MODELO ANALITICO ELECTRO DUNAS - Experimentos

Elaborado por:

Gustavo Cordero Rueda

Nelson Eduardo Melo Salamanca

Andrés Arias García

Materia: Proyecto aplicado en analítica de datos

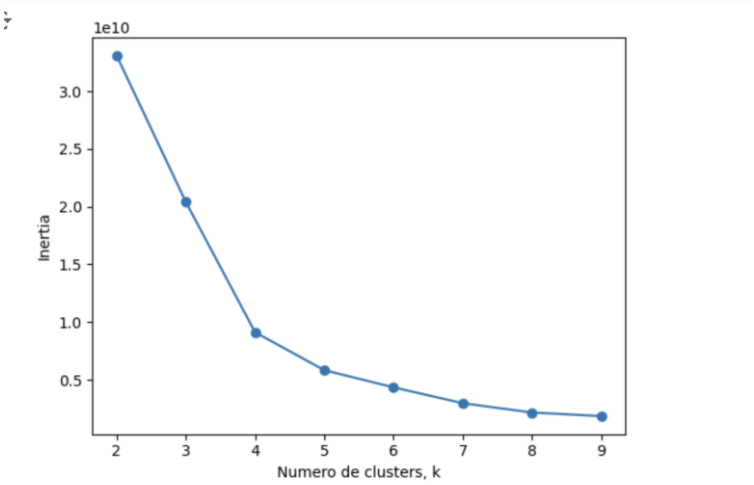
Universidad de los Andes

20 de mayo 2024

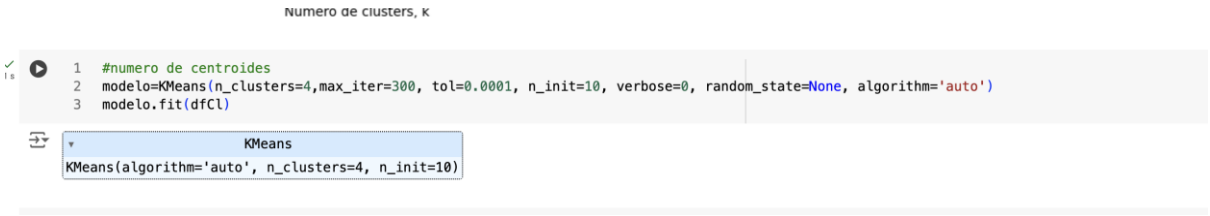
1. DESARROLLO

1.2. Clusterizacion.

Para el proceso de clusterizacion con K-means, realizamos pruebas con la cantidad de cluster que deberia tener el modelo. Esto lo hicimos con el metodo del codo donde usamos 3 valores diferentes para identificar cual de los tres nos generaba mejor resultado.



Prueba 1: Cluster 4



```
[66]  1  #medir el desempeño del modelo del modelo k-means con
      2  #indice de compactabilidad (davies and boudin index)
      3  #entre mas cercano a 1 mejor, el valor oscila entre -1 y 1.
      4  dv=metrics.davies_bouldin_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      5  silueta=metrics.silhouette_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      6  silueta=metrics.silhouette_score(dfCl, modelo.predict(dfCl))
      7  print(f'indice de Davies and Bouldin={dv}')
      8  print(f'indice de silueta={silueta}')
      9
```

```
indice de Davies and Bouldin=0.3290682316412272
indice de silueta=0.7654291831733724
```

Prueba 2: Cluster 5

```
[90]  1  #numero de centroides
      2  modelo=KMeans(n_clusters=5,max_iter=300, tol=0.0001, n_init=10, verbose=0, random_state=None, algorithm='auto')
      3  modelo.fit(dfCl)
```

```
KMeans(algorithm='auto', n_clusters=5, n_init=10)
```

```
      1  #medir el desempeño del modelo del modelo k-means con
      2  #indice de compactabilidad (davies and boudin index)
      3  #entre mas cercano a 1 mejor, el valor oscila entre -1 y 1.
      4  dv=metrics.davies_bouldin_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      5  silueta=metrics.silhouette_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      6  silueta=metrics.silhouette_score(dfCl, modelo.predict(dfCl))
      7  print(f'indice de Davies and Bouldin={dv}')
      8  print(f'indice de silueta={silueta}')
      9
```

```
indice de Davies and Bouldin=0.46275673701746767
indice de silueta=0.5604084527372052
```

Prueba 3: Cluster 6

```
      1  #numero de centroides
      2  modelo=KMeans(n_clusters=6,max_iter=300, tol=0.0001, n_init=10, verbose=0, random_state=None, algorithm='auto')
      3  modelo.fit(dfCl)
```

```
KMeans(algorithm='auto', n_clusters=6, n_init=10)
```

```
      1  #medir el desempeño del modelo del modelo k-means con
      2  #indice de compactabilidad (davies and boudin index)
      3  #entre mas cercano a 1 mejor, el valor oscila entre -1 y 1.
      4  dv=metrics.davies_bouldin_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      5  silueta=metrics.silhouette_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
      6  silueta=metrics.silhouette_score(dfCl, modelo.predict(dfCl))
      7  print(f'indice de Davies and Bouldin={dv}')
      8  print(f'indice de silueta={silueta}')
      9
```

```
indice de Davies and Bouldin=0.5303007739723454
indice de silueta=0.5697536355524436
```

Conclusion: Encontramos que con 4 cluster, obtuvimos el mejor resultado en el ejercicio de clusterizacion. En el ejercicio de 4 cluster también realizamos algunos

ajustes adicionales de los parámetros como el de la cantidad de interacciones que lo dejamos en 100. El resultado fue igual que con 300. Finalmente nos quedamos con esta configuracion.

```
#medir el desempeño del modelo del modelo k-means con
#indice de compactabilidad (davies and boudin index)
#entre mas cercano a 1 mejor, el valor oscila entre -1 y 1.
dv=metrics.davies_bouldin_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
silueta=metrics.silhouette_score(dfCl.drop('cluster',axis=1),dfCl.cluster)
silueta=metrics.silhouette_score(dfCl, modelo.predict(dfCl))
print(f'indice de Davies and Bouldin={dv}')
print(f'indice de silueta={silueta}')

indice de Davies and Bouldin=0.3290682316412272
indice de silueta=0.7654291831733724
```

Durante el ejercicio, tambien utilizamos la opción de analisis de DBSCAN y en la cual los resultados que obtuvimos no mejoraron los del KMEANS.

```
[165] #discretizamos la variable Sector ECONOMICO
      dfDs.Sector=dfDs.Sector.replace({'Elaboración de cacao y chocolate y de productos de confitería':1,'Cultivo de Árboles Frutales y Nueces':2,'Cu

[173] # inicializar el conjunto de datos con el que trabajaremos
      training_data, _ = make_classification(n_samples=300, random_state=None)

      # definir el modelo
      dbscan_model = DBSCAN(eps=0.25, min_samples=9)

[175] # entrenar el modelo
      dbscan_model.fit(training_data)

      DBSCAN
      DBSCAN(eps=0.25, min_samples=9)
```

1.3. Isoletion Forest.

Para esto modelo, el cual se baso mas en el proceso de deteccion de anomalizas realizamos algunos ajustes en sus parametros para conocer su resultado. Es importante resaltar que teniendo en cuenta los datos, nuestro objetivo principal no fue la prediccion de consumos anómalos. En el parametro indicado realizamos las siguientes pruebas:

Contaminacion= 0.04

```
y= dfIf.Anomalo#variables objetivo
```

```
[73]
# Split the data into training and testing sets
xtrain, xTest, ytrain, yTest = train_test_split(x,y,test_size=0.3,random_state=1)
model = IsolationForest(n_estimators=10, max_samples='auto', contamination=float(.04), max_features=1.0, bootstrap=False, n_jobs=-1, random_state=42,v
model.fit(xtrain)
```

```
IsolationForest(contamination=0.04, n_estimators=10, n_jobs=-1, random_state=42)
```

```
[126]
print("Matriz de confusión: \n", confusion_matrix(yTest, ypred))
```

```
Matriz de confusión:
 [[     0      0      0]
 [     0      0 129604]
 [  5270      0   4154]]
```

```
# Evaluate the model
print(classification_report(yTest, ypred))
```

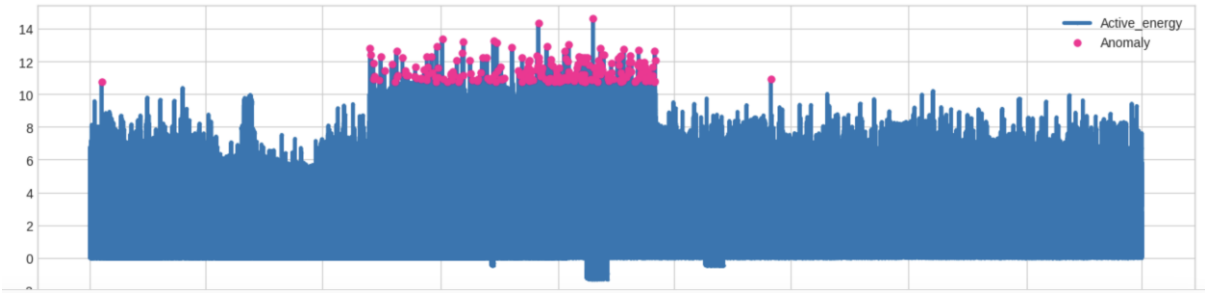
```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.00      0.00      0.00    129604
           1       0.03      0.44      0.06      9424

    accuracy                           0.03    139028
   macro avg       0.01      0.15      0.02    139028
weighted avg       0.00      0.03      0.00    139028
```

```
#IQR Methodology using the ADTK Library
iqr_ad = InterQuartileRangeAD(c=5.0)
anomalies = iqr_ad.fit_detect(dfIf['Active_energy'])
plot(dfIf['Active_energy'], anomaly=anomalies, ts_linewidth=3, ts_markersize=3, anomaly_markersize=5, anomaly_color='deeppink', anomaly_tag="marker
```

```
[<Axes: >]
```



Contaminacion= 0.03

```
# Split the data into training and testing sets
xtrain, xTest, ytrain, yTest = train_test_split(x,y,test_size=0.3,random_state=1)
model = IsolationForest(n_estimators=10, max_samples='auto', contamination=float(.03), max_features=1.0, bootstrap=False, n_jobs=-1, random_state=42,v
model.fit(xtrain)
```

```
IsolationForest(contamination=0.03, n_estimators=10, n_jobs=-1, random_state=42)
```

```
[120]  1  print("Matriz de confusión: \n", confusion_matrix(yTest, ypred))
```

```
Matriz de confusión:
 [[     0      0      0]
 [     0      0 129604]
 [  4006      0   5418]]
```

```
1  # Evaluate the model
2  print(classification_report(yTest, ypred))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.00      0.00      0.00    129604
           1       0.04      0.57      0.08      9424

    accuracy                           0.04    139028
   macro avg       0.01      0.19      0.03    139028
weighted avg       0.00      0.04      0.01    139028
```

Contaminacion= 0.02

```
1  # Split the data into training and testing sets
2  xtrain, xTest, ytrain, yTest = train_test_split(x,y,test_size=0.3,random_state=1)
3  model = IsolationForest(n_estimators=10, max_samples='auto', contamination=float(.02), max_features=1.0, bootstrap=False, n_jobs=-1, random_state=42,v
4  model.fit(xtrain)
```

```
IsolationForest(contamination=0.02, n_estimators=10, n_jobs=-1, random_state=42)
```

```
[131]  1  print("Matriz de confusión: \n", confusion_matrix(yTest, ypred))
```

```
Matriz de confusión:
 [[     0      0      0]
 [     0      0 129604]
 [  2351      0   7073]]
```

```
1  # Evaluate the model
2  print(classification_report(yTest, ypred))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.00      0.00      0.00    129604
           1       0.05      0.75      0.10      9424

    accuracy                           0.05    139028
   macro avg       0.02      0.25      0.03    139028
weighted avg       0.00      0.05      0.01    139028
```

Realizamos otros cambios en los parámetros como el número de estimadores:

```
[133] 1  # Split the data into training and testing sets
      2  xtrain, xTest, ytrain, yTest = train_test_split(x,y,test_size=0.3,random_state=1)
      3  model = IsolationForest(n_estimators=5, max_samples='auto', contamination=float(.02), max_features=1.0, bootstrap=False, n_jobs=-1, random_state=42,ve
      4  model.fit(xtrain)
```

```
IsolationForest(contamination=0.02, n_estimators=5, n_jobs=-1, random_state=42)
```

```
[137] 1  print("Matriz de confusión: \n", confusion_matrix(yTest, ypred))
```

```
Matriz de confusión:
 [[     0      0      0]
 [     0      0 129604]
 [  2399      0   7025]]
```

```
1  # Evaluate the model
2  print(classification_report(yTest, ypred))
```

```
              precision    recall  f1-score   support

          -1       0.00      0.00      0.00         0
           0       0.00      0.00      0.00    129604
           1       0.05      0.75      0.10      9424

    accuracy                           0.05    139028
   macro avg       0.02      0.25      0.03    139028
weighted avg       0.00      0.05      0.01    139028
```

```
[80] 1  #Curva ROC
```

No se tuvo mayor variación, siendo este una mejor combinación de parametros para mejorar los resultados.

Conclusión : La combinacion con una contaminacion de 0.02, fue la que mejor resultado nos genero. Adicional a este modelo de isolation forest realizamos pruebas con Local outlier pero su resultado no mejor lo obtenido con isolation forest.

```
[133] 1  # Split the data into training and testing sets
      2  xtrain, xTest, ytrain, yTest = train_test_split(x,y,test_size=0.3,random_state=1)
      3  model = IsolationForest(n_estimators=5, max_samples='auto', contamination=float(.02), max_features=1.0, bootstrap=False, n_jobs=-1, random_state=42,ve
      4  model.fit(xtrain)
```

```
IsolationForest(contamination=0.02, n_estimators=5, n_jobs=-1, random_state=42)
```

Finalmente realizmos la validacion de IQR donde obtuvimos la cantidad de ouliers con sus valores minimos y maximos los cuales fueron de gran utilidad para el ejercicio final.

[46]
```python
#Analizamos estadisticamente la deteccion de outliers, para esto creamos una funcion
def find_outliers_IQR(dfCliente5):
  q1=dfCliente5.quantile(0.25)
  q3=dfCliente5.quantile(0.75)
  IQR=q3-q1
  outliers = dfCliente5[((dfCliente5<(q1-1.5*IQR)) | (dfCliente5>(q3+1.5*IQR)))]
  return outliers
```

[47]
```python
# llamamos la funcion creada
outliers = find_outliers_IQR(dfCliente5["Active_energy"])
print("number of outliers: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))
outliers
```

```
number of outliers: 64
max outlier value: 14.622644283216186
min outlier value: 11.78592931792592
336577    12.820507
336597    12.393231
336642    11.888171
336778    12.297893
336985    11.857016
             ...
341601    12.694572
341619    11.926254
341713    11.972343
341902    12.643094
341906    12.044382
Name: Active_energy, Length: 64, dtype: float64
```