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) $\exp(-\frac{||\bar{X}_i - \bar{X}_j||^2}{\sigma^2})$, polynomial kernel $(\bar{X}_i^T \bar{X}_j + c)^h$, and sigmoid kernel $\tanh(\kappa \bar{X}_i^T \bar{X}_j - \delta)$. 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 $J = \frac{1}{2}||\bar{W}||^2 + \frac{C}{N}\sum_{i=1}^N \max(b - \bar{W}^T \Phi(\bar{X}_i), 0) - b$.

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.