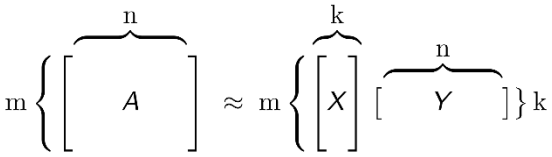
Analysis of Mixed Data Sets

R. Clukey, Data Scientist | 2021

An Application of Generalized Low Rank Models to Compress Heterogenous Survey Data using the H20.ai API in Python

Data Scientists often encounter data sets which contain a large number of features of mixed data types (i.e. scale, ordinal, categorical). This data may come from survey data, data warehouses, or the processes of aggregating unstructured data. When all the data consists of scale-level features a common pre-processing step is to reduce the number of dimensions using Principal Component Analysis (PCA), or Factor Analysis techniques. However, these techniques are not suitable for complex heterogenous data types, which makes data exploration and visualization difficult. Additionally, the analyst is usually interested in subsequent analysis steps on the data, such as Cluster analysis or Regression analysis, which makes dimensionality reduction a necessary step.

This paper provides an overview of mixed-type dimension reduction using the Generalized Low Rank Model algorithm from H20.ai. A Low Rank model is a condensed vector representation for every row and column in the dataset.[1] GLRM decomposes a heterogenous data set into k number of columns representing the compression of the original data into a smaller number of k dimensions. Because of its ability to handle complex and mix-data types, this technique is suitable alternative to situations where PCA cannot be readily applied.



In the diagram above, the GLRM algorithm creates projections from A to the reduced spaces X and Y, which are called Archetypical Representations (Y) and Archetypical Features (X); similar to principal components. The formal definition is described as[2]:

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij}) + \sum_{i=1}^{m} r_i(x_i) + \sum_{j=1}^{n} \tilde{r}_j(y_j),$$

Thus, we can recover the projections of the X matrix as feature compressions for each row of the original dataset A, yielding k

[1] http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/glrm.html

[2] https://people.orie.cornell.edu/mru8/doc/udell14_glrm_short.pdf

vectors to utilize in subsequent Machine Learning algorithms (i.e. Cluster Analysis).

This paper will demonstrate the application of GLRM dimensionality reduction using a survey data set from which customer segments needed to be derived. For this analysis we are using Python 3.7 with he H20.ai API version 3.30.1.3 (~ November 2020).

Data

The data used in this demonstration is from a survey of users of particular SaaS application. Each survey respondent provided answers to a variety of questions including Likert-type scales (ordinal 1 – 7), as well as a variety of categorical variables indicating different demographic-type responses: age, gender, department, etc. These variables might typically be one-hot encoded, or excluded altogether from PCA and Cluster Analysis methods.

```
# Import the Python modules
import h2o
import pandas as pd
import numpy as np
from h2o.estimators import
H2OGeneralizedLowRankEstimator
h2o.init()

# Import the data
df = pd.read_csv("test_data.csv")
df.head()
```

This code snippet loads the required Python modules, initializes H2o and loads the dataset. (shown below)

q5	q6	q7	q8	q9	...	q17	q18	q19	q20	q21	q22
3-4 ears	Weekly	Administrator (admin)	Yes	6+	...	2.14605	2.54	2.06	2.22	2.70	1.71
< 1 year	Weekly	Administrator (admin)	Yes	2-5	...	0.46599	-0.50	0.10	0.04	-0.08	-0.23
1-2 ears	Daily	Administrator (admin)	Yes	2-5	...	-0.43247	-0.50	0.23	-0.64	-0.08	-0.04
3-4 ears	Monthly	Administrator (admin)	Yes	6+	...	2.56395	-0.50	3.14	2.89	3.61	1.98
3-4 ears	Weekly	Administrator (admin)	Yes	6+	...	1.59462	0.15	1.82	0.92	1.79	1.23

This data consists of a mixture of data types, both categorical, scale and ordinal, as well as missing values where respondents opted not to provide a response. In those instances we can either leave the response empty, or we can impute. For categorical data we can impute using the Mode, for continuous and ordinal data we can impute using the mean. However, for our purposes in this example we will not impute any values and let the algorithm run with the missing data included.

An additional step not shown here is to explicitly cast the specific variables in the dataset as Categorical. In Python:

```
# recast categorical variables
df['q25'] =
df['q25'].astype('category')
```

This tells the GRLM algorithm to treat those variables as categorical.

Recall, the goal here is to reduce the dimensionality of the dataset from n dimensions to k vectors.

Run the GLRM

The first step is to cast the DataFrame as an H2O DataFrame, which is a special format for the H2O API.

```
# recast DataFrame to H2O format
hf_df = h2o.H2OFrame(df)
```

Once the dataframe has ben recast, we can build the GRLM model. There are a number of parameters to consider when constructing the algorithm. We will summarize a few of the main parameters of interest below:

k – the number of projected vectors (similar to specifying the number of components in PCA)

loss – specifies the numeric loss function: Quadratic is the default. Other options include Absolute, Huber, Poisson, Hinge, and Periodic.

gamma_x – allows the user to define the regularization weight on the X matrix (default is 0).

gamma_y – allows the user to define the regularization weight on the Y matrix (default is 0).

max_iterations – specifies the number of training iterations.

recover_svd – allows the recovery of singular values (SVD) and Eigenvectors of the XY matrices.

init – Specifies the initialization mode: Random, PlusPlus, furthest, or User

transform – specifies the transformation method to perform on the dataset prior to running the algorithm. This includes None, Standardize, Normalize, Demean, Descale. The default is none and the specific type of transformation will depend on the data and the nature of analysis that is needed.

In the example below, we're specifying the number of dimensions (k) as 2, setting the loss to "Quadratic" and setting the regularization parameters to .5.

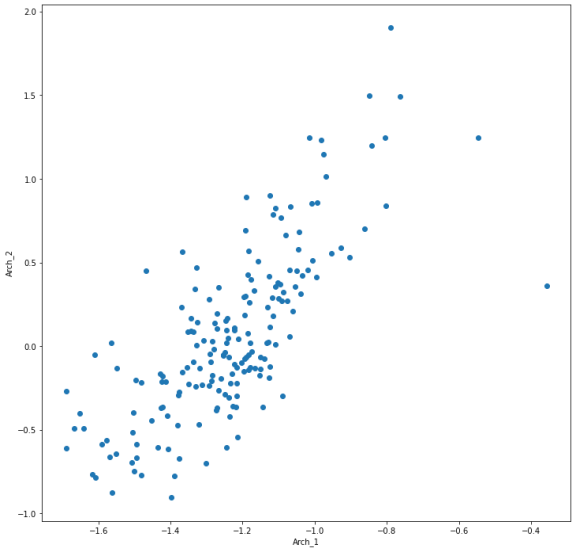
```
# Build and Train the model
glrm_model =
H2OGeneralizedLowRankEstimator(
 k=2,
 loss="quadratic",
 gamma_x=0.5,
 gamma_y=0.5,
 max_iterations=700,
 recover_svd=True,
 init="SVD",
 transform=None)

glrm_model.train(x = hf_df.names,
training_frame= hf_df)
```

The next step is to actually recover the X matrix with the compressed values.

```
# Get the X Matrix Architypes
fit_x =
h2o.get_frame(glrm_model._model_j
son['output']['representation_nam
e'])
glrm_x =
np.array(h2o.as_list(fit_x))
glrm_df = pd.DataFrame(glrm_x)
glrm_df
```

	0	1
0	-1.270561	0.194112
1	-1.182037	0.260923
2	-0.842117	1.198874
3	-1.312825	-0.232226
4	-1.366809	-0.156485
...	...	...
178	-1.035683	0.420307
179	-1.428042	-0.167120
180	-1.331443	0.340075
181	-1.467006	0.448709
182	-1.667228	-0.490706

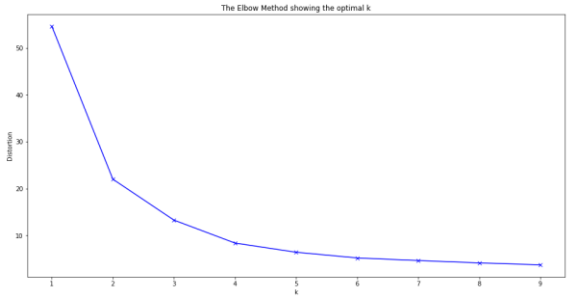
The output above now shows each of the 2 k vectors as a Pandas DataFrame. We can plot the results in Python.



Now that we have the data reduced to 2 dimensions, including all categorical, scale, and ordinal variables, we can uses these new vectors as inputs to our K-Means Clustering algorithm.

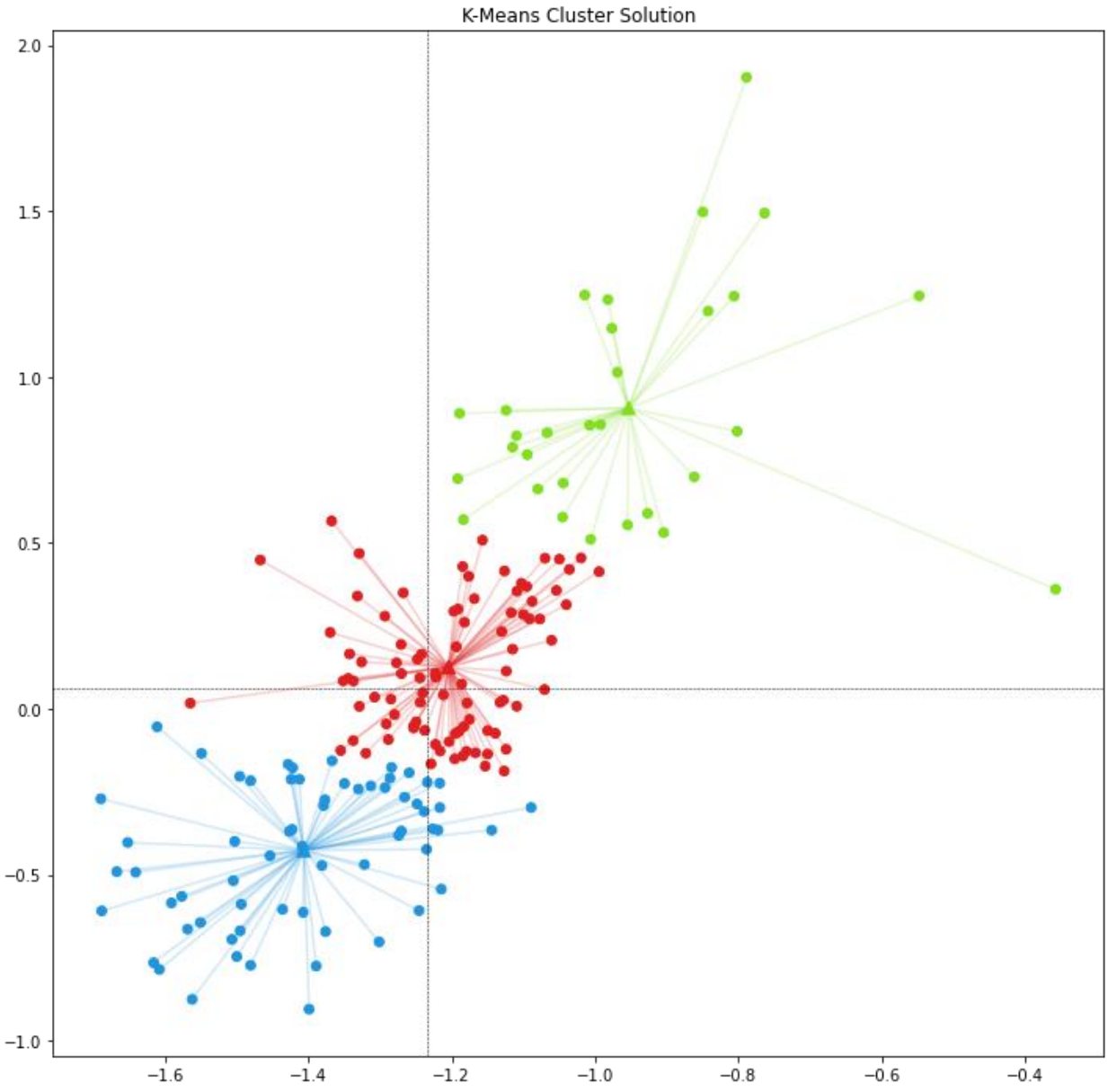
```
# Find the Optimal K Clusters
distortions = []
K = range(1,10)
for k in K:
    kmeanModel =
KMeans(n_clusters=k)
    kmeanModel.fit(glrm_df)

distortions.append(kmeanModel.ine
rtia_)

plt.figure(figsize=(16,8))
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method
showing the optimal k')
plt.show()
```

The code above iterates over the data set to find the optimal number of clusters using the “elbow” method. In this case, it appears that we have 2 or perhaps 3 optimal clusters.



From here we can complete our K-Means analysis and visualize the results. The code below runs the K-Means algorithm on the Archetypes from the GLRM and produces a Scatterplot with spikes to better illustrate the cluster pattern.

```
# Identify and Plot K-Means
kmeanModel = KMeans(n_clusters=3)
kmeanModel.fit(glrm_df)
glrm_df['k_means']=kmeanModel.predict(glrm_df)

colors = ['#DF2020', '#81DF20', '#2095DF']
glrm_df['c'] = glrm_df['k_means'].map({0:colors[0], 1:colors[1],
2:colors[2]})

# Get Centroids
centroids = kmeanModel.cluster_centers_
cen_x = [i[0] for i in centroids]
cen_y = [i[1] for i in centroids]
# add to df
glrm_df['cen_x'] = glrm_df['k_means'].map({0:cen_x[0], 1:cen_x[1],
2:cen_x[2]})
glrm_df['cen_y'] = glrm_df['k_means'].map({0:cen_y[0], 1:cen_y[1],
2:cen_y[2]})

plt.figure(figsize=(12,12))
plt.scatter(glrm_df[0], glrm_df[1], c=glrm_df['c'])
plt.scatter(cen_x, cen_y, marker='^', c=colors, s=70)

# plot reference lines
plt.axhline(y=glrm_df[1].mean(), color='black', linestyle='--', lw=0.5)
plt.axvline(x=glrm_df[0].mean(), color='black', linestyle='--', lw=0.5)

# plot lines
for idx, val in glrm_df.iterrows():
    x = [val[0], val.cen_x,]
    y = [val[1], val.cen_y]
    plt.plot(x, y, c=val.c, alpha=0.2)
plt.title('K-Means Cluster Solution')
```



Conclusion

This paper has briefly demonstrated how to use the H20 API in Python to reduce a complex, mixed-type dataset to k dimensions for subsequent analysis. The GLRM algorithm is an alternative to Principal Component Analysis, which requires all data to be scale-level data. Using this technique we are able to incorporate additional features that maybe missing data or categorical in nature. There clear benefits to this approach in Machine Learning for memory, speed, feature engineering and imputation.