Interpeting Outliers from Clustering Algorithms

render("interpret_outliers.md", "pdf_document")

By Gaurav Sood

Assume that the data from the dominant data generating process are structured so that they occupy a few small portions of a high-dimensional space. Say we use a hard partition clustering algorithm to learn the structure of the data. And say that it does—learn the structure. Anything that lies outside the few narrow pockets of high-dimensional space is an 'outlier,' improbable (even impossible) given the dominant data generating process. (These 'outliers' may be generated by a small malicious data generating process.) Even points on the fringes of the narrow pockets are suspicious. If so, one reasonable measure of suspiciousness of a point is its distance from the centroid of the cluster to which it is assigned; the further the point from the centroid, the more suspicious it is. (The distance can be some multivariate distance, or proportion of points assigned to the cluster that are further away from the cluster centroid than the point whose score we are tallying.)

How can we interpret an outlier (score)? Tautological explanations—it is improbable given the dominant data generating process—aside.

Simply providing distance to the centroid or such metrics doesn't generally give enough context. For high-dimensional vectors, for obvious reasons, providing distance on each feature isn't reasonable either. A better approach involves some feature selection. This can be done in various ways, all of which take the same general form. Find distance to centroid on features on which the points assigned to the cluster have the least variation. Or, on the features that discriminate the cluster from other clusters the best. Or, features that predict distance from the cluster centroid the best. Limit the features arbitrarily to a small set. Next, on this limited feature set, calculate cluster means and standard deviations, and give standardized distance (for categorical variable, just provide) to the centroid on selected feature set.

Workflow

Input

1. Output table of the clustering algorithm merged back with the raw data:
    - The output table must have a cluster label and an 'outlier score' for each point.
    - User must indicate which column has the cluster label, and which the outlier score. Rest of the columns are treated by default as covariates, though user can specify which columns they want to ignore.
2. Which heuristic does the analyst wants to use?
3. How many resulting dimensions does the analyst want? (Takes values 1 to total number of covariate columns.)

Formal Input

1. oulier = column name for outlier score. Has to be numeric. If not numeric, throw a warning and cast as one.
2. cluster = cluster label column. Has to be string. If not string, throw a warning and cast as one.
3. n_out = No. of resulting dimensions. Default = 5. Max = Total number of columns. Throw an error if out of range.
4. heuristic = 'discriminating', 'pred_outlier', 'least_variance'. Throw an error if not one of three.

Example Usage

```
output= merged_table.selectfeatures(
                     outlier="outlier",
                     cluster="cluster", n_out=5,
                     heuristic = "discriminating")

merged_table.interpret(selected_features=output, normed=TRUE)
```

Most Discriminating Traits

```
# Pseudo code

# Data
cluster_label = merged_table[, "cluster"]
covariates    = merged_table[, -c("cluster", "outlier")]

# Unique cluster_labels
unique_clusters = unique(cluster_label)

# Initialize Results Table
pick_n # A number of cluster labels * number of resulting dimensions (n_out) table

for (i in unique_clusters)
{
    y = cluster_label==i # get a dummy (1/0 vector)

    # Logistic with L1 regularizer
    mod = L1_logistic_regression(y ~ covariates)

    # reverse sort model coefficients (highest to lowest), get column names
    sort_mod  = sort(mod_coefficients)
    pick_n[i] = sort_mod[1:n_out]
}
```

Best Predictors of Outlier Score

```
# Pseudo code
# Split dataset by cluster label
# Within each cluster label, predict outlier score

# Data
outlier       = merged_table[, "outlier"]
cluster_label = merged_table[, "cluster"]
covariates    = merged_table[, -c("cluster", "outlier")]

# Unique cluster_labels
unique_clusters = unique(cluster_label)

# Initialize Results Table
pick_n # A number of cluster labels * number of resulting dimensions (n_out) table

for (i in unique_clusters)
```

```
{
    # only data from cluster_label i
    subset_table = subset(mergedtable, cluster_label==i)

    # Within each subset_table
    # Linear regression with L1 regularizer
    mod = with(subset_table, L1_regression(outlier ~ covariates))

    # reverse sort model coefficients (highest to lowest), get column names
    sort_mod  = sort(mod)
    pick_n[i] = sort_mod[1:n_out]
}
```

Dimensions With Least Variation Within A Cluster

```
# Pseudo code
# Split dataset by cluster label
# Within each cluster label, find dimensions with least variation

# Data
cluster_label = merged_table[, "cluster"]
covariates    = merged_table[, -c("cluster", "outlier")]

# Unique cluster_labels
unique_clusters = unique(cluster_label)

# Initialize Results Table
pick_n # A number of cluster labels * number of resulting dimensions (n_out) table

# Normalize all dimensions
covariates = normalize(covariates) # Subtract mean and divide by standard deviation

for (i in unique_clusters)
{

    # only data from cluster_label i
    subset_table = subset(mergedtable, cluster_label==i)

    # Within each subset_table
    var = var(covariates)
    sort_var  = sort(var) # lowest to highest
    pick_n[i] = sort_var[1:n_out]
}
```

Interpret

```
# Pseudo Code

output= merged_table.selectfeatures(
                     outlier="outlier",
                     cluster="cluster", n_out=5,
                     heuristic = "discriminating")
```

```
# res = resulting data table with # of rows = number of points and
# number of cols = number of resulting dims in output
# normed_res = same dims as res

j = 1
for (i in unique_clusters)
{

    # only data from cluster_label i and only selected columns
    subset_table = subset(mergedtable, rows=cluster_label==i, columns=output[j,])
    j = j + 1

    # Within each subset_table
    # Calculate cluster means and standard deviations

    cluster_means = colMeans(subset_table[,output[j,]])
    cluster_std   = col_std(subset_table[,output[j,]])

    # Subtract cluster mean from outlier value on each of the variables
    res[j] = list(var1 = dis1, var2 = dis2) ....

    # Subtract cluster mean from outlier value on each of the variables
    # and divide by standard deviation
    # if normed==TRUE
    res_norm[j] = list(var1 = stnd_dis1, var2 = stnd_dis2) ....

}

# append all res/normed_res
```