Clustering Applications on Google Brain Ventilation Pressure dataset

---

JUNE 16, 2022

MUHAMMAD AMIN GHIAS

Table of Contents

ABSTRACT:

Health is the most important aspect of life, people need to take extra precautions to remain healthy. In this era breathing/lungs problems has become common; this project will analyze the data apply clustering models and then predict the airway pressure in the respiratory circuit during the breath with and without application of clustering models.

INTRODUCTION

As breathing problems has become common nowadays it is important for people to understand what causes lungs problem. This report will analyze the data, cluster our dataset. Further more using the dataset this project will predict people who are likely to have heart attack.

MAIN OBJECTIVE:

The main objective of this project is to study the application of clustering models and its impact/help in predicting the airway pressure in the respiratory circuit during the breath

DATA ACQUISITION AND WRANGLING

DATA SOURCE

For this project we will use online data the source is from kaggle

CSV files are from url:

Train.csv

https://www.kaggle.com/competitions/ventilator-pressure-prediction/data?select=train.csv

Test.csv

https://www.kaggle.com/competitions/ventilator-pressure-prediction/data?select=test.csv

The ventilator data used was produced using a modified open-source ventilator connected to an artificial bellows test lung via a respiratory circuit.

The Dataset consists of 8 columns and 6036000 rows

DATA DESCRIPTION

The description of columns are:

- id - globally-unique time step identifier across an entire file
- breath_id - globally-unique time step for breaths
- R - lung attribute indicating how restricted the airway is (in cmH2O/L/S). Physically, this is the change in pressure per change in flow (air volume per time). Intuitively, one can imagine blowing up a balloon

through a straw. We can change R by changing the diameter of the straw, with higher R being harder to blow.

- C - lung attribute indicating how compliant the lung is (in mL/cmH2O). Physically, this is the change in volume per change in pressure. Intuitively, one can imagine the same balloon example. We can change C by changing the thickness of the balloon's latex, with higher C having thinner latex and easier to blow.
- time_step - the actual time stamp.
- u_in - the control input for the inspiratory solenoid valve. Ranges from 0 to 100.
- u_out - the control input for the exploratory solenoid valve. Either 0 or 1.
- pressure - the airway pressure measured in the respiratory circuit, measured in cmH2O.

```
1  df_original= pd.read_csv("train.csv")
2
3  print(df_original.shape)
4  df_original.head()
```

✓ 3.4s

```
(6036000, 8)
```

	id	breath_id	R	C	time_step	u_in	u_out	pressure
0	1	1	20	50	0.000000	0.083334	0	5.837492
1	2	1	20	50	0.033652	18.383041	0	5.907794
2	3	1	20	50	0.067514	22.509278	0	7.876254
3	4	1	20	50	0.101542	22.808822	0	11.742872
4	5	1	20	50	0.135756	25.355850	0	12.234987

```
1  df.info()
```

✓ 0.4s

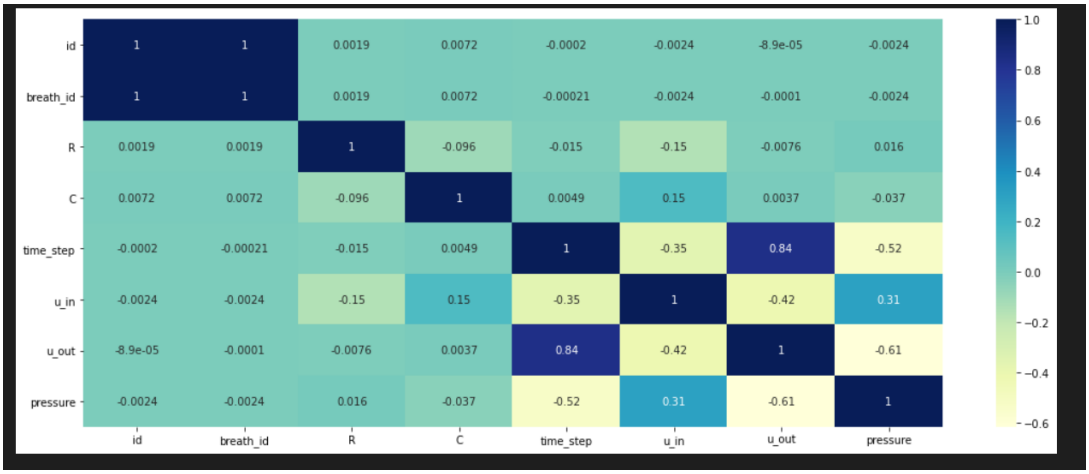
```
<class 'pandas.core.frame.DataFra
RangeIndex: 6036000 entries, 0 to
Data columns (total 8 columns):
 #   Column     Dtype
---  ------     -----
 0   id         int64
 1   breath_id  int64
 2   R          int64
 3   C          int64
 4   time_step  float64
 5   u_in       float64
 6   u_out      int64
 7   pressure   float64
dtypes: float64(3), int64(5)
memory usage: 368.4 MB
```

```
1  df.isnull().sum()
```
✓ 0.1s

```
id           0
breath_id    0
R            0
C            0
time_step    0
u_in         0
u_out        0
pressure     0
dtype: int64
```

DATA CLEANING & ANALYSIS

Finding correlation matrix



Finding skewness

```
1  skew_columns = (df.loc[:, df.columns != 'pressure']
2                  .skew()
3                  .sort_values(ascending=False))
4
5  skew_columns = skew_columns.loc[skew_columns > 0.75]
6  skew_columns
```
✓ 1.6s

```
NumExpr defaulting to 8 threads.

u_in    3.912228
dtype: float64
```

Scaling the dataset

```python
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

dfss=df_skew.copy()
dfss[dr.columns.values] = sc.fit_transform(dfss[dr.columns.values])

dfss.head(4)
```

✓ 3.9s

	id	breath_id	R	C	time_step	u_in	u_out	pressure
0	1	1	-0.359072	1.394522	0.000000	-1.253479	-1.278552	5.837492
1	2	1	-0.359072	1.394522	0.033652	1.380412	-1.278552	5.907794
2	3	1	-0.359072	1.394522	0.067514	1.556650	-1.278552	7.876254
3	4	1	-0.359072	1.394522	0.101542	1.568212	-1.278552	11.742872

Making a smaller dataset to apply clustering modles

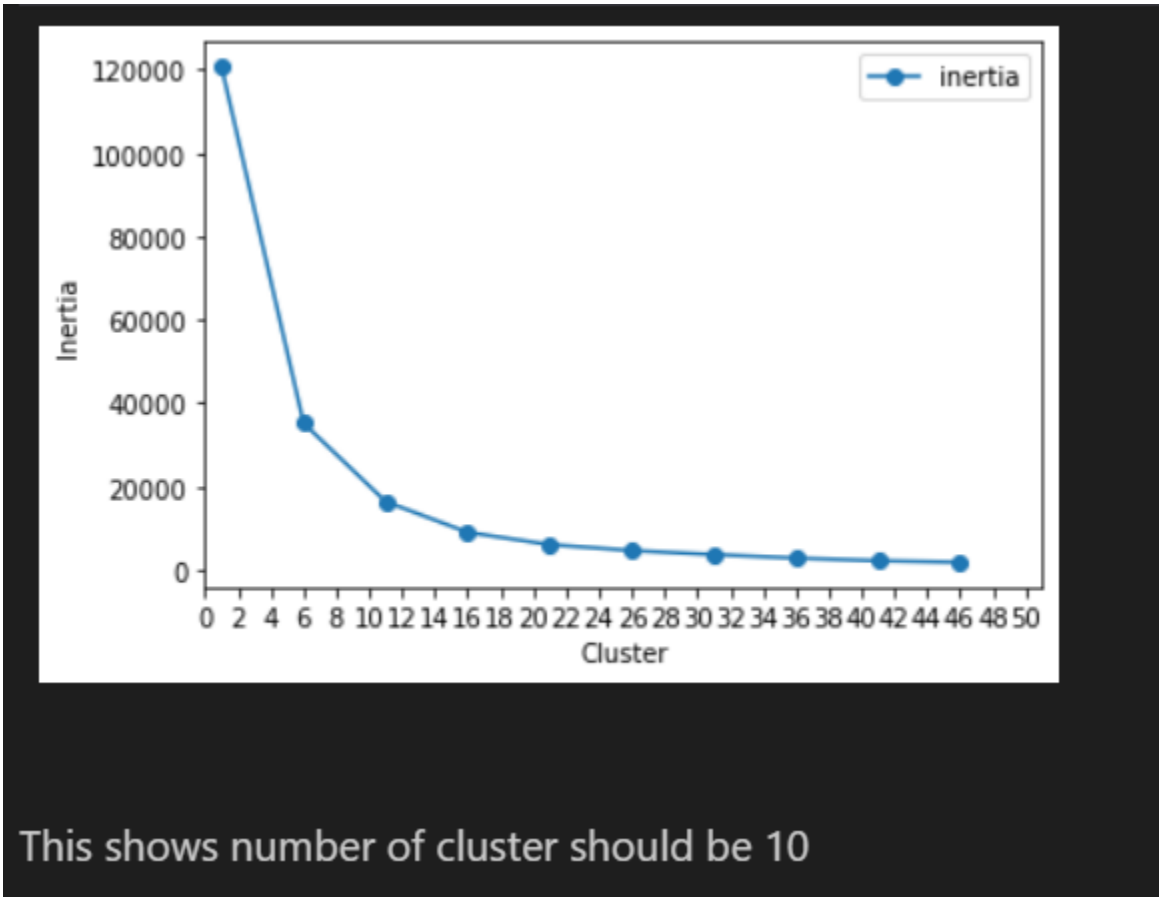
Making a smaller dataset and applying scaling

+ Code + Markdown

```python
dfp=df_skew.sample(frac =.005)
print(dfp.shape)

dfpc=dfp.copy()

xss=df_skew[['R','C','u_in','u_out']].copy()
xss.shape

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
dfps=dfp.copy()
dfps[xss.columns.values] = sc.fit_transform(dfps[xss.columns.values])



Xps = dfps[xss.columns.values]
print(Xps.shape)
Xps.head()
```

Finding the number of clusters to for Kmeans

```
km_list = list()

for clust in range(1,50,5):
    km = KMeans(n_clusters=clust, random_state=42)
    km = km.fit(Xps)

    km_list.append(pd.Series({'clusters': clust,
                              'inertia': km.inertia_,
                              'model': km}))


plot_data = (pd.concat(km_list, axis=1)
             .T
             [['clusters','inertia']]
             .set_index('clusters'))

ax = plot_data.plot(marker='o',ls='-')
ax.set_xticks(range(0,51,2))
ax.set_xlim(0,51)
ax.set(xlabel='Cluster', ylabel='Inertia');
### END SOLUTION
```

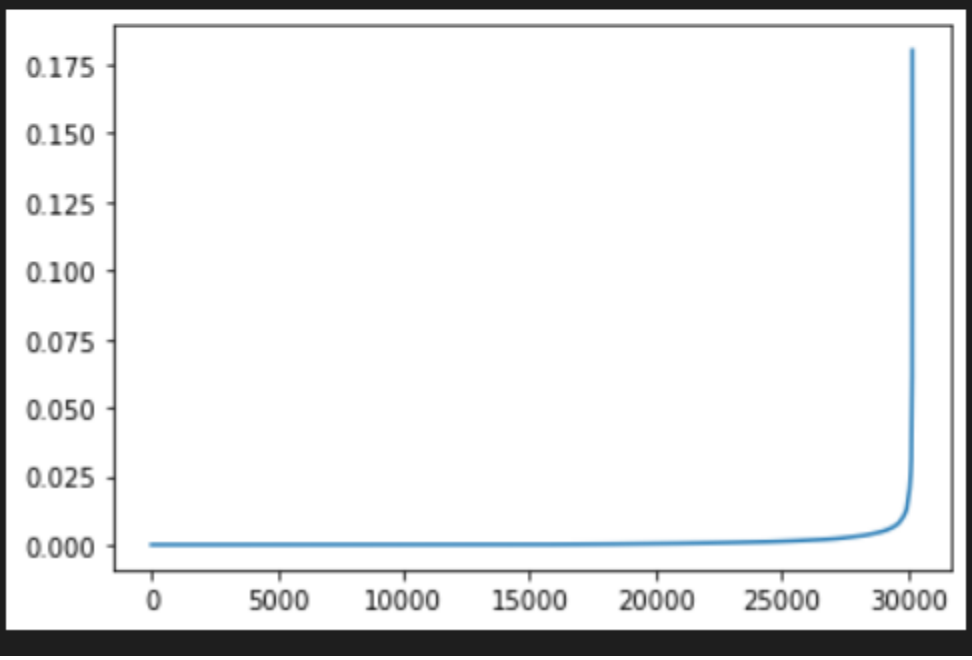


Finding the epsilon for dbscan

```
1  from sklearn.neighbors import NearestNeighbors
2  from matplotlib import pyplot as plt
3
4
5  neighbors = NearestNeighbors(n_neighbors=16)
6  neighbors_fit = neighbors.fit(Xps)
7  distances, indices = neighbors_fit.kneighbors(Xps)
8
9
10 distances = np.sort(distances, axis=0)
11 distances = distances[:,1]
12 plt.plot(distances)
13
```

✓ 0.6s

```
[<matplotlib.lines.Line2D at 0x1d194c9f610>]
```



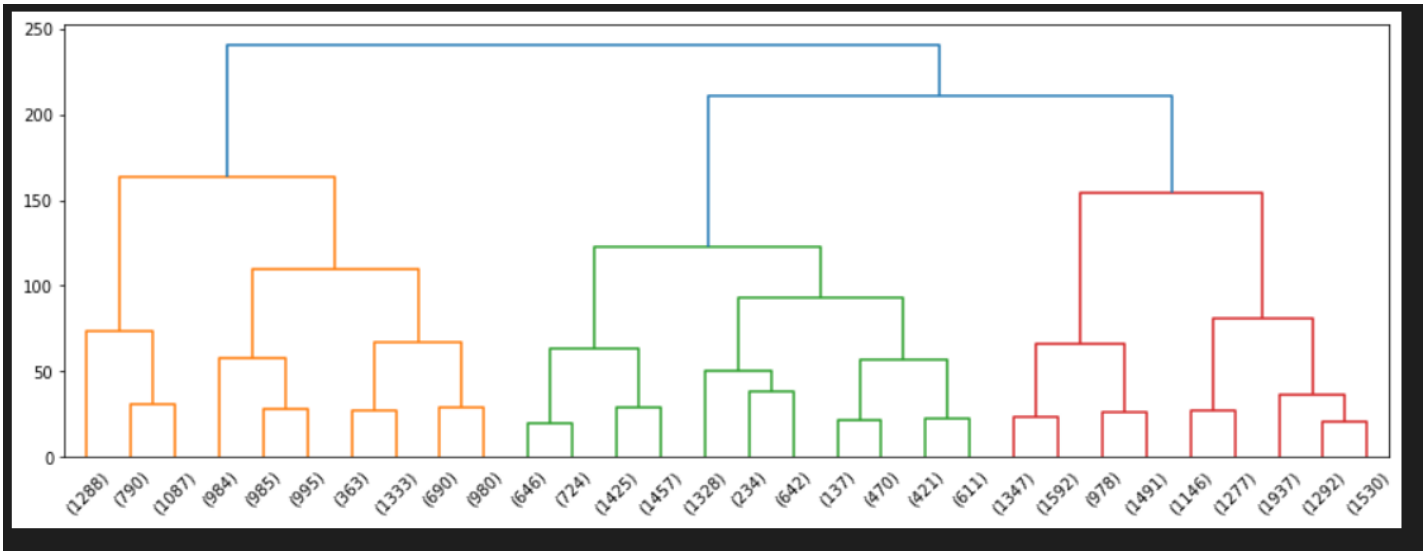
Applying 3 clustering models

Kmeans

```
km = KMeans(n_clusters=10, random_state=42)
km = km.fit(Xps)

dfpc['kmeans'] = km.predict(Xps)
```

Agglomerative clustering

```
from sklearn.cluster import AgglomerativeClustering
### BEGIN SOLUTION
ag = AgglomerativeClustering(n_clusters=10, linkage='ward', compute_full_tree=True)
ag = ag.fit(Xps)
dfpc['agglom'] = ag.fit_predict(Xps)


# First, we import the cluster hierarchy module from SciPy (described above) to obtai
from scipy.cluster import hierarchy

Z = hierarchy.linkage(Xps, method='ward')

fig, ax = plt.subplots(figsize=(15,5))



den = hierarchy.dendrogram(Z, orientation='top',
                           p=30, truncate_mode='lastp',
                           show_leaf_counts=True, ax=ax
                          )
### END SOLUTION
```



DBscan

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps = 0.15, min_samples = 16)
db=db.fit(Xps)
print(db.labels_)

dfpc['dbscan'] = db.fit_predict(Xps)
```

Combined result of 3 clustering models

	id	breath_id	R	C	time_step	u_in	u_out	pressure	kmeans	agglom	dbscan
4334979	4334980	90320	50	20	0.606581	1.674389	0	14.976771	5	0	0
2323966	2323967	48331	50	10	1.455553	0.000000	1	6.259305	2	1	1
818711	818712	17005	5	50	2.285471	1.777303	1	5.345377	3	2	2
700454	700455	14514	20	50	1.856889	1.663537	1	7.524743	3	2	3
5054052	5054053	105271	5	20	1.793038	1.611308	1	5.837492	7	7	4

Predicting clusters on test data

```
from sklearn.cluster import KMeans
### BEGIN SOLUTION
km = KMeans(n_clusters=10, random_state=42)
km = km.fit(X)
print(km.inertia_)
dftc['kmeans'] = km.predict(Xt)
dftc.head()
```

✓ 1m 19.5s

```
3759990.97194655
```

	id	breath_id	R	C	time_step	u_in	u_out	pressure	kmeans
0	1	1	20	50	0.000000	0.080043	0	5.837492	1
1	2	1	20	50	0.033652	2.964399	0	5.907794	2
2	3	1	20	50	0.067514	3.157395	0	7.876254	2
3	4	1	20	50	0.101542	3.170056	0	11.742872	2
4	5	1	20	50	0.135756	3.271690	0	12.234987	2

Using Clustering as Feature engineering

Applying Kmeans, then using random forrest with and without Kmeans to find effect of kmeans of score

```
y = dfc['pressure']
X_with_kmeans = dfc.drop(['pressure'], axis=1)

X_without_kmeans = X_with_kmeans.drop('kmeans', axis=1)


trainX, testX, trainy, testy = train_test_split(X_with_kmeans, y, test_size=0.3, random_state=2)

rf_km = RandomForestRegressor(max_depth=5,n_estimators=50, verbose=2, n_jobs=-1)
rf_km.fit(trainX,trainy)
ypred_km= rf_km.predict(testX)
r2_km= r2_score(testy, ypred_km)
mse_km= mean_squared_error(testy, ypred_km)

print("Mean squared error with Kmeans: %.2f" % mse_km)

print("R2 score with Kmeans: %.3f" % r2_km)


trainX, testX, trainy, testy = train_test_split(X_without_kmeans, y, test_size=0.3, random_state=2)

rf = RandomForestRegressor(max_depth=5,n_estimators=50, verbose=2, n_jobs=-1)
rf.fit(trainX,trainy)
ypred= rf.predict(testX)
r2= r2_score(testy, ypred)
mse= mean_squared_error(testy, ypred)

print("Mean squared error without Kmeans: %.2f" % mse)

print(" R2 score without Kmeans: %.3f" % r2)
```

```
print(" R2 score without Kmeans: ",  r2)
print("R2 score with Kmeans: " , r2_km)

print("Mean squared error without Kmeans: ", mse)
print("Mean squared error with Kmeans: ",  mse_km)
```

✓ 0.4s

```
 R2 score without Kmeans:  0.6363652279398723
R2 score with Kmeans:  0.6361039532547195
Mean squared error without Kmeans:  23.898443278255808
Mean squared error with Kmeans:  23.91561451358018
```

RESULTS:

This shows that Kmeans clustering can be applied on complete dataset hence it is better

There was limitation of memory (memory error for agglomerative clustering) and also for DBscan

DISCUSSION:

In this project the errors from different classification models was calculated:

- Kmeans
- Agglomerative clustering (with ward linkage)
- DBscan

Also the best number of clusters for kmeans was calculated using inertia values and epsilon value for DBscan was calculated

Lastly Kmean clustering was used as feature engineering to find effect of clustering

CONCLUSION

The results of score showed that R2 and MSE score is nearly similar with and without kmeans clustering

scores are Slightly greater without kmeans clustering

SUGGESTIONS

The dataset can include more features for better training of models.