DR-Clustering-Python

November 18, 2021

Import libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import plotly.express as px
pd.options.plotting.backend = "plotly"
import seaborn as sns
import dataframe_image as dfi
```

Import and clean data

[2]:
```
df = pd.read_csv('datasets/users.db.csv')
df.drop(columns=['last.pr.update'], inplace=True) # this column only contains␣
↪NaN value
df.columns = df.columns.str.replace('.', '_', regex=False)
df[['last_up_photo', 'last_connex', 'date_crea']] = df[['last_up_photo',␣
↪'last_connex', 'date_crea']].apply(pd.to_datetime)
df['account_age'] = (df['last_connex'] - df['date_crea']).dt.days # Create new␣
↪account age feature
df.drop(df[df['account_age'] < 0].index, inplace = True)
df['gender'].replace({0 : 'Male', 1 : 'Female', 2 : np.nan}, inplace=True)
df['voyage'].replace({0 : 'No', 1 : 'Yes'}, inplace=True)
df['laugh'].replace({0 : 'No', 1 : 'Yes'}, inplace=True)
df['photo_keke'].replace({0 : 'No', 1 : 'Yes'}, inplace=True)
df['photo_beach'].replace({0 : 'No', 1 : 'Yes'}, inplace=True)
df.dropna(inplace = True)
df.head()
```

[2]:
```
   userid  date_crea     score  n_matches  n_updates_photo  n_photos  \
0       1 2011-09-17  1.495834         11                5         6
1       2 2017-01-17  8.946863         56                2         6
2       3 2019-05-14  2.496199         13                3         4
3       4 2015-11-27  2.823579         32                5         2
4       5 2014-11-28  2.117433         21                1         4

  last_connex last_up_photo  gender  sent_ana  length_prof voyage laugh  \
```

```
0  2011-10-07    2011-10-02  Female  6.490446    0.000000     No     No
1  2017-01-31    2017-02-03  Female  4.589125   20.722862     No     No
2  2019-06-17    2019-06-19  Female  6.473182   31.399277     No     No
3  2016-01-15    2015-12-09    Male  5.368982    0.000000     No     No
4  2015-01-15    2015-01-02    Male  5.573949   38.510225     No    Yes

  photo_keke photo_beach  account_age
0         No          No           20
1         No         Yes           14
2         No         Yes           34
3         No         Yes           49
4         No          No           48
```

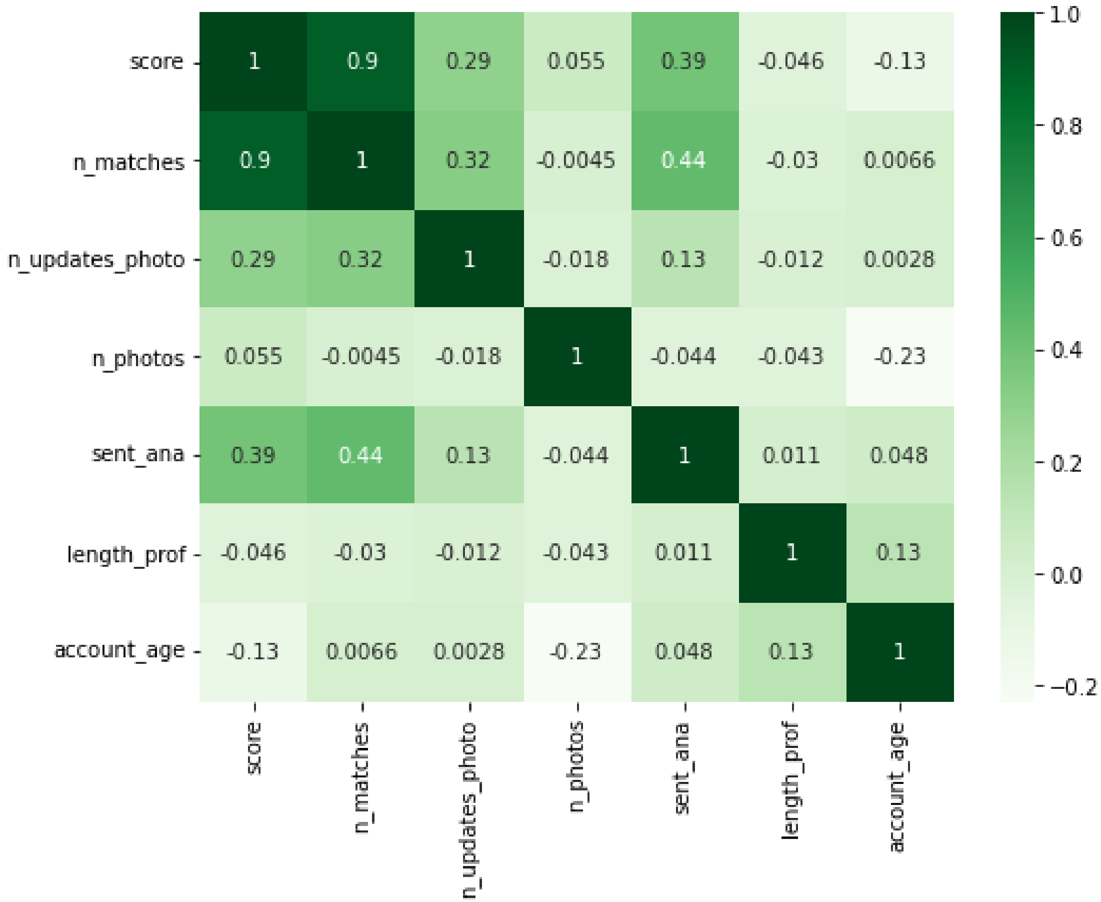
[3]:
```
#dfi.export(df.head(10), "plots/df_head.png")
#df.to_csv('datasets/data.csv')
```

1 Identifying correlations in the variables

[4]:
```
categorical_features = ['voyage', 'laugh', 'photo_keke', 'photo_beach',␣
 ↪'gender']
continuous_features  = ['score', 'n_matches', 'n_updates_photo', 'n_photos',␣
 ↪'sent_ana', 'length_prof', 'account_age']
```

Pearson correlation for continuous variables

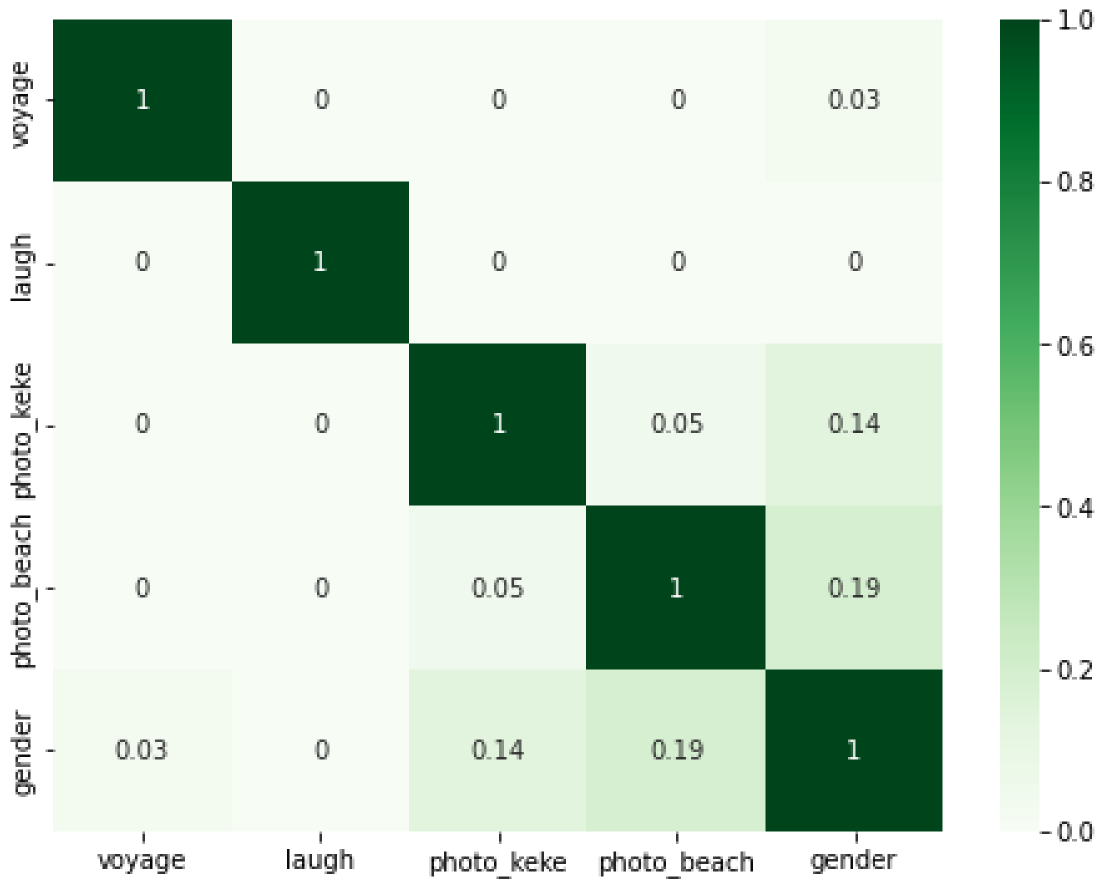
[5]:
```
pearson_corr = df[continuous_features].corr()
plt.figure(figsize=(8,6))
sns.heatmap(pearson_corr, cmap="Greens",annot=True)
plt.savefig("plots/pearson_corr.png")
```



Cramers V correlation for categorial variables

[6]:
```
from scipy.stats import chi2_contingency

def cramers_corrected_stat(confusion_matrix):
    """ calculate Cramers V statistic for categorial-categorial association.
        uses correction from Bergsma and Wicher,
        Journal of the Korean Statistical Society 42 (2013): 323-328
    """
    chi2 = chi2_contingency(confusion_matrix)[0]
    n = confusion_matrix.sum()
    phi2 = chi2/n
    r,k = confusion_matrix.shape
    phi2corr = max(0, phi2 - ((k-1)*(r-1))/(n-1))
    rcorr = r - ((r-1)**2)/(n-1)
    kcorr = k - ((k-1)**2)/(n-1)
    return np.sqrt(phi2corr / min( (kcorr-1), (rcorr-1)))

rows= []
```

```python
data_encoded = df[categorical_features]

for var1 in data_encoded:
  col = []
  for var2 in data_encoded :
    confusion_matrix = pd.crosstab(df[var1], df[var2])
    cramers = cramers_corrected_stat(confusion_matrix.values) # Cramer's V test
    col.append(round(cramers, 2)) # Keeping of the rounded value of the␣
↪Cramer's V
  rows.append(col)

cramers_results = np.array(rows)
cramers_corr = pd.DataFrame(cramers_results, columns = data_encoded.columns,␣
↪index =data_encoded.columns)
plt.figure(figsize=(8,6))
sns.heatmap(cramers_corr, cmap="Greens",annot=True)
plt.savefig("plots/cramers_corr.png")
```



Account age and Gender

[7]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

mod = ols('account_age ~ gender',
                data=df).fit()

aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
df.plot.box(x='gender', y='account_age',
           labels={
                   "gender": "Gender",
                   "account_age": "Age of account"}
           )
```

```
              df          sum_sq        mean_sq            F  PR(>F)
gender       1.0  406572.062810  406572.062810  2782.040071     0.0
Residual  2929.0  428049.036474     146.141699          NaN     NaN
```

[8]:
```
#dfi.export(aov_table, "plots/account_age~gender.png")
```

Number of photos and Gender

[9]:
```
mod = ols('n_photos ~ gender',
                data=df).fit()

aov_table = sm.stats.anova_lm(mod, typ=1)
print(aov_table)
df.plot.box(x='gender', y='n_photos',
           labels={
                   "gender": "Gender",
                   "n_photos": "Number of photos"}
           )
```

```
              df        sum_sq     mean_sq           F        PR(>F)
gender       1.0    872.884180  872.88418  331.620261  2.792013e-70
Residual  2929.0   7709.654885    2.63218         NaN           NaN
```

[10]:
```
#dfi.export(aov_table, "plots/n_photos~gender.png")
```

2 Dimensional Reduction

[11]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standard scaler
scaler = StandardScaler()
pca_features = ['n_matches', 'n_updates_photo', 'n_photos', 'sent_ana',␣
 ↪'length_prof', 'account_age']
```

```
df[pca_features] = scaler.fit_transform(df[pca_features])

# Apply PCA
pca = PCA(n_components=2)
components = pca.fit_transform(df[pca_features])
```

2.1 K-Means

[12]:
```
from sklearn.cluster import KMeans

ks = range(1, 10)

inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(components)

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

fig = px.line(x = ks, y = inertias, markers=True,
            labels={'x':'Number of clusters (k)', 'y' : 'Inertia %'})

fig.show()
```

[13]:
```
labels = KMeans(3, random_state=0).fit_predict(components)

fig = px.scatter(components, x=0, y=1, color=labels,
                labels={'0':'PC1', '1' : 'PC2'})
fig.show()
```

Clustering report function

[14]:
```
from IPython.display import display, HTML
from sklearn.tree import _tree, DecisionTreeClassifier

def pretty_print(df):
    return display( HTML( df.to_html().replace("\\n","<br>") ) )

def get_class_rules(tree: DecisionTreeClassifier, feature_names: list):
  inner_tree: _tree.Tree = tree.tree_
  classes = tree.classes_
  class_rules_dict = dict()

  def tree_dfs(node_id=0, current_rule=[]):
```

```
    # feature[i] holds the feature to split on, for the internal node i.
    split_feature = inner_tree.feature[node_id]
    if split_feature != _tree.TREE_UNDEFINED: # internal node
      name = feature_names[split_feature]
      threshold = inner_tree.threshold[node_id]
      # left child
      left_rule = current_rule + ["({} <= {})".format(name, threshold)]
      tree_dfs(inner_tree.children_left[node_id], left_rule)
      # right child
      right_rule = current_rule + ["({} > {})".format(name, threshold)]
      tree_dfs(inner_tree.children_right[node_id], right_rule)
    else: # leaf
      dist = inner_tree.value[node_id][0]
      dist = dist/dist.sum()
      max_idx = dist.argmax()
      if len(current_rule) == 0:
        rule_string = "ALL"
      else:
        rule_string = " and ".join(current_rule)
      # register new rule to dictionary
      selected_class = classes[max_idx]
      class_probability = dist[max_idx]
      class_rules = class_rules_dict.get(selected_class, [])
      class_rules.append((rule_string, class_probability))
      class_rules_dict[selected_class] = class_rules

  tree_dfs() # start from root, node_id = 0
  return class_rules_dict

def cluster_report(data: pd.DataFrame, clusters, min_samples_leaf=50,␣
↪pruning_level=0.01):
    # Create Model
    tree = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf,␣
↪ccp_alpha=pruning_level)
    tree.fit(data, clusters)

    # Generate Report
    feature_names = data.columns
    class_rule_dict = get_class_rules(tree, feature_names)

    report_class_list = []
    for class_name in class_rule_dict.keys():
        rule_list = class_rule_dict[class_name]
        combined_string = ""
        for rule in rule_list:
            combined_string += "[{}] {}\n\n".format(rule[1], rule[0])
        report_class_list.append((class_name, combined_string))
```

```
    cluster_instance_df = pd.Series(clusters).value_counts().reset_index()
    cluster_instance_df.columns = ['class_name', 'instance_count']
    report_df = pd.DataFrame(report_class_list, columns=['class_name',␣
↪'rule_list'])
    report_df = pd.merge(cluster_instance_df, report_df, on='class_name',␣
↪how='left')
    pretty_print(report_df.sort_values(by='class_name')[['class_name',␣
↪'instance_count', 'rule_list']])
```

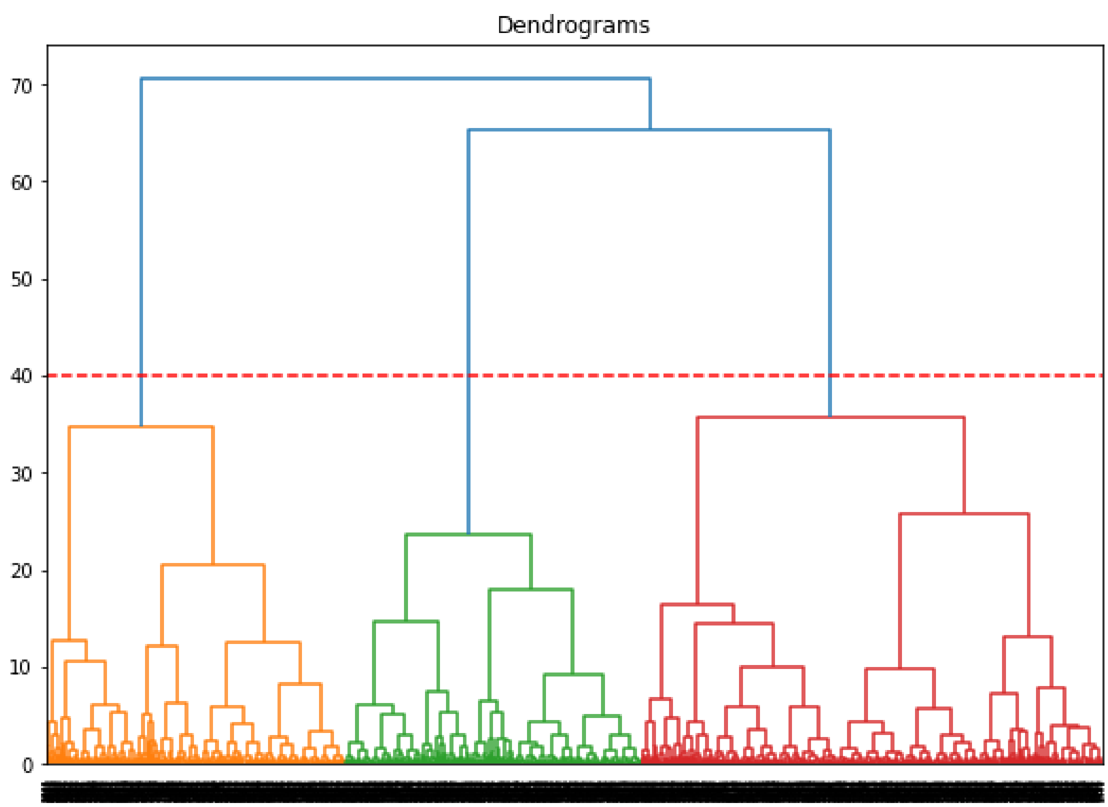
[15]:
```
cluster_report(pd.DataFrame(components, columns = ['PC1','PC2']), labels,␣
↪min_samples_leaf=20, pruning_level=0.05)
```

```
<IPython.core.display.HTML object>
```

2.2 Hierarchical clustering

[16]:
```
import scipy.cluster.hierarchy as shc
plt.figure(figsize=(10, 7))
plt.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(components, method='ward'))
plt.axhline(y=40, color='r', linestyle='--')
```

[16]:
```
<matplotlib.lines.Line2D at 0x14a46d790>
```



From the dendograms, we choose 3 as the number of clusters

[17]:
```
from sklearn.cluster import AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean',␣
 ↪linkage='ward')
cluster.fit_predict(components)
```

[17]:
```
array([1, 1, 0, …, 1, 1, 1])
```

[18]:
```
fig = px.scatter(components, x=0, y=1, color=cluster.labels_,
                 labels={'0':'PC1', '1' : 'PC2'})
fig.show()
```

[19]:
```
cluster_report(pd.DataFrame(components, columns = ['PC1','PC2']), cluster.
 ↪labels_, min_samples_leaf=20, pruning_level=0.05)
```

```
<IPython.core.display.HTML object>
```