Σ, q_0, δ, F), donde: - Q es el conjunto finito de estados por el que está compuesto el autómata. - Σ es el alfabeto con el que actúa (dominio). - q_0 pertenece a Q. - δ es la función que define las transiciones entre estados. - F es el conjunto que contiene a los estados del autómata en los que termina (estados de aceptación).
Tabla de Transición

Salida qϵQ	Símbolo sϵΣ	Llegada q'ϵQ
S1	0	S2
S1	1	S1
S2	0	S1
S2	1	S2

Tipos

Autómatas Finitos Deterministas	**Autómatas Finitos No Deterministas**
Es un autómata finito que además es un sistema determinista; es decir, para cada estado $q \in Q$ en que se encuentre el autómata, y con cualquier símbolo $s \in \Sigma$ del alfabeto (dominio) leído, existe siempre una transición posible de un estado a otro. Por ejemplo, el autómata a mano izquierda es un AFD.	Es un autómata finito que, a diferencia de los autómatas finitos deterministas (AFD), posee al menos un estado $q \in Q$, tal que para un símbolo $s \in \Sigma$ del alfabeto, exista más de una transición posible.

Se concluye que lo más apropiado para nuestra solución es implementar un autómata que automatice un procedimiento que analice los datos que se tienen, sería un Autómata finito determinista. En efecto, este permite tener todos los posibles resultados para cualquier tipo de dato que entre (símbolo), a diferencia del AFN.

Implementación

Se tiene un conjunto (C) de datos. El estado inicial (podría ser q1 en el ejemplo) del autómata representaría el subconjunto (S0) de datos con el que comienzo a analizar.

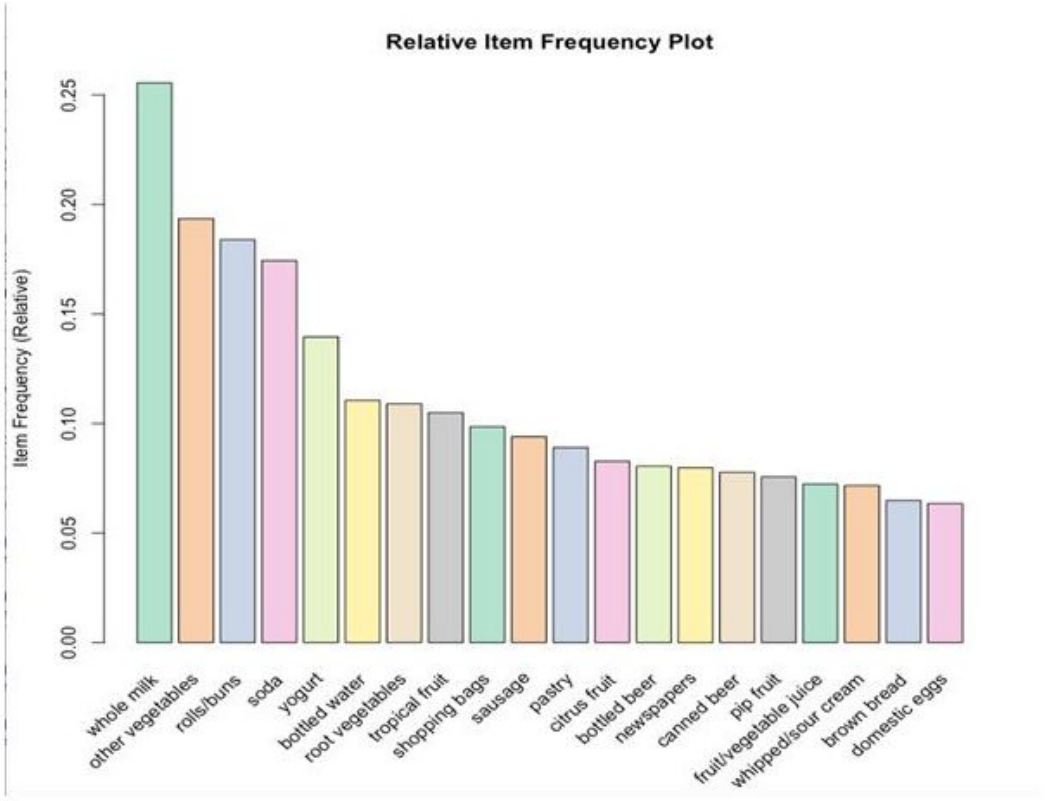
En seguida entra otro subconjunto de datos (S1) tal que S1 pertenece a C, es decir los subconjuntos son los datos que voy a ir analizando. El AFD, que por el momento se encuentra en el estado inicial, decide que procedimiento realizar dependiendo de la clasificación que tenga S2. (esta clasificación estaría representada por los símbolos posibles que tenga el autómata). En el diagrama a continuación estos serían los símbolos: 0 y 1.

Se continuaría de esta forma hasta que se llegue a un estado final. En el diagrama sería: q0

Por ejemplo. En el caso hipotético de que se tenga el primer subconjunto S0 (q1) y entre otro subconjunto de datos S1 de tipo 1, entonces el autómata procedería a realizar las operaciones de análisis que lleven directamente al estado final q0. En el cual la información que se tiene se puede considerar un análisis relevante.

Presentación de resultados mediante el uso de histogramas para mostrar la frecuencia de los itemsets.

Diseño



Se utilizan los componentes de Visual Studio para crear histogramas con la frecuencia de los itemsets de un tamaño dado.

Presentación de resultados mediante el uso de tablas para mostrar las asociaciones encontradas.

Diseño

1. **Pasillo de comidas - Leche, Huevos, Pan**
2. **Pasillo de licores - Licor, Red/Blush Vino, Cerveza, Soda**
3. **Pasillo de desayuno - Cereal Yogurt, Arroz, Avena**