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.