```
from sklearn.model_selection import train_test_split
```

In [2]:
```
#Conjunto de datos secuencial
dataset = pd.read_csv('https://alhernandezsua.gitlab.io/amd-misti/datasets/CSDMC_API_Train.csv')
```

In [3]:
```
dataset.head()
```

Out[3]:

| | y | x |
|---|---|---|
| **0** | 1 | LoadLibraryW HeapAlloc HeapAlloc HeapFree Heap... |
| **1** | 1 | RegOpenKeyExW LoadLibraryA GetProcAddress GetP... |
| **2** | 1 | HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFre... |
| **3** | 1 | HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFre... |
| **4** | 1 | HeapAlloc HeapFree HeapAlloc HeapAlloc HeapFre... |

In [4]:
```
# Transformar la columna x en una lista
valores = dataset['x'].values
```

In [5]:
```
#Juntar todas las secuencias de cada muestra
palabras = ' '.join([palabra for palabra in valores])
#Dividir la secuencia en palabras (términos)
tokens = [p.split(' ') for p in valores]
```

In [6]:
```
#FreqDist calcula la frecuencia absoluta de cada palabra única
frecc = nltk.FreqDist(palabras.split(' '))
```

In [7]:
```
vocabulario =  []
for api32, valor in frecc.items():
    if api32 !='':
        vocabulario.append(api32)
```

In [ ]:
```
#for muestra in matriz_numeros:
#    print(cosine_distances([matriz_numeros[0]],[muestra]))
```

## Ejemplo con árboles de desición

In [13]:
```
y = dataset['y']
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(matriz_numeros,y,train_size=.8)
```

In [15]:
```
arbolito = DecisionTreeClassifier(criterion='entropy',max_depth=3)
arbolito.fit(X_train,y_train)
```

Out[15]:
DecisionTreeClassifier

DecisionTreeClassifier(criterion='entropy', max_depth=3)

In [16]:
```
arbolito.score(X_test,y_test)*100
```

Out[16]: 84.61538461538461

# Ejemplo con TF-IDF (term-frequency inverse-document-frequency)

In [17]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [18]:
```
vec = TfidfVectorizer()
X_vec = vec.fit_transform(dataset['x'])
```

In [19]:
```
X_vec.shape
```

In [26]:
```
arbolito.score(X_test,y_test)*100
```

Out[26]: 94.87179487179486

## Ejemplo de un sistema básico de recomendación

El Regresor de Bosque Aleatorio es un algoritmo de aprendizaje automático utilizado para tareas de regresión, donde el objetivo es predecir resultados continuos. Es una técnica de aprendizaje en conjunto que opera mediante la construcción de una multitud de árboles de decisión durante el entrenamiento y la emisión de la predicción media de los árboles individuales.

In [83]:
```
# Ejemplo de datos
descripciones = [
    "El software X antes de la versión 2.0.4 tiene una vulnerabilidad de ejecución de código remo
    "La aplicación Y en la versión 1.1.1 permite escalada de privilegios a través de la red local
    "Vulnerabilidad de denegación de servicio en el dispositivo Z cuando se manejan paquetes malf
]
rankings = [8.5, 7.0, 9.0]  # Supongamos que estos son los rankings de las vulnerabilidades

# Convertir textos a un formato numérico usando TF-IDF
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(descripciones)

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, rankings, test_size=0.2, random_state=42)

# Entrenar un modelo de regresión
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Función para predecir el ranking de una nueva descripción
def predecir_ranking(nueva_descripcion):
    nueva_descripcion_transformada = vectorizer.transform([nueva_descripcion])
    ranking_predicho = model.predict(nueva_descripcion_transformada)
```

```
# Extraer las descripciones
descriptions = dataset_cve['summary']
```

In [77]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Crear el objeto TF-IDF
vectorizer = TfidfVectorizer()

# Ajustar y transformar los datos
tfidf_matrix = vectorizer.fit_transform(descriptions)
```

In [78]:
```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Seleccionar una muestra para comparar, por ejemplo, la primera CVE
sample_index = 2 #comparando la tercera CVE
sample_vector = tfidf_matrix[sample_index]

# Calcular la similitud del coseno entre la muestra y todas las descripciones
cosine_similarities = cosine_similarity(sample_vector, tfidf_matrix).flatten()
```

In [79]:
```
# Crear un DataFrame para mostrar las similitudes
similarities_df = pd.DataFrame({
    'name': dataset_cve['name'],
    'summary': dataset_cve['summary'],
    'cosine_similarity': cosine_similarities
})

# Ordenar por la similitud del coseno en orden descendente
similarities_df = similarities_df.sort_values(by='cosine_similarity', ascending=False)

# Mostrar las 5 CVEs más similares a la muestra seleccionada
print(similarities_df.head(5))  # Incluye la muestra misma
```

```
print(f"Ranking predicho para la nueva vulnerabilidad: {predicted_ranking}")
```

```
Ranking predicho para la nueva vulnerabilidad: 8.02
```