Chapter 4 - Proximity-Based Outlier Detection

1 Introduction

There are three proximity-based approaches: clustering, distance, and density. Clustering partitions the data points while density partitions the data space. Distance-based methods are the most granular, but most compute intensive.

2 Clusters and Outliers: The Complementary Relationship

Conventional wisdom says every point is a member of a cluster or it's an outlier. One outlier score is to measure the Mahalanobis distance from a point to its cluster centroid. We can then do extreme value analysis on the Mahalanobis distances. The negative logarithm of the fraction of points in the nearest cluster can also be a useful outlier score (especially for histogram methods). Since the output clusters depend on random initialization, we run several copies of the algorithm and ensemble.

The Mahalanobis distance is good for elliptical clusters, but clusters could be arbitrarily shaped. You may want to run nonlinear PCA before clustering. However, this is good for finding global embeddings, not local embeddings. To find local embeddings, use sparsification (assume a point's similarity to points not inside its k nearest neighbors is 0). We can also do local normalization, where we let ρ_i be the sum of similarities of the i^{th} row of S and replace s_{ij} with $s_{ij}/\sqrt{\rho_i \rho_j}$. We then keep the top-$m$ eigenvalue dimensions and project our points down to that space. Then, standardize to zero mean and unit variance and do the clustering. To compute outlier scores, we compute distance to nearest centroid, but we use all nonzero eigenvalue dimensions instead of m (remember, the small eigenvalues matter for outlier detection).

Clustering is fast compared to distance-based methods, but not as granular. The outlier scores can also vary based on random initialization, so make sure to ensemble.

3 Distance-Based Outlier Analysis

Distance based methods are differentiating between outliers and noise, while clustering tries to ignore noise. A naive distance-based method requires $O(N^2)$ time, so we need to speed it up. One simple approach is to use the k nearest neighbor distance as the outlier score. This is highly dependent on k, so we can make it more robust by instead averaging the 1 to k nearest neighbor distances and using that as the score (we can also use the harmonic mean rather than arithmetic mean, but that it's not commonly used).

Indexing structures work well for low-dimensional data, but struggle in high dimensions. A better option is to sample some points and only consider these points as candidate neighbors. We can repeat this many times and average the results.

If we are ok with binary labels instead of scores, we can seek the points whose k nearest neighbor distance are in the top r of all k nearest neighbor distances. You could also say that outliers are points where the k nearest neighbor distance exceeds β. This requires us to pick just one parameter and enables some pruning.

In cell based pruning, we divide the data space into cells of width $\frac{\beta}{2\sqrt{2}}$. A cell's L1 neighbors are reachable by crossing one cell boundary and the L2 neighbors are reachable by crossing 2 or 3. This enables two rules: (1) if over k points are in a cell and its L1 neighbors, none of them are outliers (2) if at most k data points are in a cell and its L1 and L2 neighbors, all the points in the cell are outliers. These rules let us quickly label many points. For the remaining points, we only need to compute distances to points in the L2 neighbors of their cells (L1 neighbors are all within distance β). This for dimensionality 2, but you can easily extend it to high dimensional data.

Sampling based pruning should be your first line of attack because it is fast and flexible. First, we compute pairwise distances between sample points and the entire dataset. We then take the score (i.e. k nearest neighbor distance) of the top r outlier in the sample points - this score is a lower bound on the true top r outlier scores. For all points in the dataset, we know an upper bound on their score (the distance to k nearest neighbor in-sample point). If this score is less than the lower bound, we can throw out the point - it's not an outlier. For the remaining points, we compute their k nearest neighbors explicitly. We can further speed this up by tightening the upper bound as we scan the remaining points and terminating when it crosses the lower bound.

In index based pruning, we can combine minimum bounding rectangles with branch-and-bound and an R^* tree. Using partitioning techniques can further speed this up.

We can use other distance metrics than Euclidean. Locality sensitive metrics tend to be useful (e.g. a pair of Caucasians are more similar if they are in Asia than if they are in Europe). One measure of locality is the number of shared k nearest neighbors (this is more computationally expensive and adds a hyperparameter though). Using the Mahalanobis distance is also a good idea (or Mahalanobis from cluster centroid). Another option is to augment the dataset with points sampled uniformly at random from the data space, fit a random forest to separate the real data from synthetic data, and take average path length between pairs of points in the random forest trees as our locality measure.

q is a reverse k nearest neighbor of p iff q is among the k nearest neighbors of p. The Outlier Detection using In-Degree Number (ODIN) algorithm considers outliers to be points that are the reverse k nearest neighbors for fewer than some threshold number of points. To speed this up, use sampling and ensembling.

We also might want to explain why points are outliers (e.g. outlier in 1D space of feature1, outlier in 2D space of feature1 feature2).

4 Density-Based Outliers

Datasets send to have local structures with varying data densities, so density based outlier detection can be effective.

Let's discuss the Local Outlier Factor (LOF) algorithm. Let $D^k(\bar{X})$ be the distance to $\bar{X}$'s k nearest neighbor and let $L^k(\bar{X})$ be the set of points within this distance. Define reachability (notice that it's not symmetric) as $R_k(\bar{X}, \bar{Y}) = \max(dist(\bar{X}, \bar{Y}), D^k(\bar{Y}))$. Define average reachability as $AR_k(\bar{X}) = \text{MEAN}_{\bar{Y} \in L_k(\bar{X})} R_x(\bar{X}, \bar{Y})$. The LOF is then $LOF_k(\bar{X}) = \text{MEAN}_{\bar{Y} \in L_k(\bar{X})} \frac{AR_k(\bar{X})}{AR_k(\bar{Y})} = \frac{AR_k(\bar{X})}{\text{HMEAN}_{\bar{Y} \in L_k(\bar{X})} AR_k(\bar{Y})}$. In a homogeneous cluster, LOF is around 1. For outliers, it is higher. If you have duplicate or very closeby points, you should avoid zero harmonic means by removing duplicates or adding α to the numerator and denominator of the harmonic mean expression for regularization. Unless you know a good value of k, prefer the simple k nearest neighbors detector over LOF.

Let's discuss the LOCI algorithm. Let $M(\bar{X}, \epsilon)$ be the number of points within distance ϵ of $\bar{X}$. The average density, for $\delta > \epsilon$, is $AM(\bar{X}, \epsilon, \delta) = \text{MEAN}_{\bar{Y}: dist(\bar{X}, \bar{Y}) \leq \delta} M(\bar{Y}, \epsilon)$. The multi-granularity deviation factor is $MDEF(\bar{X}, \epsilon, \delta) = 1 - \frac{M(\bar{X}, \epsilon)}{AM(\bar{X}, \epsilon, \delta)}$. Now, we define $\sigma(\bar{X}, \epsilon, \delta) = \frac{\text{STD}_{\bar{Y}: dist(\bar{X}, \bar{Y}) \leq \delta} M(\bar{Y}, \epsilon)}{AM(\bar{X}, \epsilon, \delta)}$. We usually pick $2\epsilon = \delta$ for fast computation. Then, we say a point is an outlier if $MDEF$ exceeds $k\sigma(\bar{X}, \epsilon, \delta)$ (we usually pick $k = 3$). By breaking the data space into cells and counting the number of points per cell, we can easily approximate the neighbor counts. If we plot $M(\bar{X}, \delta/2)$ vs. δ and $AM(\bar{X}, \delta/2, \delta)$ vs. *delta*,

we get LOCI plots. The more that the first plot lies below the second, the more of an outlier the point is.

In a histogram based method, we split the data space into bins and compute the number of points in each bin. You can sample the points and build the histogram if you have too many points. Then, for each point, we look up the frequency in its bin f_i, identify whether it was an in-sample point I_i, and compute the outlier score $\log(f_i - I_i + \alpha)$ where α does regularization. We can use a Student's t distribution to turn the scores into binary labels. For multi-dimensional data, you can apply this method one dimension at a time or make multi-dimensional bins. To avoid picking a single bin size, you can just try many bin sizes and average the results. They suffer from the curse of dimensionality though because high dimensional histograms are very sparse.

In kernel density estimation (KDE), we define the density as $\hat{f}(\bar{X}) = \frac{1}{N}\sum_{i=1}^{N} K'_h(\bar{X} - \bar{X}_i)$. The Gaussian kernel is popular here: $K'_h(\bar{X} - \bar{X}_i) = (\frac{1}{h\sqrt{2\pi}})^d \exp - \frac{||\bar{X} - \bar{X}_i||^2}{2h^2}$. To pick h, use the Silverman approximation rule $h = 1.06\hat{\sigma}N^{-1/5}$ where $\hat{\sigma}$ is sample variance. You can use $\hat{f}(\bar{X})$ as your outlier score or feed it to extreme value analysis with a Student's t distribution. KDE struggles when the dimensionality d is large. To make KDE more locality sensitive, you can compute the bandwidth h by using the local neighborhood or by averaging the kernel densities of a point's neighbors.

For high dimensional data, ensemble KDE or histograms with rotated bagging, which reduces data dimensionality from $O(d)$ to $O(\sqrt{d})$. Randomly generate a subspace of dimension $2 + \lceil\sqrt{d}/2\rceil$, project the points, and run your density algorith. Do this many times and average the results.

5 Limitations of Proximity-Based Detection

These techniques work well in ensembles. Make sure to consider local structures. Be as granular as you can without making it computationally infeasible (but remember pruning struggles in high dimensionality). High dimensionality means we have a lot of noise features, which can make the distance calculations hard.

6 Conclusions and Summary

Clustering methods can be combined with other proximity based methods. Histogram and KDE don't work well on their own, but can help an ensemble.