UNIVERSIDAD TECNOLÓGICA DE QUERÉTARO

Nombre del trabajo:

2a. Evaluación SA

Nombre de la materia:

Extracción de conocimiento

INGENIERÍA EN GESTIÓN Y DESARROLLO DE SOFTWARE

Presenta:

Santiago Garcia Abel Alejandro

Matricula:

2021171016

Profesor

Filiberto Ruiz Hernández

Santiago de Querétaro, Qro. 30 de noviembre 2023

SA

https://github.com/alexjamesmx/edc_2

Análisis Supervisado.

Nota: Gráfica personalizada e interpretación de resultados se encuentran en el código de cada ejercicio.

1. Basado en el conjunto de datos "beisbol.csv".

Justificación del algoritmo:
Utilicé dos algoritmos de regresión, buscando un resultado de predicción numérica continua.

La regresión lineal simple es un modelo sencillo que puede funcionar bien con conjuntos de datos pequeños, especialmente si la relación entre la variable independiente y dependiente parece ser aproximadamente lineal. Es fácil de interpretar y menos propenso al sobreajuste.

Aunque Random Forest generalmente se considera bueno para conjuntos de datos grandes, también puede funcionar bien para conjuntos de datos más pequeños. Random Forest tiene la ventaja de manejar relaciones no lineales y trabajar bien sin mucha configuración.

Descripción de los modelos
Regresión lineal

```
df= pd.read_csv(filepath_or_buffer = "./beisbol.csv", sep=',', low_memory=False)
X = df['bateos'].values.reshape(-1,1)
y = df['runs'] .values.reshape(-1,1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# modelo regresion lineal simple
model = LinearRegression()

# Entrena el modelo
model.fit(X_train, y_train)
```

Random forest

```
# Cargar datos
df_rf= pd.read_csv(filepath_or_buffer="./beisbol.csv", sep=',', low_memory=False)

# Preparar datos
X = df_rf['bateos'].values.reshape(-1, 1)
y = df_rf['runs'].values.reshape(-1, 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

model_rf = RandomForestRegressor(random_state=42)
```

```
model_rf.fit(X_train, y_train.flatten())
y_pred_rf = model_rf.predict(X_test)
```

Básicamente ambos modelos la preparación de datos es la misma, al existir pocos datos en el dataset, preferí usar 20% de datos de testing solamente, para que el modelo tenga mas chance de aprender patrones.

Evaluación del modelo
En ambos modelos se evalúa el modelo, se imprimen los resultados y después buscamos la forma de optimizar los modelos. en ambos casos. Al ser tan pocos datos, escalar no mejora el resultado, entonces vi conveniente probar con otro modelo, el cual dio resultados un poco mejores.

2. Basado en el conjunto de datos "diabetes_indiana.csv".

Justificación del algoritmo

La regresión logística es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación binaria. produce resultados que son fácilmente interpretables. El resultado del modelo se interpreta como la probabilidad de que una instancia pertenezca a una clase particular. La regresión logística puede incorporar términos de regularización (L1 o L2) para prevenir el sobreajuste y mejorar la generalización del modelo, especialmente cuando se trata con conjuntos de datos de alta dimensionalidad.

Es eficiente en conjuntos de datos relativamente grandes y no requiere grandes cantidades de recursos computacionales. También, hay una gran cantidad de recursos, implementaciones y documentación disponibles.

Descripción del diseño del modelo

```
# preparación de los datos
df = pd.read_csv(filepath_or_buffer = "./diabetes_indiana.csv", sep=',', low_memory=False)
df.describe()

print(X.shape, y.shape)
# separación de los datos en train y test, 80% y 20% respectivamente
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Implementación de escalador, por uso general y estandarizado para mejorar la predicción de los datos.

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled  = scaler.transform(X_train)
X_test_scaled  = scaler.transform(X_test)

#usar modelo
modelo = LogisticRegression(random_state=42, class_weight='balanced')
modelo.fit(X_train_scaled, y_train)
```

Este fue el modelo inicial, más tarde en la optimización se realizó la afinación de hiper parámetros para un modelo de regresión logística mediante una búsqueda aleatoria.

Evaluación y optimización del modelo

El modelo fue probado con datos sin escalar y escalados, los resultados no varían mucho, el modelo optimizado alcanza un f-score de 0.61 a 0.85 los cuales son buenos resultados.
El modelo fue optimizado mediante el proceso de búsqueda de los mejores hiper parámetros para un modelo de regresión logística mediante una búsqueda aleatoria, mejorando así el rendimiento del modelo en comparación con los valores predeterminados de los hiper parámetros.

3. Basado en el conjunto de datos "samsung.csv"

Justificación del algoritmo.
K-Means es un algoritmo simple y fácil de entender. La eficiencia se debe a su naturaleza iterativa y algoritmo de optimización basado en el descenso del gradiente. Los K means son fáciles de interpretar, ya que los clústeres formados son representados por los centroides, y cada punto de datos pertenece al clúster cuyo centroide está más cerca.

Éste algoritmo es escalable y puede manejar grandes conjuntos de datos con eficiencia.

Descripción del diseño del modelo.

```
df = pd.read_csv("./samsung.csv")
df.head()
df.describe()
# graficar los datos
sb.pairplot(df.dropna(), height=4, vars=["Close", "Volume"], kind='scatter')

# Normalizar
df_norm = df.copy()
df_norm.head()
scaler = MinMaxScaler()
scaler.fit(df_norm[['Close']])
df_norm['Close'] = scaler.transform(df_norm[['Close']])
scaler.fit(df_norm[['Volume']])
df_norm['Volume'] = scaler.transform(df_norm[['Volume']])
df_norm.head()

# graficar los datos normalizados
sb.pairplot(df_norm.dropna(), height=4, vars=["Close", "Volume"], kind='scatter')

# Seleccionar las características relevantes para clustering
X = df_norm[['Close', 'Volume']]

# Rango de número de clústeres (1 a 10, por ejemplo)
num_clusters = range(1, 11)

# Lista para almacenar las sumas de distancias cuadradas intra-cluster
inertia = []

# Iterar sobre diferentes número de clústeres
for k in num_clusters:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    # Obtener la suma de distancias cuadradas intra-cluster
    inertia.append(kmeans.inertia_)
# Graficar el codo para determinar el número óptimo de clústeres
```

```
plt.figure(figsize=(10, 6))
plt.plot(num_clusters, inertia, marker='o')
plt.title('Método del Codo para Determinar el Número Óptimo de Clústeres')
plt.xlabel('Número de Clústeres')
plt.ylabel('Suma de Distancias Cuadradas Intra-Cluster')
plt.show()
```

Los datos no los comprendemos con verlos, entonces primero los graficamos para tratar de entenderlos, al ver que son datos con volúmenes grandes y enormes tamaños de los mismos, es necesario realizar una normalización de los datos para que se sitúen alrededor del 0 y su interpretación sea más sencilla.

Una vez normalizados podemos generar la gráfica de codos para saber masomenos que tanta cantidad de centroides necesitamos. Cuando la línea sea recta o ya no haya gran diferencia entre cada punto del gráfico, se puede concluir que hasta ese punto se pueden agrupar los centroides. es decir, la cantidad de centroides que son recomendables.

Con el número de centroides establecido, es momento de usar el modelo y visualizar los conjuntos de datos.

4. Basado en el conjunto de datos "comprar_alquilar.csv"

Justificación del algoritmo
En muchos conjuntos de datos, especialmente aquellos con muchas variables, hay redundancias y colinealidades entre las variables. PCA busca proyectar los datos en un nuevo conjunto de dimensiones (llamadas componentes principales) que conservan la mayor cantidad de información posible. Al reducir la dimensionalidad, se simplifica el conjunto de datos, se mejora la eficiencia computacional y se facilita la interpretación.

Los componentes principales pueden representar patrones latentes o características subyacentes en los datos.

Descripción del diseño del modelo

```
#leer datos
df = pd.read_csv("./comprar_alquilar.csv")
df.describe()
y = df['comprar']
x = df.drop(['comprar'], axis=1)

x.head()


# Normalizar los datos
X_scaled  = StandardScaler().fit_transform(x)
X_scaled  = pd.DataFrame(X_scaled )
X_scaled .describe()

# Aplicar PCA para reducir la dimensionalidad
# primero calculamos con la misma cantidad de variables
pca = PCA(n_components= 9)
#nombre de las columnas
nombres = df.columns
# Entrenar
pca.fit(X_scaled )
```

Leemos los datos y vemos que cuentan con 10 características o variables las cuales tienen relación con la realización de la compra de un alquiler. Debemos descubrir las características que más influyen en esta decisión.