Chapter 1 - An Introduction to Outlier Analysis

1 Introduction

An outlier is a point that is so different from the other points that we suspect it is created by a different generating process. The output of outlier analysis is a score or binary label for each point indicating whether the point is an outlier or inlier. Typically, data falls on a range between inlier to noise to outlier.

We can also detect outliers in time series, spatial data, and graphs. Each data point can have categorical or numerical information.

2 The Data Model is Everything

Typically, we create a model for the normal data and the outlier score is how much the point differs from the normal model.

For example, suppose we have one-dimensional points $X_1, ..., X_N$. We can use a Gaussian model where we compute mean μ and variance σ and define the outlier scores as $Z_i = \frac{|X_i - \mu|}{\sigma}$. If N is small, we can use a Student's t distribution instead of a Gaussian. This Z value test fails if the underlying data is not Gaussian or if the outlier is not an extreme value, but is strange compared to its neighbors (contextual outlier).

You can think of outlier detection as binary classification where outliers constitute one class and normal points constitute the other. We can repurpose common machine learning models (e.g. SVM, Neural Net, Random Forest) for outlier detection. In addition to parametric models, we have instance based models like k-nearest neighbors and Local Outlier Factor (LOF), which tend to be more popular.

In unsupervised outlier detection, we don't use a training/validation/test set. This means tuning hyperparameters is hard, so try to use models with few or no hyperparameters.

3 The Basic Outlier Detection Models

We often want interpretable models, which provide hints about why a point is considered an outlier.

To measure the non-uniformity of a set of 1-dimensional points $x_1, ..., x_N$ is to compute mean μ, variance σ, z-scores $z_i = \frac{x_i - \mu}{\sigma}$ and combine them together to get the Kurtosis measure:

$$K(z_1, ..., z_N) = \frac{\sum_{i=1}^{N} z_i^4}{N} \tag{1}$$

To get this to work for multidimensional data, you can compute the Kurtosis measure on the Mahalanobis distances between points and their centroid. You can use the Kurtosis measure to identify features that could be good for outlier detection.

Another way to identify good features is to predict one feature given the others (i.e. make a supervised regression or classification model). If the feature is uncorrelated with the others, drop it.

Extreme Value Analysis can find outliers by looking for points that have very high or low values of a feature. In a multidimensional case, you can consider one feature at a time or do the analysis on the Mahalanobis distances to the centroid.

You can fit a probability distribution (e.g. Gaussian Mixture Model trained with Expectation Maximization) and use the PDF values as outlier scores. You can then run Extreme Value Analysis on the PDF values.

In linear methods, we project the data onto a lower dimensional hyperplane and use the distance from point to hyperplane as outlier score. This, and other dimensionality reduction techniques are not very interpretable.

Spectral methods, which are related to matrix factorization, are good for graph data.

Proximity methods are the most popular way to do outlier detection. They are clustering (segments data points), density (segments data space), or nearest-neighbor based. The distance to the kth nearest neighbor can be treated as the outlier score. k nearest neighbors is slow ($O(N^2)$), but we can prune out many points if we just need binary labels. In clustering, we cluster the points and take each point's distance to nearest cluster as outlier score (we run many copies of the algorithm and average the scores to overcome bad random initializations). Density methods like histograms are interpretable.

Information theoretic methods consider outliers to be points that increase the minimum code length required to describe the points. Frequent pattern mining, histograms, probabilistic models, and PCA can be used as information theoretic models.

For high dimensional data, it's hard to compute distances. So, we do subspace outlier detection, where we look for subspaces where anomalous behavior is exhibited. The subspaces are typically interpretable (e.g. Age ¡ 20 and Disease = 1).

4 Outlier Ensembles

We can combine many algorithms into an ensemble. A sequential ensemble (e.g. boosting) runs the algorithms one after the other where they depend on the previous algorithm. An independent ensemble runs the algorithms in parallel and averages their results.

5 The Basic Data Types for Analysis

Data points can have categorical, numerical, or mixed attributes. To deal with categorical variables, we can one-hot encode them, use Latent Semantic Analysis (LSA) (my note - you could also train an entity embedding).

Data points can be related to each other through a time series, graph, or spatial distribution. A point that is an outlier because of its neighbors is a contextual (or conditional) outlier. A group of points that are strange are called a collective anomaly.

For a time series, anomalies can be spikes, repetitions, changing frequencies, etc. Temporal data typically includes a trend (or concept drift), but this is slow and expected, so we should make sure not to let this fool us. Time series may be continuous vectors or be a sequence of events. For the latter, Hidden Markov Models work well. The former can use autoregressive techniques.

Data can also have a spatial distribution (e.g. temperature or pressure field).

Data can take the form a graph. A node outlier is a data point that is strange. An edge outlier is a relationship between points that is strange. We can also have a temporal component to graphs as they evolve over time.

6 Supervised Outlier Detection

If we have known outliers, we can use supervised learning. This is harder than regular supervised learning because the classes are heavily unbalanced. Some challenges are the Positive-Unlabled Classification problem (i.e. we have known outliers and a set of points whose outlier status is unknown), limited datasets (e.g. we do not know all the kinds of outliers, so we have a subset of them), and active learning (we select possible outlier points that we can send to a labeler who tells us for sure whether they are outliers or not).

7 Outlier Evaluation Techniques

If you have labeled examples, use that. If you have a specific task where you are using the model, use task specific metrics. You can also use internal validity metrics (e.g. mean squared radius of cluster), but it's easy to overfit if you rely on this.

If you have labeled points, use precision and recall as your metrics. Given a threshold t on outlier scores we can compute their predicted binary labels $S(t)$ and compare against the true labels G. We then have

$$\text{Precision} = \frac{|S(t) \cap G|}{|S(t)|} \tag{2}$$

$$\text{Recall} = \frac{|S(t) \cap G|}{|G|} \tag{3}$$

By varying t, you can plot a precision-recall curve. Alternatively, you can plot the true positive rate vs. false positive rate to get a Receiver Operating Characteristic (ROC) curve, which is easier to interpret.

$$TPR(t) = \frac{|S(t) \cap G|}{|G|} \tag{4}$$

$$FPR(t) = \frac{|S(t) - G|}{|D - G|} \tag{5}$$

The lift above the line $TPR = FPR$ indicates the superiority to a random method. If you want a single number, you can compute the area under the ROC curve (ROC AUC). The ROC AUC is the probability that a randomly selected outlier-inlier pair is ranked correctly. A random method gets ROC AUC of 0.5.

Usually, the initial part of the ROC curve matters more than the later parts. For example, it may not matter if a point is ranked 501 or 601, but it matters if it is 1 or 101. Use the normalized discounted cumulative gain in this case.

Avoid tuning hyperparameters to maximize your ROC AUC. So, you should predefine a set of hyperparameters and compute the median ROC AUC. You might also compute the variance to ensure that your algorithm is robust to different hyperparameter choices.

8 Conclusions and Summary

Outlier analysis has many applications and algorithms.

Chapter 2 - Probabilistic and Statistical Models for Outlier Detection

1 Introduction

Extreme value analysis is good for univariate data. Generative models are also popular.

2 Statistical Methods for Extreme-Value Analysis

The Markov Inequality says that, for nonnegative random variable X, we have for any constant α where $E[X] < \alpha$ that

$$P(X > \alpha) \leq E[X]/\alpha \tag{1}$$

The Chebyshev Inequality says that, for random variable X, we have for any constant α that

$$P(|X - E[X]| > \alpha) \leq Var(X)/\alpha^2 \tag{2}$$

The Chernoff Bound says that if X is the sum of N independent Bernoulli random variables (where the i^{th} variable has parameter p_i) then for any $\delta \in (0, 1)$ we have

$$P(X < (1 - \delta)E[X]) < \exp(-\frac{1}{2}E[X]\delta^2) \tag{3}$$

and for $\delta \in (0, 2e - 1)$ we have

$$P(X > (1 + \delta)E[X]) < \exp(-\frac{1}{4}E[X]\delta^2) \tag{4}$$

The Hoeffding Inequality says that if X is the sum of N independent random variables (where the i^{th} variable lies between $[l_i, u_i]$), that we have, for any $\theta > 0$

$$P(X - E[X] > \theta) \leq \exp(-\frac{2\theta^2}{\sum_{i=1}^{N}(u_i - l_i)^2}) \tag{5}$$

$$P(E[X] - X > \theta) \leq \exp(-\frac{2\theta^2}{\sum_{i=1}^{N}(u_i - l_i)^2}) \tag{6}$$

The Central Limit Theorem says the sum of N i.i.d random variables with mean μ and standard deviation σ converges to a normal distribution with mean $N\mu$ and standard deviation $\sigma\sqrt{N}$.

For univariate data, we can fit a Gaussian distribution and compute the Z-scores and PDF values. If the dataset is small (under 1000 examples), we fit a Student's t distribution where the degrees of freedom are $\nu = N - 1$.

We can also make a box-and-whiskers plot. The box is drawn between the first and third quartiles. The lower whisker is 1.5 times the interquartlie range (IQR) below the bottom of the box and the upper whisker is 1.5 times the IQR above the top of the box. Points outside the whiskers are plotted as points.

3 Extreme-Value Analysis in Multivariate Data

Extreme value analysis finds outliers at the boundaries of the data, not in the middle.

Depth based methods find a convex hull around the data and assign the points on the hull a depth of 1. We then remove those points and repeat to find points at depth 2. We do this r times and consider those points outliers. The convex hull algorithm scales exponentially with the dimensionality of the data.

In deviation based methods, we identify a subset of the points whose removal reduces dataset variance the most. There are hueristics for picking the subsets.

In angle based methods, we use the intuition that outlier points should be able to close the rest of the data by casting out two rays with a small angle. So, given a test point $\bar{X}$ and random points $\bar{Y}$ and $\bar{Z}$, we can compute the angle as

$$W \cos(\bar{Y} - \bar{X}, \bar{Z} - \bar{X}) = \frac{(\bar{Y} - \bar{X})^T(\bar{Z} - \bar{X})}{||\bar{Y} - \bar{X}||_2^2||\bar{Z} - \bar{X}||_2^2} \tag{7}$$

If we vary $\bar{Y}$ and $\bar{Z}$ we can compute the angle based outlier factor

$$ABOF(\bar{X}) = Var_{Y,Z \in D} W \cos(\bar{Y} - \bar{X}, \bar{Z} - \bar{X}) \tag{8}$$

A naive implementation takes $O(N^3)$ time, but we can speed this up by only considering the k nearest neighbors of $\bar{X}$ and ignoring it altogether if its first ABOF is large. Note that this method suffers from the curse of dimensionality because of the cosine.

We could also fit a Gaussian with mean μ and covariance Σ and compute the PDF (the term in the exponential is $-1/2$ times the squared Mahalanobis distance).

$$f(\bar{X}) = \frac{1}{\sqrt{\Sigma}(2\pi)^{d/2}} \exp(-\frac{1}{2}(\bar{X} - \mu)\Sigma^{-1}(\bar{X} - \mu)^T) \tag{9}$$

If Σ is not invertible, we can replace it with $\Sigma + \lambda I$ for small λ (this is regularization). This nice thing about the above Mahalanobis technique is that is it is parameter-free, accounts for variance in each dimension, and runs in linear time (to compute mean and covariance).

4 Probabilistic Mixture Modeling for Outlier Analysis

We can model the data with a mixture of k distributions. That is, the probability density of $\bar{X}_j$ is

$$f^{point}(\bar{X}_j | \mathcal{M}) = \sum_{i=1}^{k} \alpha_i f_i(\bar{X}_j) \tag{10}$$

where α_i and f_i are the mixing proportion and PDF of the i^{th} distribution, respectively. We can infer these parameters (call them Θ) by maximizing the log-likelihood of the data using the Expectation Maximization algorithm. This is an iterative algorithm where we first fix the parameters and compute $P(\bar{X}_j|\mathcal{G}_r, \Theta)$ for all (point, distribution) pairs $(\bar{X}_j, \mathcal{G}_r)$. This is the E-step. Then we fix the probabilities above and infer the parameters again. This is the M-step.

The E-step computes:

$$P(\mathcal{G}_r|\bar{X}_j, \Theta) = \frac{\alpha_r f^{r,\Theta}(\bar{X}_j)}{\sum_{i=1}^{k} \alpha_i f^{i,\Theta}(\bar{X}_j)} \tag{11}$$

The M-step computes (where the α_i expression uses Laplacing smoothing to push the estimate towards $1/k$):

$$\alpha_i = \frac{1 + \sum_{j=1}^{N} P(\mathcal{G}_r|\bar{X}_j, \Theta)}{k + N} \tag{12}$$

To estimate f_r, we treat $P(\mathcal{G}_r|\bar{X}_j, \Theta)$ as the weight of the point in component r and compute the maximum likelihood estimate of the model parameters.

Mixture models are a kind of stochastic clustering or soft clustering.

If you use a single component, you get a soft version of Principal Components Analysis (PCA). It becomes even more powerful when you combine it with kernel methods.

If you have a score for each point computed by another outlier detection algorithm, then you can use a mixture of an exponential distribution (for outlier points) and Gaussian distribution (for inlier points). Then you can compute the posterior probability that each point belongs to the outlier component. Thus, we have turned scores into probabilities.

5 Limitations of Probabilistic Modeling

Parametric methods like mixture models can fail to fit the data if it does not obey the assumed distributions. If you have too many components in your mixture, you can overfit. Note that the E-step and M-step each take $O(Nk)$ time. Mixture models are not easily interpretable.

6 Conclusions and Summary

Extreme value analysis is good for low dimensional data. Mixture models are powerful for fitting data or converting scores into probabilities.

Chapter 3 - Linear Models for Outlier Detection

1 Introduction

We can predict one variable (feature) from the others using linear regression or project the data to a lower dimensional linear subspace with Principal Components Analysis (PCA).

2 Linear Regression Models

$$y = \sum_{i=1}^{d} w_i x_i + w_{d+1} \tag{1}$$

This is a linear model. We learn the weights by minimizing the sum of squared errors. Outliers are points whose squared error is large. If you put the data into design matrix D, the weights into $\bar{W}$, and the regression targets into $\bar{y}$, you get $\bar{y} \approx D\bar{W}^T$. Minimizing $||D\bar{W}^T - \bar{y}||^2$ gives $\bar{W}^T = (D^T D)^{-1} D^T \bar{y}$. If $D^T D$ is not invertible, we can regularize with parameter α and minimize $||D\bar{W}^T - \bar{y}||^2 + \alpha||\bar{W}||^2$ to get $\bar{W}^T = (D^T D + \alpha I)^{-1} D^T \bar{y}$.

Ironically, the present of outliers can mess up the inference of the parameters, so it is typical to randomly sample some points and fit those. We do this many times and create an ensemble. For a given point, we average the squared error over each ensemble component to get its outlier score. We can also use extreme value analysis on these squared errors.

To use linear regression in an unsupervised setting, we can consider each feature one at a time, make it a dependent variable, and fit a model. Then, we can average over all the models to get the outlier score for each point.

3 Principal Components Analysis

Linear regression projects d-dimensional data to a $(d-1)$-dimensional hyperplane. PCA lets us project to a k-dimensional hyperplane.

The covariance is $\Sigma = \frac{D^T D}{N}$ and can be diagonalized as $\Sigma = P \triangle P^T$ where $\triangle$ is the diagonal eigenvalue matrix (the eigenvalue measures the variance on each dimension in the hyperplane) and P is the eigenvector matrix. We can project the data to the eigenvector space with $D' = DP$. When we project, we should look at the dimensions corresponding to the small eigenvalues because these are the dimensions with maximum variance. Any data point that is extreme on these dimensions is an outlier. More generally, we can remove the k largest eigenvalue-eigenvector dimensions and measure the reconstruction cost (the points with highest reconstruction cost are outliers). Alternatively, we can compute the outlier score in a soft way (where λ_j and $\bar{e}_j$ are the eigenvalue, eigenvector pair).

$$Score(\bar{X}) = \sum_{j=1}^{d} \frac{|(\bar{X} - \bar{\mu})^T \bar{e}_j|^2}{\lambda_j} \tag{2}$$

PCA can handle a few outliers. If there are many outliers, you can run PCA to identify the outliers, remove them, run PCA again, and repeat. Before you use PCA, make sure to standardize D so each feature has zero mean and unit variance. If there is not much data, we can regularize by replacing Σ with $\Sigma + \alpha I$.

How do you pick the number of eigenvalues, k? You can use a t test and find outliers. These are typically the large eigenvalues.

To make PCA nonlinear, we use the kernel trick. First, we note that we can do PCA with similarity matrix $S = DD^T$ instead of Σ. Then, we replace S with a similarity matrix where $S_{ij} = K(\bar{X}_i, \bar{X}_j)$ (we need to make sure to use a valid kernel function). We then diagonalize as follows: $S = Q\Lambda^2 Q^T$ and pick k eigenvectors. Typically we pick k to get all nonzero eigenvalues because we are looking for outlier behavior, not aggregate trends. This gives us the embedding D'. We divide each column of D' by its standard deviation and report its squared distance from the centroid of D' as its outlier score.

Popular kernels are the linear kernel $\bar{X}_i^T \bar{X}_j$, Gaussian Radial Basis Function (RBF) $\exp\left(-\frac{||\bar{X}_i - \bar{X}_j||^2}{\sigma^2}\right)$, polynomial kernel $(\bar{X}_i^T \bar{X}_j + c)^h$, and sigmoid kernel $\tanh\left(\kappa \bar{X}_i^T \bar{X}_j - \delta\right)$. The RBF kernel is the most popular. For the RBF kernel, use the median pairwise distance between points (or three times the pairwise distance if you have fewer than 1000 data points). To speed up kernel computation, you can simply set similarity to 0 if points are not k nearest neighbors. There are kernel functions for complex data types like strings, time series, and graphs.

The similarity matrix is $N \times N$, which is huge. So, we sample s points and use the technique described above to get $Q_k \Lambda_k$. We then compute the similarity of other other points to the sample points to get $(N - s) \times s$ matrix S_o. The embedding is then $V_k = S_o Q_k \Lambda_k^{-1}$. We stack $Q_k \Lambda_k$ vertically on top of $S_o Q_k \Lambda_k^{-1}$ to get the $N \times k$ embedding E. We standardize each column to zero mean and unit variance (this means the mean row vector is the zero vector). We then compute the squared distance of each row to the mean row vector (0) as the outlier score for the data point at that row.

4 One-Class Support Vector Machines

We assume that the origin in the kernel transformed space belongs to the outlier class. Our decision boundary is thus: $\bar{W}^T \Phi(\bar{X}) - b = 0$. We then minimize $J = \frac{1}{2}||\bar{W}||^2 + \frac{C}{N} \sum_{i=1}^{N} \max\left(b - \bar{W}^T \Phi(\bar{X}_i), 0\right) - b$.

To support kernel functions, we rewrite $\bar{W}^T \Phi(\bar{Y}) - b = \sum_{i=1}^{N} \alpha_i K(\bar{Y}, \bar{X}_i) - b$. We then minimize (where S is the similarity matrix) $\frac{1}{2} \bar{\alpha}^T S \bar{\alpha}$ subject to $0 \leq \alpha_i \leq \frac{C}{N}$ for $i \in \{1...N\}$ and $\sum_{i=1}^{N} \alpha_i = 1$. Solving the above with quadratic programming yields $\bar{\alpha}$ and we compute $b = \sum_{i=1}^{N} \alpha_i K(\bar{Y}, \bar{X}_i)$ where $\bar{Y}$ is a support vector (this is a point where $0 < \alpha_j < C/N$). The score of a point is then $Score(\bar{X}) = \sum_{i=1}^{N} \alpha_i K(\bar{X}, \bar{X}_i) - b$.

We can optimize the the quadratic programming problem with gradient descent ($\bar{\alpha} \leftarrow \bar{\alpha} - \eta S \bar{\alpha}$, constrain each α_i to $[0, C/N]$, and scale $\bar{\alpha}$ so that $\sum_{i=1}^{N} \alpha_i = 1$).

One-class SVM is bad because of the origin assumption and scaling difficulties. You can mitigate the former with binary features and the latter by sampling a set of points and fitting them instead of fitting the whole dataset (you can ensemble this). Use kernel PCA instead.

5 A Matrix Factorization View of Linear Models

From PCA, we get $D' = DP_k$. If we do PCA the other way, we get $DP_k \approx Q_k \Lambda_k$ and thus $D \approx Q_k \Lambda_k P_k^T$. Combining the diagonal matrix Λ_k into one of the other factors means we have decomposed $D \approx UV^T$ where U and V are low rank. Another way to find the matrices is by minimizing the Frobenius norm $||D - UV^T||$ subject to the columns of U (and V) having mutually orthogonal (orthonormal) columns. We can remove the constraints if we want. You can also add different constraints.

What do we do if the data is incomplete? Assume we have N users and d movies and we represent user-movie ratings with $N \times d$ matrix D, where the element at (i, j) is x_{ij}. Let $H = \{(i, j) : x_{ij} \text{ is not missing}\}$. Decomposing D into UV^T means the predicted value for (i, j) is $\sum_{s=1}^{k} u_{is} v_{sj}$. We then minimize $J = \frac{1}{2} \sum_{(i,j) \in H} (x_{ij} - \sum_{s=1}^{k} u_{is} v_{sj})^2 + \frac{\alpha}{2}(||U||^2 + ||V||^2)$. Letting $e_{ij} = x_{ij} - \sum_{s=1}^{k} u_{is} v_{sj})^2$ and computing the gradient gives us the gradient descent update equations $U \leftarrow U(1 - \alpha\eta) + \eta EV$ and $V \leftarrow V(1 - \alpha\eta) + \eta E^T U$. Here, E is a matrix that has the values in H, but is 0 where H is not defined. Letting n_i be the number of observed entries in row i of $\bar{X}_i$, the outlier score is then $Score(\bar{X}_i) = \frac{1}{n_i} \sum_{j:(i,j) \in H} e_{ij}^2$.

6 Neural Networks: From Linear Models to Deep Learning

A feedfoward neural network is typically a sequence of layers of the form $z = \Phi(\bar{W}^T \bar{X} + b)$ where Φ is an activation function like the sigmoid.

A one-class neural network aims to produce an output of $z = 0$ for any input and nonzero weights. We aim to minimize z^2. We can do this with gradient descent (and we force $\bar{W}$ to have unit norm). The outlier score is then just the z^2 value. To avoid overfitting, we can split the data into two parts. We train on part 1 and score part 2. Then we train on part 2 and score part 1. We can ensemble this. If the neural network is a perceptron (i.e. single layer with linear activation $\Phi(X) = X$), then this is the same as a linear model. We can add many layers and different activation functions and train with backpropagation.

A better alternative to one-class neural networks is replicator networks (also called autoencoders), that simply aim to reconstruct (x') the data x and have a bottleneck layer (i.e. the layer's output dimensionality is less than that of x). We minimize $\sum_{i=1}^{d} (x_i - x'_i)^2$. This is more powerful than matrix factorization because it's nonlinear. The outlier score is the reconstruction error. Pretraining (e.g. greedily training layers one at a time) can help the model avoid local minima. When training a neural net, don't make it too large (you'll overfit) and train on a random subset of the data to reduce outlier impact.

7 Limitations of Linear Modeling

If the features are not correlated and do not fall onto a lower dimensional manifold, it's hard to use linear modeling. Linear models are good to ensemble with proximity based models (chapter 4). Linear models overfit when your dataset is small. Linear models are not easily interpretable.

8 Conclusions and Summary

When features are correlated, linear models can remove outliers. PCA is the best approach to try and can be extended to be nonlinear. Other methods like SVM, matrix factorization, and neural networks can also help.

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.

Chapter 5 - High-Dimensional Outlier Detection: The Subspace Method

1 Introduction

With high dimensional data, most features are noisy, which makes distance computations almost useless. Outliers are best detected in a lower dimensional subspace. Identifying the relevant subspaces is hard. There may also be multiple relevant subspaces, so ensembling techniques helps. We have three methods: rarity-based (tries to find subspaces with rare distributions), unbiased (randomly pick subspaces and find outliers - used with bagging), aggregation-based (use cluster/variance/nonuniformity statistics to pick subspaces).

2 Axis-Parallel Subspaces

Here, we select a subset of the features to represent a subspace. We can consider one point at a time and identifying the subspace where they are unique. Another option is to make a subspace model and score points with it (ensembling models here helps).

The first approach is a genetic algorithm. We take each feature and divide it into bins such that each bin has the same fraction $f = 1/\phi$ of the points. If we have k dimensions, we can create a grid where each cell should have about f^k fraction of the points. If it has much fewer than this, the points are probably outliers. If there are N points, the number of points in a cell has expected value Nf^k and standard deviation $\sqrt{Nf^k(1-f^k)}$. If the number of points in cell $\mathcal{D}$ is $n(D)$, we define the sparsity coefficient $S(\mathcal{D}) = \frac{n(\mathcal{D})-Nf^k}{\sqrt{Nf^k(1-f^k)}}$. This is a nice score (you can combine it with extreme value analysis), but there are a ton of grid cells, so we will to use a genetic algorithm to pick the best cells. We generate some candidate solutions, do recombination and mutation, and then make the next generation of solutions. Doing this iteratively helps us find more fit solutions. We represent each grid cell (a solution) as a string where we indicate the cell or put a "don't care' marker". We then do selection, where we rank solutions by fitness (score) and sample them with greater weight given to fitter solutions. We then do crossover where we greedily pick the k best subspaces from $2k$ subspaces chosen from the two parents. We then do mutation where we randomly change values in the candidate. We repeat this until convergence (95% of solutions are the same).

The *HOSMiner* approach defines the following. For point X, we find subspaces where sum of its k-nearest neighbor distances are at least δ. There are some tricks to make this fast.

Feature Bagging is a powerful technique here. We randomly pick an integer r between $\lfloor d/2 \rfloor$ to $d-1$. We then randomly pick r features without replacement. We run our outlier algorithm on this subset. We do this many times and ensemble the results. You can use many different outlier algorithms too, but you need to normalize the scores (and account for the subspace dimensionality) to combine them (sum them or sum ranks). Use LOF or average k nearest neighbors as your outlier algorithm.

You can also do random clustering in different subspaces and ensemble them.

Subspace histograms are another technique. First, randomly pick a subspace. Then, sample some points and build a histogram. Now, normalize the histogram. Next, score each point using the histogram. Do

this whole process many times and average the results for each point. The bins of the histogram can be representing by hashing their coordinates and using a hashtable to look them up. There are some hyperparameters, but there are hueristics to pick them (see the actual book).

Isolation forests are similar to random forests, which are successful in practice. An isolation tree randomly picks features and randomly picks a split point for each feature and splits the data recursively until each point is in its own leaf. Outliers will be the points closest to the root of the tree, if you average over many trees (i.e. the forest). This definition of isolation forest has no parameters, but it tends to be inefficient if we want to make every point have its own leaf. To speed it up, we can pick a sample of the data (e.g. size 256), train a tree, and score all the points with the tree. We can also limit the depth of the tree. To help pick promising features, we can standardize each feature to zero mean and unit variance and then compute the Kurtosis $K(z_1, ..., z_N) = \frac{1}{N}\sum_{i=1}^{N} z_i^4$. Non-uniform features have high Kurtosis. If you limit the height of the tree (e.g. limit to 10), you need to account for this by assigning a credit score. If the leaf node has r points, the credit is $c(r) = \ln(r-1) - \frac{2(r-1)}{r} + 0.5772$, and should be added to the path length.

We've talked about picking random subspaces, but can we pick high contrast subspaces (HiCS) somehow? Once we find these subspaces, we can run an algorithm like LOF on each one and ensemble the results. Indexing our dimensions as $\{1...p\}$, we observe that $P(x_1|x_2, ..., x_p) = P(x_1)$ if the dimension is uncorrelated with the others. To compute the probabilities, we select a random rectangular region, take the points from there, and compute the relative frequencies (we might repeat this M times and average). We then check for independence using hypothesis testing (e.g. with Student's t distribution). We pick these subspaces using the Apriori algorithm and pruning subspaces. This can be computationally expensive though, so a multidimensional Kurtosis measure might be preferable.

Now let's assume we have a single point and want to find subspaces where it looks like an outlier. The $OUTRES$ method searches for subspaces where the data are nonuniform in the locality of the test point. The local density of the point $\bar{X}$ in subspace S is defined as $den(S, \bar{X}) = |\mathcal{N}(\bar{X}, S)| = |\{\bar{Y} : dist_S(\bar{X}, \bar{Y}) \leq \epsilon\}|$. Then, we need to see if $N(\bar{X}, S)$ is uniformly distributed (null hypothesis H_0) or not (H_1). The Kolmogorov-Smirnoff goodness-of-fit test can do this hypothesis test. Once we find many subspaces that pass the test, we can combine outlier scores from those subspaces $OS(\bar{X}) = \prod_i O(S_i, \bar{X})$ (here, low scores mean the point is more likely to be an outlier). To get $O(S_i, \bar{X})$, we first define $dev(S_i, \bar{X}) = \frac{\mu - den(S_i, \bar{X})}{2\sigma}$ where μ and σ are the mean and standard deviations of points in the neighborhood of $\bar{X}$. Then we set $O(S_i, \bar{X}) = \frac{den(S_i, \bar{X})}{dev(S_i, \bar{X})}$ if $dev(S_i, \bar{X}) > 1$ and we set $O(S_i, \bar{X}) = 1$ otherwise. The book has pseudocode for the $OUTRES$ algorithm.

There's a distance-based way to look for interesting subspaces for a given point. First, we let $S(\bar{X})$ be the k nearest neighbors of $\bar{X}$. The subspace $Q(\bar{X})$ is the set of dimensions of $S(\bar{X})$ where variance is small (according to a threshold). $G(\bar{X}$ is the Euclidean distance of $\bar{X}$ to the centroid of $S(\bar{X})$. We normalize this by the number of dimensions in $Q(\bar{X})$ to get $SOD(\bar{X}) = G(\bar{X})/Q(\bar{X})$. This is not a great approach though because it considers dimensions independently.

3 Generalized Subspaces

Sometimes axis-aligned subspaces don't work. In these cases, the data lies on local nonlinear manifolds. We'll combine PCA-like methods with axis-parallel subspace methods to do this.

One approach is to run a clustering algorithm and then take each point's Mahalanobis distance to nearest cluster centroid as outlier score. Ensemble this by using many clusterings (and perhaps many clustering algorithms).

If you are willing to pay $O(N)$ time for each point (so $O(N^2)$ time total), you can do this. First, find the k nearest neighbors of $\bar{X}$. Then do PCA on $\bar{X}$ and its neighbors to figure out its locality sensitive outlier score.

The Rotated Subspace Sampling improves on Feature Bagging. We recommend using LOF for the outlier analysis. First, we randomly rotate the axes. Then, we randomly pick $r = 2 + \lceil\sqrt{d}/2\rceil$ directions (see

book for how this is done) from the axes and project the points onto this subspace. Then, we run the outlier detector on this subspace. We repeat this many times and average (or take the maximum) scores. Make sure to standardize the scores.

In the case of nonlinear subspaces, you need to use spectral methods. First, find k nearest neighbors (with a kernel for similarity) and find the top eigenvectors with PCA. The similarity matrix is noisy because the data is high dimensional. Using a spectral embedding can clean up the similarity matrix. When we clean up the matrix, we can find the k nearest neighbors again and use PCA again. We repeat this a few more times.

You can also solve d regression problems and look for outliers with regression.

4 Discussion of Subspace Analysis

High dimensionality means we have some uncorrelated features, which means we have noise when measuring distance/similarity. The data lie on a low dimensional (and likely nonlinear) manifold, which we don't know about. This is why subspace analysis is hard. You really need to use ensembles to get good results with subspace analysis.

5 Conclusions and Summary

To deal with high dimensional data, we need to find subspaces where the outliers present themselves.

Chapter 6 - Outlier Ensembles

1 Introduction

Outlier ensemble algorithms can combine many base detectors to create an improved detector. You can also select hyperparameters by trying all of them and creating an ensemble.

When using an ensemble method, you need to pick the base detector and methodology for combining the normalized scores together.

As in classification, we have a bias/variance tradeoff. Variance is how much the outlier score for a point changes depending on which dataset we train our model on (assume the points are sampled i.i.d from some unknown distribution). Bias measures how much our outlier scores deviate from the theoretically optimal scores (assuming they exist). Ensemble methods can reduce both bias and variance.

2 Categorization and Design of Ensemble Methods

A model-centric ensemble uses different models (or the same model with different hyperparameters) as base detectors. A data-centric ensemble uses the same model as base detector, but trains each on a different subset of the dataset.

Ensembles can be independent, where each base detector is run independently. They can be sequential, where one base detector feeds into the next.

Once the base detectors run, how do we combine their scores? We first need to normalize them so that we can compare them. First, we need to decide whether a larger score is more or less anomalous (you can flip the signs of scores based on your decision). Then we need to scale the scores. We can use range based scaling, which produces score $S_j(i)$ for detector j and point i from its original score $s_j(i)$ with

$$S_j(i) = \frac{s_j(i) - min_j}{max_j - min_j} \tag{1}$$

To make our scores less sensitive to the max and min, we can use standardization (even if the scores are not normally distributed, that's ok).

$$S_j(i) = \frac{s_j(i) - \mu_j}{\sigma_j} \tag{2}$$

Another option is to run the EM algorithm on the data points to turn them into probabilities. If you do this, you can get scores that different by orders of magnitude, so you should take a logarithm and use log probabilities as your scores.

To combine scores, you can average them (reduces variance) or take the maximum (can decrease bias, but increase variance).

3 Theoretical Foundations of Outlier Ensembles

The usual bias/variance analysis uses the mean squared error. However we don't have labeled data. We assume our algorithm output scores and oracle outlier scores have zero mean and unit variance. The bias is how much our algorithm output scores differ from the oracle scores. The variance is how much our algorithm output scores change as we randomly pick datasets (to this, you need to designate a set of test points). One way to randomly sample datasets is to take a large set of points and pick random subsets (or you can use a model that makes random choices, like isolation forest). You can then define a bias and variance expression as in classification.

Ensembling can easily reduce variance by averaging different base detector outputs, but it's harder to reduce bias.

4 Variance Reduction Methods

Ensembling by averaging scores always helps reduce variance. If your base algorithm is unstable, the ensemble will be much better than a single detector. That being said, a stable base algorithm might yield a higher overall accuracy (albeit a smaller improvement) when ensembled.

You can ensemble k-nearest neighbors by picking many values of k and averaging scores. More generally, you can avoid picking hyperparameters this way.

If you are using k-means (or histograms), which depends highly on initialization centroids, you can ensemble over many random initializations to reduce variance.

In feature bagging, we pick random subspaces, project points, and score the projection distances. If your subspaces dimensions are small (compared your data dimensionality). ensembling is more effective because each base algorithm is more unstable. Another way to look at this is that each ensemble component looks at the dataset from a different viewpoint and is therefore able to find outliers more easily than a single base detector. Since feature bagging usually uses between $d/2$ and $d-1$ dimensions, base detectors tend to be correlated, which makes the ensemble less useful. Rotated bagging fixes this by randomly rotating the random subspaces.

Isolation forests ensemble isolation trees. The score is the average path length from root to separate out the point. The trees are grown until they fully separate all points (we train each tree on a random subset of the data). We randomly pick a dimension and split point. Isolation forests do well at finding outliers at the edges of the space defined by the data points (i.e. multivariate extreme values). It struggles with outliers near the center of the data points.

For unstable detectors, bootstrap from your original dataset (i.e. randomly sample with replacement). When doing this, make sure you exclude a point (or its copies) when computing k nearest neighbor distances. Each base detector must be trained on a different bootstrapped sample.

If you sample without replacement, you are doing subsampling (not bootstrapping). Bootstrapping works better for isolation forest while subsampling works better for distance methods.

Pick small sample sizes for simple detectors or slow detectors (like k nearest neighbors). You can also pick the sample size randomly. Just randomly pick f between $\min(1, 50/N)$ and $\min(1, 1000/N)$ and sample fN points. Make sure to standardize outlier scores if you do this.

If you do the random sample size selecting, you could use rotated bagging as well. This is computationally efficient and works quite well.

Another variance reduction technique is randomized feature weighting, where we randomly scale the features. You can use wagging, which is bagging, but points are given a random (from Gaussian) weight. You can also use geometric subsampling, which is another way to randomly pick sample sizes.

5 Flying Blind with Bias Reduction

Bias reduction is hard. In classification, we can use boosting, but we can't do that here because we do not have labels. We can roughly do this with some hueristics.

One approach is remove outliers before fitting the detector - but how do we find the outliers? Well, we can fit a detector and remove points with high outlier scores. Then we fit a new outlier detector with the remaining points and keep repeating this.

Another approach is to remove bad ensemble components. First, we compute each points ensemble score and treat these as ground truth. Then we find the detector most correlated with this ground truth and put that in ensemble . We then iteratively add to by adding the detector most correlated with . Use Pearson correlation here. You can also focus on outlier points only here, which requires you to binarize the scores.

When sampling random subspaces, you can use $HiCS$ or $OUTRES$ to pick them instead of just picking randomly.

When sampling random points, you can keep track of which points are marked as outliers and make them less likely to be chosen in the subsequent sample.

6 Model Combination for Outlier Ensembles

You can average (or take median) of outlier scores - this promotes stability (reduces variance). Alternatively, you can take a maximum (this works better with rank scores), but this does not always work. The maximum can sometimes reduce bias because it can ignore bad detectors and find pretty unique outliers, but it may increase variance (but you can mitigate this by using other variance reduction techniques).

Using rank scores can help avoid extreme scores, but can make it harder to distinguish between points (decrease variance, increase bias).

We can combine averaging and maximization. The Average of Maximums approach divides components into buckets, takes a max within each bucket, and averages over the results. The thresh method takes standardized scores and clips them to minimum value of t (usually $t = 0$) to avoid treating inliers differently from strong inliers. You then add these thresholded standardized scores. This can lead to points that have the same final score, so you break ties using the standardized (non-thresholded) scores.

7 Conclusions and Summary

Ensembles are becoming more popular recently, especially on challenging datasets. We have variance reduction and bias reduction techniques, where the latter are more difficult.

Chapter 7 - Supervised Outlier Detection

1 Introduction

Supervised outlier detection is harder than classification. The outlier class tends to be small compared to the rest of the data. The normal class is contaminated with some outliers (we usually only label points as outliers and assume the rest of the points are mostly normal points). We also may not have all possible anomalies in the outlier class (e.g. in intrusion detection, we might not know all possible intrusion methods).

You can first run unsupervised learning algorithms and feed them as features to the supervised learning problem. We also need to do some active learning, where we identify promising points to send to labelers for labeling. We also discuss how to use regression modeling (predict one attribute from the rest and combine the error scores to create outlier score).

2 Full Supervision: Rare Class Detection

False negatives are worse than false positives (i.e. it is better to mistakenly call a point an outlier than it is to miss an outlier). We cannot use the accuracy metric because classes are unbalanced. We must use cost-sensitive learning. Here, we weight the normal class (label 1) less than the outlier classes (labels $2, ..., K$) - the weighting $1/N_i$ for class i is popular. Alternatively we can use adaptive resampling where we oversample the rare case (or even better, undersample the normal class) to build our dataset.

In the MetaCost approach, we train our classifier and score each point with the probability that it belongs to class i ($p_i(\bar{X})$). The misclassification cost is then $\sum_{i \neq r} c_i p_i(\bar{X})$, where r is the true class. We relabel it with the class that minimizes the misclassification cost. Many classification models output scores, so you can use that for $p_i(\bar{X})$. For binary outputs, use bagging with unstable detectors and small samples to get a score. This bagging approach isn't perfect, because the components are correlated.

In weighting methods, we modify the training algorithm so that it can support a weight (misclassification cost) for each training example. We can do this for the Bayes Classifier, Proximity-Based Classifiers, Rule Based Classifiers, Decision Trees, and Support Vector Machines. Weighting works better than sampling because it retains more information. It is slower than sampling though.

Undersampling the normal class can achieve similar effects to weighting and also yields a smaller dataset that is easier to train on. If you ensemble (10 to 25 components), you can acheive accuracy similar to weighting.

Oversampling is problematic because you can duplicate points and thus overfit. To mitigate this, you can use the SMOTE approach. Basically, to oversample a point, you randomly pick a nearest neighbor and sample a point on the line segment to that nearest neighbor. You can boost SMOTE and ensemble it.

Adaboost classifies the dataset many times, updating the point weights each time based on whether they were misclassified in the previous round. In round t, the weight of point i is $D_t(i)$ (it starts at $1/N$). The weight of the next round is $D_{t+1}(i) = D_t(i)e^{\alpha_t}$ if point i is correctly classified and $D_{t+1}(i) = D_t(i)e^{-\alpha_t}$ otherwise. $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ and ϵ_t is the fraction of incorrectly sampled points on a weighted basis. The Adacost algorithm extends this for outlier detection by replacing α_t with either

$\beta_-(c_i)\alpha_t$ or $\beta_+(c_i)\alpha_t$, dependeing on whether the point is correctly classified, to adjust point weights based on their misclassification cost c_i. If you combine boosting with SMOTE, you get SMOTEBoost. Use high bias, low variance classifiers with boosting because boosting reduces bias. Don't use boosting on very noisy datasets.

3 Semi-Supervision: Positive and Unlabled Data

Sometimes, negative (i.e. normal) examples are hard to define. How do you define a representative sample of non-spam emails, for example? Additionally, the normal class may have some outlier class points in it (contamination). One way to deal with this is to use heuristics to pick good normal examples that are not contaminants (we can also just give a normalness weight if we do not want to pick a subset of the unlabeled data as the normal one). We can also learn the normalness weights with a probabilistic model.

4 Semi-Supervision: Partially Observed Classes

Sometimes we don't know all the different kinds of anomalies. This is common in adversarial settings because attackers develop new strategies. One approach here is to train a model on the outliers only and consider ouliers to be the points most similar to them. In addition, we train a model on the normal points and consider outliers to be the points least similar to them. Let's look at each of these.

Sometimes, we do not have normal examples, we only have anomalies. Proximity based methods are really our only option here. It's difficult, but you might be able to get one-class SVMs to work here too.

Sometimes, we only have normal examples, no anomalies. You can use any outlier detection method here. For some reason this scenario is treated differently from regular outlier detection, and maybe it should not be.

Sometimes, we want to detect new kinds of outliers. In this case, we first check if it is similar to any of our known normal examples or outliers. If so, we do multiclass classification to identify what kind of outlier (or normal class) it is.

5 Unsupervised Feature Engineering in Supervised Methods

You can run unsupervised algorithms (LOF, k-nearest neighbors) and use the resulting outlier scores as features. Some features may be highly correlated, so using L_1 regularization can help.

6 Active Learning

First, identify a some interesting examples. Get those labeled. Then train a model on those, and find more interesting examples. And repeat.

How do you find interesting examples? Pick the most ambiguous examples (i.e. the ones closest to decision boundary) - these are high uncertainty. We also pick points that have low likelihood under our trained model (i.e. these are the outliers). Likelihood is easy to measure, because classifiers output it. How do you measure uncertainty? One approach is to pick points where the probability of being an outlier is greater than the fraction of outliers (you need to estimate this from domain knowledge). Another option is query by committee, where you train an ensemble and pick points that have the greatest disagreement by ensemble components.

7 Supervised Models for Unsupervised Outlier Detection

Make sure to standardize your data first. Go one feature at a time and make it the dependent variable. Learn a regression model and compute the error for each example (with cross validation). We then computed a weighted (by regression quality of fit) average of each error for each data point. That is, assuming the squared error of model k on point $\bar{X}_i$, we have $RMSE(M_k) = \sqrt{\frac{\sum_{i=1}^{N} \epsilon_k^2 \bar{X}_i}{N}}$ and $w_k = 1 - \min(1, RMSE(M_k))$. Thus, $Score(\bar{X}_i) = \sum_{k=1}^{d} w_k \epsilon_k^2(\bar{X}_i)$. Notice that this approach lets you identify the dimensions that contribute most to the error. The random forest is a good base detector. This is more flexible than nonlinear PCA because we use the random forest. Make sure not to do PCA before running this method because PCA will try to make each dimension independent of the others.

Another option is to consider r targets instead of just one. Pick r features, do PCA to get uncorrelated targets, and then train regressors to estimate each of them. Then we get: $Score(S_r, \bar{X}) = \frac{\sum_{k=1}^{r'} w_k \epsilon_k^2(\bar{X})}{r'}$.

You also may omit some features from being targets in this regression approach. If a feature is categorical, use a classifier instead of regressor.

You can also create synthetic outlier data (e.g. randomly sample some features from ranges of extreme values). This can cause high bias models though.

8 Conclusions and Summary

It's typical that your normal class is contaminated and you don't know what all the different kinds of outliers are, but we can handle both of these. Active learning is common here as well.

Chapter 8 - Outlier Detection in Categorical, Text, and Mixed Attribute Data

1 Introduction

Extreme value analysis, PCA, Linear Models, proximity algorithms, and LOCI all assume numerical data, so we need to handle categorical variables. One option is to simply one-hot encode each categorical variable, but this is hard to scale if there are many categories.

2 Extending Probabilistic Models to Categorical Data

A mixture model has component weights α_m and distributions $\mathcal{G}_m$ for $m \in \{1...k\}$. We use a Gaussian distribution for numerical data and a Bernoulli distribution for categorical data. Let p_{ijm} be the probability that the i^{th} attribute takes on its j^{th} category as determined by $\mathcal{G}_m$. Now consider data point $\bar{X}$ and let j_r be the category of the r^{th} attribute. Let Θ be the mixture model parameters. We have $g^{m,\Theta}(\bar{X}) = \prod_{r=1}^{d} p_{rj_rm}$. The E-step is then $P(\mathcal{G}_m|\bar{X},\Theta) = \frac{\alpha_m g^{m,\Theta}(\bar{X})}{\sum_{r=1}^{k} \alpha_r g^{r,\Theta}(\bar{X})}$ - these are soft assignments. The α_m values are set to the weighted fraction of points assigned to that component (you can add small value λ to numerator and $k\lambda$ to denominator for Laplacian smoothing). Letting, $\mathcal{D}_{ij}$ be data points where attribute i takes on category j, we have $p_{ijm} = \frac{\sum_{\bar{X}\in\mathcal{D}_{ij}} P(\mathcal{G}_m|\bar{X},\Theta)}{\sum_{\bar{X}\in\mathcal{D}} P(\mathcal{G}_m|\bar{X},\Theta)}$. To smooth this, let v_i be the number of distinct values taken on by attribute i and add small value β to numerator and $v_i\beta$ to denominator. This is the M-step. The outlier score is just $Score(\bar{X}) = \log \sum_{m=1}^{k} \alpha_m g^{m,\Theta}(\bar{X})$.

If you have some categorical attributes and some numerical attributes, you can use two mixture models: $h^{m,\Theta}(\bar{X}) = f^{m,\Theta}(\bar{X}) g^{m,\Theta}(\bar{X})$.

3 Extending Linear Models to Categorical and Mixed Data

If you one-hot encode categorical attribute i (assume it takes on n_i possible values and that it takes on value j with relative frequency f_{ij}), make sure to divide each element of the vector by $\sqrt{n_i f_{ij}(1-f_{ij})}$. For numerical attributes, just make them zero mean and unit variance. Now you can use this data for PCA and linear regression.

4 Extending Proximity Models to Categorical Data

Define similarity between $\bar{X} = (x_1, ..., x_d)$ and $\bar{Y} = (y_1, ..., y_d)$ be $Sim(\bar{X}, \bar{Y}) = \sum_{i=1}^{d} S(x_i, y_i)$. If we set $S(x_i, y_i) = 1$ if $x_i = y_i$ and 0 otherwise, we call this overlap similarity. This isn't great because it ignores aggregate statistical properties of the data (e.g. it is interesting if two users have the same address, but not if they have the same gender) and it ignores local neighborhoods (e.g. Red is more similar to Orange than to Green).

The Eskin measure takes the overlap measure, but returns $\frac{n_i^2}{n_i^2+2}$ if $x_i \neq y_i$. The Inverse Occurrence Frequency (IOF) measure returns $\frac{1}{1+\log f(x_i)+\log f(y_i)}$ if they do not match (here f represents count). Another option is to return $(\log \frac{1}{p_i(x_i)})^2$ if $x_i = y_i$ and 0 otherwise.

To measure contextual similarity (e.g. Red is more similar to Orange than to Green), we need to compare similarity amongst the other attributes. You can do this with random-forest/hierarchical-clustering from Chapter 4. You can also use the Iterative Contextual Distance algorithm (which assumes binary categorical feature). In the first step, you compute a real vector representation for each row by measuring its distance to other data points (distance is measured as distance between attributes). In the second step, you consider each attribute and compute the centroid of points where it is 1 and the centroid of points where it is 0. The L1 distance between the centroids is the attribute distance. Repeat these two steps until convergence. You can alternatively do matrix factorization $D \approx UV^T$ and use the low-rank matrices as distances (this only works on binary categorical features). You can't use PCA here.

If you have a mix of numerical and categorical variables, define $Sim(\bar{X}, \bar{Y}) = \lambda NumSim(\bar{X}_n, \bar{Y}_n)/\sigma_n + (1-\lambda)CatSim(\bar{X}_c, \bar{Y}_c)/\sigma_c$. Set λ to be the fraction of numerical attributes. To turn a distance into a similarity, use $\frac{1}{1+dist}$ or $\exp \frac{-dist^2}{t}$

Density methods already discretize numerical data, so they can handle categorical data easily.

For clustering methods, just use the mixture model described earlier in this chapter.

5 Outlier Detection in Binary and Transaction Data

Transaction data is usually binary and sparse (it indicates which items in a catalog that a user has bought). A dense subspace indicates a frequent pattern (i.e. many customers buy this group of items) and is therefore not indicative of outliers. Given transation database $\mathcal{D}$ containing $T_1, ..., T_N$, let $s(T_i, \mathcal{D})$ be the support of T_i in $\mathcal{D}$. Let $FPS(\mathcal{D}, s_m)$ be the set of frequent patterns at minimum support level s_m. The Frequent Pattern Outlier Factor is $FPOF(T_i) = \frac{\sum_{X \in FPS(\mathcal{D}, s_m), X \subseteq T_i} s(T_i, \mathcal{D})}{|FPS(\mathcal{D}, s_m)|}$

6 Outlier Detection in Text Data

You can represent a document with a word frequency vector (or a vector with a 1 if word appears and 0 otherwise = bag of words). We can use a mixture model where $P(\mathcal{G}_j|\bar{X}_i)$ is the probability document i belongs to component j and $P(t_l|\mathcal{G}_j)$ is the probability that term l occurs in a document of component j. We use a Bernoulli distribution for bag of words and a multinoulli distribution for word frequency vectors. We have $P(\mathcal{G}_j|\bar{X}_i) = \frac{\alpha_j P(\bar{X}_i|\mathcal{G}_j)}{\sum_{r=1}^k \alpha_r P(\bar{X}_i|\mathcal{G}_r)}$. To compute this, we need the Bernoulli parameters p_l^j which is the probability of term t_l appearing in component j. So, we use the EM algorithm. First, randomly hard-assign documents to mixture components and estimate α_j as the fraction of documents in component j and p_l^j as the fraction of documents in component j that have term t_l. The E-Step computes the soft assignments $P(\mathcal{G}_j|\bar{X}_i)$ using the equation shown before. The M-Step sets p_l^j to the weighted fraction of documents in component j that have term t_l and α_j is the weighed fraction of documents belonging to component j. We then have $P(\bar{X}_i) = \sum_{r=1}^k \alpha_r \prod_{t_l \in \bar{X}_i} p_l^r \prod_{t_l \notin \bar{X}_i} (1 - p_l^r)$.

Synonemy is when two words describe the same concept (e.g. car and automobile) and polysemy is when a single word describes multiple concepts (e.g. lead means either a dense metal or to be a leader). Latent Semantic Analysis (LSA) is SVD for text. Let D be an $N \times d$ matrix where D_{ij} is the normalized frequency of term j in document i. Compute the top 300 or 400 eigenvalues of $D^T D$ (this approximation mitigates synonemy and polysemy). Documents with large values on the small eigenvectors are likely noise or outliers. Probalistic LSA is extension where we create a generative process for generating documents (that can be formulated as SVD) and use the EM algorithm to figure out the parameters (i.e. the elements of the SVD matrices). Read the book for more info. Honestly, this topic modeling stuff seems a bit outdated given that we have word/document embeddings and deep learning.

When representing a word's frequency in a document, use the term-frequency inverse-document-frequency (TF-IDF) instead. Then normalize each document vector to unit length. Then you can compute similarity of documents with cosine distance $Cosine(\bar{X}, \bar{Y}) = \frac{\bar{X} \cdot \bar{Y}}{||\bar{X}||||\bar{Y}||}$. Now that we have a similarity function, we can use proximity based methods.

For a streaming application, you can create document vectors and cluster them. When you get a document online, you measure distance to nearest cluster. If it's too far, it's an outlier and starts a new cluster. Otherwise, you can add it to an existing cluster and move its centroid. This a useful technique for first story detection (where you are looking for the first story about some new topic).

7 Conclusions and Summary

We can extend our models for categorical variables, including text.

Chapter 9 - Time Series and Multidimensional Streaming Outlier Detection

1 Introduction

Assume we have a time series and want to find outliers within it. One kind of outlier is an abrupt change in the value, which is useful for series with high continuity (e.g. two consecutive sensor readings are usually almost identical). In cases where we have low continuity (e.g. news articles coming down a wire) we have some overall trend, but consecutive points may be a good deal different from each other. Here, we want to detect novelties (points very different from historical points) and changes (a change to the overall trend of the series).

A contextual outlier is a point that is different from its neighbors. A collective outlier is a subsequence of contiguous points that are different from the rest of the neighbors.

Our algorithms might run offline or online. Offline gives us access to history and lets us use more complex algorithms. We might have labels for time series, chunks of time series, or for points. In this case, use supervised learning.

2 Prediction-Based Outlier Detection in Streaming Time Series

We often care about deviations (contextual outliers). We care about correlations accross time and accross different series. An autoregressive model, $AR(p)$ defines the current time series value as a function of the past: $X_t = \sum_{i=1}^{p} a_i X_{t-i} + c + \epsilon_t$ for window size p and error terms (outlier scores) ϵ_t. We can infer parameters with least squares as follows. Let D be a matrix where column i is $X_i, ..., X_{n-p+i-1}$ and the final column is all 1. Let $\bar{y}$ be the column vector $X_{p+1}, ..., X_n$ and let Θ be the column vector $a_p, a_{p-1}, ..., a_1, c$. We minimize $||\bar{y} - Da||^2$ and get $a = (D^T D + \alpha I)^{-1} D^T \bar{y}$ where α is a regularization parameter. You can also do this online with an efficient algorithm for matrix inversion.

The moving average model $MA(q)$ is $X_t = \sum_{i=1}^{q} b_i \epsilon_{t-i} + \mu + \epsilon_t$ - you need nonlinear methods to fit this. The $ARMA(p, q)$ model is $X_t = \sum_{i=1}^{p} a_i X_{t-i} + \sum_{i=1}^{q} b_i \epsilon_{i-1} + c + \epsilon_t$. If p and q are too large, you overfit. Pick them with leave-one-out cross validation. Sometimes, a time series is nonstationary (e.g. random walk). So, we take successive differences and then use $ARMA(p, q)$, this is $ARIMA(p, q)$. We also might take a logarithm of each value before taking differences.

We can extend all the above models to multiple time series. For example: $X_t^j = (\sum_{k=1}^{d} \sum_{i=1}^{p} a_i^{kj} X_{t-i}^k) + c^j + \epsilon_t^j$. This is computationally intensive because the matrix is larger. So, you can just select a subset of streams. Alternatively, you can create hidden variables and do univariate time series analysis for each. The Muscles technique uses the matrix inversion lemma can help do matrix inversion incrementally as you add new points, but is slow when you have too many series, so it uses a subset of them. Basically, given a stream that we want to predict, we greedily pick the most correlated streams until we have enough (this algorithm is called Selective Muscles). The problem with these multiple time series method is they have a lot of coefficients and do not handle outliers well.

The SPIRIT algorithm handles multiple time series with PCA. We compute the $d \times d$ covariance matrix between streams and project each data point onto the top k eigenvectors (the rest are basically just constant values). This yields k uncorrelated streams that can be processed independently. We can

then transform the results back (along with the $d-k$ constant streams) to the original space. Then you can compute the error terms as usual. In order to compute a running covariance, you need to keep a running sum for each series and a running pairwise sum for each pair. This lets you compute $Cov(\bar{X}^j, \bar{X}^k) = \frac{\sum_{i=1}^t X_i^j X_i^k}{t} - \frac{\sum_{i=1}^t X_i^j}{t} - \frac{\sum_{i=1}^t X_i^k}{t}$.

You can represent groundtruth by getting a set of outlier timestamps $T_1, ..., T_r$ (these are the primary abnormal events - we will also find other anomalies called secondary abnormal events). First, run one of the previous time series analysis algorithms and compute error terms. Normalize them to zero mean and unit variance: $z_t^1, ..., z_t^d$. The alarm level at time t is then $Z_t = \sum_{i=1}^d \alpha_i z_t^i$. The alarm level during primary events is $Q^P(\alpha_1, ..., \alpha_d) = \frac{\sum_{i=1}^r Z_{T_i}}{r}$ and during normal events is $Q^n(\alpha_1, ..., \alpha_d) = \frac{\sum_{i=1}^n Z_i}{n}$. We aim to minimize $Q^P(\alpha_1, ..., \alpha_d) - Q^n(\alpha_1, ..., \alpha_d)$ subject to the regularization $\sum_{i=1}^d \alpha_i^2 = 1$. Learn this with some off the shelf optimizer.

3 Time Series of Unusual Shapes

How do we find collective outliers? We may want to detect if an entire time series has an anomalous shape. Alternatively, we may want to find subsequences within a time series that are anomalous. We focus on the full series case because you can always solve the subsequence case by breaking a time series into small pieces. Let us assume all series are normalized to zero mean and unit variance and that all series have the same length n.

Numeric multidimensional transformation (NDT) lets us take a time series and turn it into a vector that we can compare to other time series. Discrete sequence transformation (DST) turns our time series into sequences of symbols that we can analyze using methods in chapter 10.

For NDT, we can use the Haar Wavelet. The first element of our vector is the global average. We then split the series in half and compute difference between the averages of each half. Then recursively process each half to get the remaining $n-2$ coefficients. We cannot drop any of the n coefficients because they may be useful for outlier detection (you can also compare them with Euclidean distance). Now, we have a bunch of n dimensional vectors with no temporal dependency, so we can use other outlier techniques. The discrete Fourier transform is an alternative to the Haar Wavelet. Use Fourier transform when you have seasonality and use wavelets when you have series that usually do not change much over small ranges.

For DST, we use Symbolic Aggregate Approximation. First, we split the time series into windows and compute the average of each window. Now we assume the averages are distributed as a Gaussian, which we break into equal probability mass bins and assign each average its bin ID (we now have each bin ID appearing roughly the same number of times).

To detect unusual shapes, you can treat the time series as a trajectory and use the methods from Chapter 11. TROAD is a good algorithm for this. This is a hard problem.

The Hotsax technique runs a sliding window over the series, converts each into a multidimensional vector, and then computes each window's Euclidean distance to other windows to find outliers. You could also use Dynamic Time Warping instead of Euclidean distance. To make this computationally efficient, we can use pruning. We have a nested for-loop where the inner loop looks for the k nearest neighbors. We keep a running update of the top-r outliers, so we can terminate the inner loop early if it is clearly not going to be in the top-r. There are other techniques for doing this (see book).

If you turn a time series into a sequence of symbols, you can use a Hidden Markov Model.

You can slide a window over the series, treat each window as a point, and use PCA. You can also do this if you have multiple time series. Use kernel PCA if you want non-Euclidean similarity measures like Dynamic Time Warping or edit distance. You can also use one-class SVM in this way.

If you have labeled data, learn a model for the normal case and abnormal class. Then just measure distance to both to see which is a better fit. Alternatively, convert your time series into symbols and use a Hidden Markov Model to classify.

4 Multidimensional Streaming Outlier Detection

Autocorrelation and continuity may be weaker in multiple time series than in univariate time series. Again, we may have deviation (novelty) outliers and collective (change in trend) outliers.

To detect novelties, just keep a window of points and measure distance to each point. The LOF algorithm has been extended to the incremental scenario. You can also keep running clusters and measure distance to them. A novelty is far from existing clusters and creates a new cluster. You can also build a mixture model (they are well suited for online algorithms). The SPOT algorithm is good for high dimensional streaming time series.

To detect trend changes, you can use velocity density estimation (see book for the map). This is like kernel density estimation, but using temporal kernels. You can set the kernel width to determine long term or short term trends. There are also techniques to fit probability distributions to sections of the time series and look for statistically significant differences.

Supervision can indicate rare-class outliers, novel class outliers, and infrequently recurring outliers. You also need some unsupervised learning to detect completely new kinds outliers. For rare classes, you can use a classifier that can handle nonstationary distributions. You need to account for the class imbalance here (see chapter 7). For novel classes, you should have your regular supervised classifier and an unsupervised clustering model for novel classes so that you can tell if a novel class (i.e. not in training set) has been seen before during test time. For infrequently recurring outliers, you could just mark them as novel, but it's better to remember a distribution for this outlier and store it in case you see it again.

5 Conclusions and Summary

Time series can be univariate or multiple series. Outliers can be novel or collective. You can use supervised, unsupervised, and hybrid techniques.

Chapter 10 - Outlier Detection in Discrete Sequences

1 Introduction

We want to find outliers in a sequence of discrete symbols taken from an alphabet Σ. Position (or contextual) outliers are symbols that are strange given the context they are in. A combination (collective) outlier is a subsequence of points that are strange.

You can break the sequence into subsequences and cluster the sequences or do k-nearest neighbors. You can also build frequency distributions for subsequences and compare that to a training set. You can also use a generative model, like the Hidden Markov Model, to predict the likelihood of subsequences. Finally, you can transform your subsequences into a new space and use PCA.

2 Position Outliers

We try to predict symbols, and if a symbol is highly unlikely, it is an outlier. To do this, we approximate $P(a_i|a_1, ..., a_{i-1}) \approx P(a_i|a_{i-k}, ..., a_{i-1})$. We can estimate these probabilities with rule based models and Markov models.

$P(a_i|a_{i-k}, ..., a_{i-1})$ is called the confidence of a rule. For large k, the antecedent $(a_{i-k}, ..., a_{i-1})$ may be rare, so we put a lower bound the number of times it must appear in order to form a rule (support criterion). We can also vary the antecedent length and put wildcard (i.e. do not care) symbols. RIPPER is a good algorithm for inferring rules.

In a first order Markov model, each symbol is a state and we make weighted (transition probability is the weight) between states. More generally, you can make a Markov model of order k with $|\Sigma|^k$ states. These higher order models can overfit and are computationally expensive. To speed them up, you can use variable length states and wildcard symbols. You can also create a Probabilistic Suffix Tree to make it easy to compute the conditional probablities needed to compute the probability of a sequence. You need to make sure to do some pruning here (e.g. sequences with low probabilities or that do not appear often in the data can be removed). If a sequence is not present in the tree, use the longest matching one that is available or just use the overall longest suffix in the tree - both will have low probability. Markov models can also be used to correct noisy sequences and to detect combination outliers.

3 Combination Outliers

We may have some sequences labeled as normal (semi-supervised) or have no labeling (unsupervised). For both cases, use a one-class model. If sequences are short, we can convert them into points and use k-nearest neighbors. Note that if you have just one long sequence, you can break it into smaller sequences.

If you know that some sequence is bad (e.g. many login symbols in a row), you can incorporate those as well as comparison units.

Now, assume we have training sequence database $\mathcal{D} = T_1, ..., T_N$, test sequence V, comparison units $U_1, ..., U_r$ (these can be provided by the user or generated by sliding a window over V), and a model $\mathcal{M}$.

We seek an outlier score for the comparison units. If they are given by the user, we can compute the rarity (via M) of the units in $\mathcal{D}$ and then in V and take a difference. If they come from sliding over V, we just compute the rarity in $\mathcal{D}$. We then need to combine the scores.

For $\mathcal{M}$, we have a number of options. Consider distance based methods first. We can measure the distance between two sequences by measuring the fraction of symbols they have in common. We can measure the longest common subsequence (a subsequence is a child sequence you get by dropping symbols from the parent sequence) and normalize $NL(T_1, T_2) = \frac{L(T_1,T_2)}{\sqrt{|T_1|}\sqrt{|T_2|}}$, but this is expensive. We could use edit distance (expensive). We could use compression-based dissimilarity (compress T_1, then T_2, then $concat(T_1, T_2)$ - the greater the savings when compressing the concat, the more similar the sequences are). We may also want to give contiguous mismatches more weight than faraway mismatches because anomalies tend to occur in contiguous symbols.

Once we score each comparison unit, how do we combine them all? We could binarize the scores and sum them (not ideal because it ignores actual scores). We could aggregate scores with a sum or average (not ideal because normal units can be noisy). We could merge the two approaches and group the units into windows, do the binarize approach for the windows, and then aggregate over the results. We could also try to show preference for clustered anomalous units as follows. In locality frame count, we can examine groups of contiguous units and if the number of anomalous units is over a threshold, we say the group is anomalous. In leaky bucket, we keep a running count of the difference between number of anomalous and numbber of regular units (with a minimum value of 0).

If comparison units are given by the user, we let $f(T, U_j)$ be the number of times U_j occurs in T. We can normalize to get $\hat{f}(T, U_j) = \frac{f(T,U_j)}{|T|}$ and get an anomaly score $A(T_i, V, U_j) = |\hat{f}(V, U_j) - \hat{f}(T_i, U_j)|$. If the comparison units are too short, this might not be useful. This also works if you slide over V to make comparison units, but people don't really use it.

Hidden Markov Models (HMMs) have states that transition to each other and output a symbol at each state. We start with a state (defined by n initialization probabilities) and at each time step we transition to a new state (defined by n^2 transition probabilities) and emit a symbol (defined by $n|\Sigma|$ emission probabilities). You should think through how many states you want your system to have. Let our states be $\{s_1, ..., s_n\}$, symbols be $\sigma_1, ..., \sigma_{|\Sigma|}$. Let $\theta^j(\sigma_i) = P(\sigma_i|s_j)$. Let p_{ij} be the probability of transitioning from s_i to s_j. Let $\pi_1, ..., \pi_n$ be the initialization probabilities. We are given training sequences $T_1, ..., T_N$. There are three common tasks we do with HMMs. First, we estimate the probabilities from the training data (using the Baum-Welch algorithm). Second, we measure how well a comparison unit U_i fits our HMM and use this as the anomaly score (we use the forward algorithm here). Third, given a comparison unit U_i, we find the most likely state sequence (with the Viterbi algorithm) to help understand why an outlier appeared. See the book for details on Baum-Welch, forward, and Viterbi.

The HMM does not work well if the test sequence is extremely long, which is why we extract small comparison units. Before combining the anomaly scores from the comparison units, take a logarithm (because they are probabilities).

You can use a mixture of HMMs to do clustering of sequences.

If we have short sequences, we can use kernel based methods. First, we compute our kernel matrix (i.e. pairwise similarity matrix) S, which is both symmetric and positive semi-definite. We can do an eigendecomposition with the top k eigenvectors and get $S \approx Q\Lambda^2 Q^T$. We can then normalize $Q\Lambda$ so each column has unit norm and use the Mahalanobis method to find outliers. Make sure to use a valid string kernel. If you don't, then just set negative eigenvalues to 0.

4 Complex Sequences and Scenarios

For multivariate sequences, we have multiple symbol sets $\Sigma_1, \Sigma_2, ..., \Sigma_r$ - one per dimension. Additionally, the symbols may arrive at different timestamps (imagine you have different sensors sending values at different frequencies). To get an anomaly score, we consider time windows, do univariate analysis on

each dimension and multiply the outlier scores for each dimension together. If each sequence is short, we can take all dimensions together and use a kernel method.

Sometimes, instead of a symbol we will get a set of symbols. In this case, we represent each set as a binary vector of size $|\Sigma|$. We can do first-story detection here. We can also use temporal difference length algorithms.

Sometimes we need to detect anomalies in an online fashion. For position outliers, we can keep a short window in memory and use an incremental suffix tree. For combination outliers, we can keep a short window and add our window to the dataset when we are done (if the dataset is too large, take a sample from it).

5 Supervised Outliers in Sequences

Usually, anomaly detection in discrete sequences tends to find noise, so supervision helps a lot. Make sure to deal with imbalanced classes and cost-sensitivity here. Our labels may tag a single position, a subsequence, or a whole sequence. Then you can combine the techniques from chapter 7 and use any sequence classification algorithm that you want. Some sequence classification algorithms are as follows. You can compute k-grams or do a wavelet decomposition, turn them into features, and classify them with an SVM. You can use edit distance to see if the test example is closer to the normal class or outlier class. You can also use the HMM and use a separate emission probability model for each class. You need to adjust the forward algorithm to incorporate cost-sensitivity here. You can use kernel methods with string kernels. Of all of these methods, SVMs with good features work the best.

6 Conclusions and Summary

Discrete sequences are common in situations where you are logging different events. We can find positional and collective outliers. HMMs are popular here. Supervision helps a lot.

Chapter 11 - Spatial Outlier Detection

1 Introduction

Spatial data measures how some behavioral attribute (e.g. wind speed) varies over a contextual attribute (e.g. latitude and longitude). Sometimes, a spatial quantity, like latitude/longitude, can be the behavioral attribute and time is the contextual attribute (this is called trajectory analysis). Sometimes, we have space AND time as contextual attributes, which we call spatiotemporal data.

We leverage two characteristics in our data. First is spatial autocorrelation, which simply says nearby spaces are more similar than distant ones. Second is spatial heteroscedasticity, which says the variance of the behavioral attribute depends on its location in space. The former is more used than the latter. We can find contextual outliers (points in space with weird behavior compared to their surroundings) and collective outliers (weird patterns that can cover many points in space). Supervision is useful here.

The key difference between spatial data and time series data is that spatial is multidimensional and not unidirectional. Also, space may be discrete (e.g. zip code).

2 Spatial Attributes are Contextual

Neighborhood algorithms look for abrupt changes in a neighborhood. The neighborhoods can be defined via distances or via a edges in a graph.

You could use LOF, but that doesn't incorporate the difference between the behavioral and contextual attributes. For example, k nearest neighbors in spatial data must just look for points nearby in space. We can find the nearest neighbors and average (or take median) of their behavioral values. If the point in question has very different behavioral attributes than the neighborhood, it is an outlier. You should normalize the difference by the standard deviation of the neighborhood behavioral attributes.

We may represent space with a graph. Let o be our point, $o_1, ..., o_k$ be the k-nearest neighbors, $w(o, o_i)$ be the edge weight between o and o_i, and $f(o)$ be one of the behavioral attributes. We can compute a neighborhood mean with $g(o) = \frac{\sum_{i=1}^{k} w(o,o_i) f(o_i)}{\sum_{i=1}^{k} w(o,o_i)}$. You can then do extreme value analysis on the normalized difference of each point to its neighborhood mean as described above. If you have multiple behavioral attributes, compute the normalized differences independently, use the Mahalanobis method, and then do extreme values on the Mahalanobis outlier scores.

An autoregressive model assumes space is represented with coordinates in a grid and that most of space has behavioral attributes (i.e. not sparse). Our model is then $X_{t_1,t_2} = \sum_{i=-p}^{p} \sum_{j=-p}^{p} a_{ij} X_{t_1-i,t_2-j} + c + \epsilon_{t_1,t_2}$ where we force $a_{00} = 0$ to avoid using a value to predict itself. You can solve this with least squares. We have ARMA and ARIMA variants of this like we did with time series. Autoregressive models are not used often in the literature because data is usually sparse.

Consider a pair of points and measure the distance between their spatial attributes and measure another distance between their behavioral attributes. If you plot behavioral distance vs. spatial distance for all pairs of points, you get a variagram cloud. High behavioral distance with small spatial distance represents an outlier. Processing all pairs of points is expensive, so you can simply discretize space into a grid and make each grid cell a point.

How do we detect anomalous shapes in spatial data? One option is contour detection. A contour is a boundary of a shape. Notice that this can be turned into a time series if the x-axis is given by a clockwise sweep of the contour and the y-axis is the distance from the centroid. If you do not normalize the time series, that means you care about how large the contour is. If you scale to unit mean, that means you do not care about size but you do care about relative local variation. If you do zero-mean and unit variance, it means you do not care about size or relative local variation (circular shapes can cause problems here). None of these account for rotation, however, which shifts the time series. To account for rotation, we can use the rotation invariant Euclidean distance between two time series $RIDist(T_1, T_2) = \min_{i=1} n\sum_{j=1}^{n} (a_j - b_{1+(j+i) \bmod n})^2$. With this in hand, an outliers can be detected with the k-nearest neighbors method (use early inner loop termination to avoid doing too much computation).

You can also split space into $p \times p$ grids, turn each grid into a vector, and then use multidimensional methods for outlier detection. You can also use the Haar wavelet transform to turn the grids into features.

Supervision helps a lot. Just learn some shapes from the normal class (and the outlier class, if you have it) and then see if the test shape is more similar to the normal class or outlier class (using the shape to time series method discussed earlier).

3 Spatiotemporal Outliers with Spatial and Temporal Context

You can extend previous approaches to spatiotemporal data. Just define the neighborhood to span time as well (be careful about how you normalize here so you can compare across both space and time).

4 Spatial Behavior with Temporal Context: Trajectories

You can think of this as multiple time series and use the time series analysis methods. You can use kernel methods to make a similarity matrix, use the Mahalanobis method on the matrix, and then do extreme value analysis on the outlier scores.

In a streaming situation, you can process each time series separately. Then you can sum their outlier scores (assumed to come from a normal distribution) and model it with a χ^2 distribution.

For unusual shape detection, ignore the time dimension and use some of the techniques described earlier. Another option is to break a trajectory into pieces and classify each piece into one of a set of fixed trajectories (the symbols). Now you get a symbol frequency vector and you can find k-nearest neighbors. This also lets you turn the trajectory into a discrete sequence and you can use the techniques in chapter 10 for this.

You may have some supervised trajectories. Turning these trajectories into a sequence of symbols and then using a supervised discrete sequence technique is helpful here. If you do not know good symbols beforehand, you can do clustering to infer them.

5 Conclusions and Summary

Spatial outlier dimension is similar to temporal outlier detection (time series). We can also handle spatiotemporal data and trajectories. This chapter also showed how to convert a shape into a time series.

Chapter 12 - Outlier Detection in Graphs and Networks

1 Introduction

The dataset may consist of many small graphs over a common set of nodes (in this case we are looking for outlier graphcs) or it could be one large graph (in this case we are looking for outlier nodes and edges). We also may have graphs that change over time.

2 Outlier Detection in Many Small Graphs

We treat each graph is a single point and use multidimensional outlier detection. To represent a graph as a point, we can mine frequent subgraphs and represent a graph as a bag of frequent subgraphs. You can also use k nearest neighbors with a graph distance metric (e.g. graph edit distance, largest commons subgraph, largest matching node set). There are also graph kernels (e.g. random walk kernel, shortest path kernel), so you can make a kernel (similarity) matrix and use a one-class SVM, soft kernel PCA, and/or the Mahalanobis method.

3 Outlier Detection in a Single Large Graph

Let our undirected graph be $G = (N, A)$ with n nodes and m edges.

To find node outliers, we need some features. We consider a node i's 1-neighborhood and compute the number of neighbors (n_i), number of edges between all neighbors (e_i), sum of weights to neighbors (w_i), and the principal eigenvalue of the weighted subgraph in the 1-neighborhood of i (λ_i). We can then plot n_i vs. e_i, w_i vs. e_i, and λ_i vs. w_i and look for deviations from the trend (which will be a power law distribution). Or, more generally, we can consider the $1, 2, ..., k$-neighborhood and compute features for each, create a point, and use our regular multidimensional outlier detection algorithms. You can also use domain specific features (e.g. number of messages sent between node and its k-neighborhood). If you want to use the Mahalanobis method, you need to make a similarity matrix (use SimRank, PageRank difference, Jaccard coefficient, or Katz measure as your similarity metric).

To find linkage (edge) outliers, matrix factorization works well. The edges are represented with adjacency matrix A. We then minimize $||A - UV^T||^2$ subject to $U, V \geq 0$. Then $R = A - UV^T$ gives us the outlier scores for each edge, and you can use extreme value analysis here. The entries of U and V can provide some interpretability as well. When doing the optimization, you should use regularization term $\alpha(||U||^2 + ||V||^2)$. For large networks, you need to use special techniques to make the optimization tractable. You can compute a score for each node i by averaging row i and column i of R (ignore the diagonal entries though). Notice that this technique can be extended to directed and weighted graphs. If you set the diagonal entries of A to the node's degree, you get an eigendecomposition where $U = V = P\sqrt{\Lambda}$ and $P\sqrt{\Lambda}$ is an embedding of each node (so you can use Mahalanobis method). And of course, if you don't want to use the adjacency matrix, you can use a kernel method instead.

We can use clustering methods for linkage outliers as well. We cluster nodes and edges connecting clusters are therefore outliers. To avoid the instability of random initialization, we use randomized ensembles for clustering. To do this, we randomly sample (you can be clever about the sampling strategy) edges from

the adjacency matrix and find connected components. Given clusters $\mathcal{C} = C_1, ..., C_k$, we want to find $p_{ij}(\mathcal{C})$, which is the probablity that a randomly chosen edge connects C_i and C_j (this is easily computed from the data). We also define $I(i, \mathcal{C})$ to be the cluster index (1 to k) for node i. The likelihood fit of an edge is therefore $p_{I(i,\mathcal{C}),I(j,\mathcal{C})}$. We should do this for each clustering in the ensemble and take a median, we call this $\mathcal{MF}(i, j, \mathcal{C}_1, ..., \mathcal{C}_r)$. If we have multiple graphs, we can define this metric for an entire graph $\mathcal{GF}(G, \mathcal{C}_1, ..., \mathcal{C}_r) = (\prod_{(i,j)\in G} \mathcal{MF}(i, j, \mathcal{C}_1, ..., \mathcal{C}_r))^{1/|G|}$.

To find subgraph outliers, we can use minimum description length (MDL). Basically, we look for a common substructure and compress it into a single node. The description length is $F1(S, G) = DL(G|S) + DL(S)$, which leads to high outlier scores for individual nodes, so we fix this with $F2(S, G) = Size(S) + Instances(S, G)$, where $Size(S)$ is is the number of nodes in the subgraph and $Instances(S, G)$ is the number of instances of S in G. Another option is the SUBDUE method.

4 Node Content in Outlier Analysis

Sometimes, nodes have content (e.g. words) inside them that we want to leverage. Imagine we have adjacency matrix A and content matrix C (one row per example, one column per word, and value is frequency). We can then use matrix factorization and minimize $||A - UV^T||^2 + \beta||C - UW^T||^2 + \alpha(||U||^2 + ||V||^2 + ||W||^2)$ subject to $U, V, W \geq 0$ using gradient descent. Then you can take $A - UV^T + C - UW^T$ to get outlier scores. Heterogeneous Markov Random Fields and Tie Strength are also useful here.

5 Change-Based Outliers in Temporal Graphs

This is also called evolutionary network analysis. We have node anomalies (e.g. the edges around a node are doing strange things), linkage anomalies (e.g. strange edges), community evolution (e.g. unsual changes in community structure), distance evolution (e.g. distance between nodes changes a lot), and latent embedding changes (i.e. if you embed the graph in a smooth space, are there changes there?).

We can create an adjacency matrix, $A(i)$, around a node i. If we keep track of how the eigenvectors of $M(i) = A(i)^T A(i)$ change, then we can detect anomalies in the amount of activity (magnitude of eigenvectors changes) and linkage patterns (angles of eigenvectors change).

We can also use linkage anomaly detection functions here. We just need to update clusters as we get new clusters. To do this, we need reservior sampling with a monotonic set function (i.e. monotonically non-decreasing set function has if $S_2 \subseteq S_1$ then $f(S_1) \geq f(S_2)$). Some options are the number of connected components in S or the number of nodes in the largest connected component of S. Making some insights from this, we can use a special hash function to determine which edges to drop and add as we process a stream.

The ENetClus (extends NetClus) lets us do soft clustering on data streams and detects changes in those clusters.

We can also maintain a differential graph, which measures how much activity happens on an edge over time.

GraphScope is a good technique for handling bipartite graphs.

We may also want to look at how shortest paths between nodes changes. This is expensive, so there are randomized heuristic techniques for approximating it. See the book for links to these papers.

We can also do matrix factorization $A_t \approx U_t V_t^T$ by minimizing $\sum_{t=1}^{T} ||A_t - U_t V_t^T||^2 + \alpha \sum_{t=1}^{T-1} ||U_t - U_{t-1}|| + \beta \sum_{t=1}^{T-1} ||V_t - V_{t+1}||^2$ subject to nonnegativity constraints. There a variety of ways to extend this method with new regularization functions, orthogonality constraints, etc. This method works for many kinds of graphs.

6 Conclusions and Summary

There are various ways to define outliers in graphs (e.g. nodes, edges, subgraphs) and we get even more options when the graph evolves over time. We also have techniques for the situation when the nodes have content.

Chapter 13 - Applications of Outlier Analysis

1 Introduction

Most domains have dependency oriented data (e.g. time series, spatial data), which are harder to analyze. Supervision is often essential because unsupervised methods usually find noise, so use active learning if you can. Simple algorithms like k-nearest neighbors and the Mahalanobis method work very well. Use ensembles whenever you can.

2 Quality Control and Fault Detection Applications

If you have n widgets and know that widgets fail with probability p, you can use the probabilistic inequalities and extreme value analysis techniques from chapter 2.

Suppose we are monitoring various sensors in a running system and we want to detect faults. Extreme value analysis and abrupt change detection can be useful here. Obviously time series analysis is useful here (especially with supervision). If you treat time series as trajectories, the unusual shape detection algorithms can help. Streaming algorithms are useful here.

Suppose we have a 2D or 3D surface and we want to detect defects over time. This is spatiotemporal data. Neighborhood algorithms and unusual shape detection are useful here. You can also use temporal analysis. Obviously, supervision helps.

3 Financial Applications

Suppose we are trying to identify credit card fraud given user transactions and locations. Supervision is common here because we often have labeled data. Extreme value analysis is a good first step. Also, building per-user profiles from a short sequence of their transactions and comparing new sequences to that works well. In this case, a false negative is more dangerous than a false positive, so you should account for this. Discrete sequence methods are popular here.

Suppose we want to detect anomalous insurance claims. Note that user-specific profiles are not useful here because users do not usually make many claims. We typically extract features from the claim document and use a multidimensional outlier supervision technique.

Suppose we want to detect anomalous stock market behavior (e.g. to detect insider trading, flash crashes). News streams are useful here. Streaming methods are useful here. Time series analysis is useful.

Financial entities interact with each other. Suppose we want to detect anomalous interaction patterns. This is a temporal graph, so use those methods. Usually the edges have content, so you should extend the methods from chapter 12 to account for this.

4 Web Log Analytics

Suppose we log access to a website and want to detect anomalies. Domain knowledge helps here (e.g. lots of accesses to login page are suspicious). Position and combination outliers are useful here. Hidden Markov Models are popular here.

5 Intrusion and Security Applications

Suppose we want to detect a break-in into a computer system or network. If you have a single host being monitored, this is discrete sequence analysis (the OS system calls) and is very similar to the web log analytics case. For the network case, you could have various kinds of data (e.g. the packets), so it is more like multidimensional outlier detection because the temporal link is weaker. Some features, like number of bytes in a packet are useful. This is also a streaming situation. You also need to be able to detect novel classes.

6 Medical Applications

Given some sensor data, we want to detect if a patient has a disease. This is like fault diagnosis, but in the medical domain. Use the methods from chapter 9, if the sensors produce a time series. Supervised methods are very useful here.

We may want to detect anomalies in imaging data (e.g. MRI scan). This is spatial data where we are looking for anomalous shapes. If you have a patient history of MRIs, you can treat it as spatiotemporal data. If you know what anomalous shapes are, provide them as supervision.

7 Text and Social Media Applications

Given a set of documents (e.g. user posts), we are looking anomalous documents. See chapter 8 for techniques applicable here.

Given a stream of documents (e.g. emails), we want to identify spam. This is basically supervised text classification.

Given a social network with links, identify the noisy/spam links. See chapter 12 for linkage outliers techniques.

Give an evolving network with text in nodes, identify anomalous regions. Again, see chapter 12 for community change techniques.

8 Earth Science Applications

Suppose we have sea surface temperatures and we are looking for unsual spatial variations, strange temperature shapes, unexpected changes in temperature, and the relationship of temperature to weather patterns. This is spatial or spatiotemporal data, so neighborhood methods, unusual shape detection, and change detection are useful here.

If you have multiple behavioral attributes (e.g. temperature, wind speed), you can handle each one independently and combine the anomaly scores.

9 Miscellaneous Applications

We may want to clean a dataset by removing outliers from it. The autoregressive models from chapter 9 are good for time series and PCA is good for multidimensional data.

Given entities with trajectories, we want to find anomalies. Trajectory methods from chapter 11 work well here. Streaming algorithms may be relevant here.

Given a set of images (or stream of images), we want to find unusual shapes, changes in the temporal pattern of images, and prediction of rare classes (assuming we have some labeled data). You can treat this as spatiotemporal data. Or, if you have good image representations (i.e. image embeddings), you can treat it as a multiple time series problem.

10 Guidelines for the Practitioner

Make sure to normalize your data, or techniques like proximity methods or linear models will not work. Give your attributes zero mean and unit variance.

Many outlier methods may find noise, not interesting outliers. If you can filter out noise with domain knowledge, do it.

It's rare that your dataset will be in an immediately usable format, so you will likely want to do some feature extraction.

If you have domain knowledge, incorporate it into your algorithm. This is a way to make up for the fact that we do not have supervision. You can do this by adjusting transition probabilities for a Hidden Markov Model, tweaking your cost function (e.g. if false positive is not as bad as false negative), designing distance functions, crafting good features, providing your own sequences as comparison tokens in discrete sequence processing.

If you have labeled data, use it.

Do data visualization (e.g. see chapter 3) to get an idea of what kinds of models would work.

Add a human in the loop and do active learning if you can. If you start with an unsupervised algorithm, you can pick out some examples for the human to label and then you can switch to a supervised algorithm.

Using outlier ensembles helps overcome the limitations of a single algorithm run.

Simple algorithms on large datasets tend to beat complicated algorithms. Don't underestimate k-nearest neighbors, the Mahalanobis method, and the isolation forest. The latter two should be ensembled (notice they are almost parameter-free, too).

Your choice of parameters can impact performance, especially when you have an unstable algorithm. LOF, for example, tends to struggle when you pick bad parameters.

The linear and nonlinear Mahalanobis methods work quite well, especially when they are run on kernel matrices. Ensemble them for even better performance.

The exact and average k-nearest neighbors work quite well, and usually beat LOF. They are robust across many datasets. The key problem here is scaling to large datasets, so you'll need to use indexing/pruning or with variable subsampling (see chapter 6). Rotated bagging also helps here.

Subspace histogram methods (e.g. RSHash and isolation forest) work very well. Their only downside is they tend to prefer outliers that lie at the interior regions of the dataset. They also struggle with clusters of anomalies.

If you use an ensemble of Mahalanobis, k-nearest neighbors, and isolation forest (each of them is also ensembled), you get a powerful method called TRINITY.

11 Resources for the Practitioner

The KDD Nuggets website links to some outlier detection resources. The ELKI repo has implementations of some algorithms. Python's scikit-learn implements a bunch of algorithms too.

12 Conclusions and Summary

We've talked about various applications of outlier analysis and suggestions for how to approach different outlier analysis problems. In short, use an ensemble of k-nearest neighbors, Mahalanobis method, and subspace histograms (e.g. isolation forest).