ANOMALY DETECTION

Lesson 5: High Dimensional Anomaly Detection

Learning objectives

You will be able to:

- Describe high dimensional anomaly detection
- Understand the curse of dimensionality
- Apply the subspace method with feature bagging
- Apply isolation forests
- Use Python* to perform anomaly detection on high-dimensional data

High dimensional anomaly detection

Introduction

- In previous lessons we illustrated various approaches to anomaly detection using datasets with one or two dimensions for clarity
- In reality, datasets often have a large number of dimensions $d >> 1$
- While the approaches presented previously may still work, some of them slow down significantly as d increases
- Furthermore, new challenges emerge in high dimensions that aren't problematic in 1D, 2D or 3D

The curse of dimensionality

A few of the problems faced in high dimensions

- Data sparsity
- Relevant vs. irrelevant features
- Combinatorial explosion

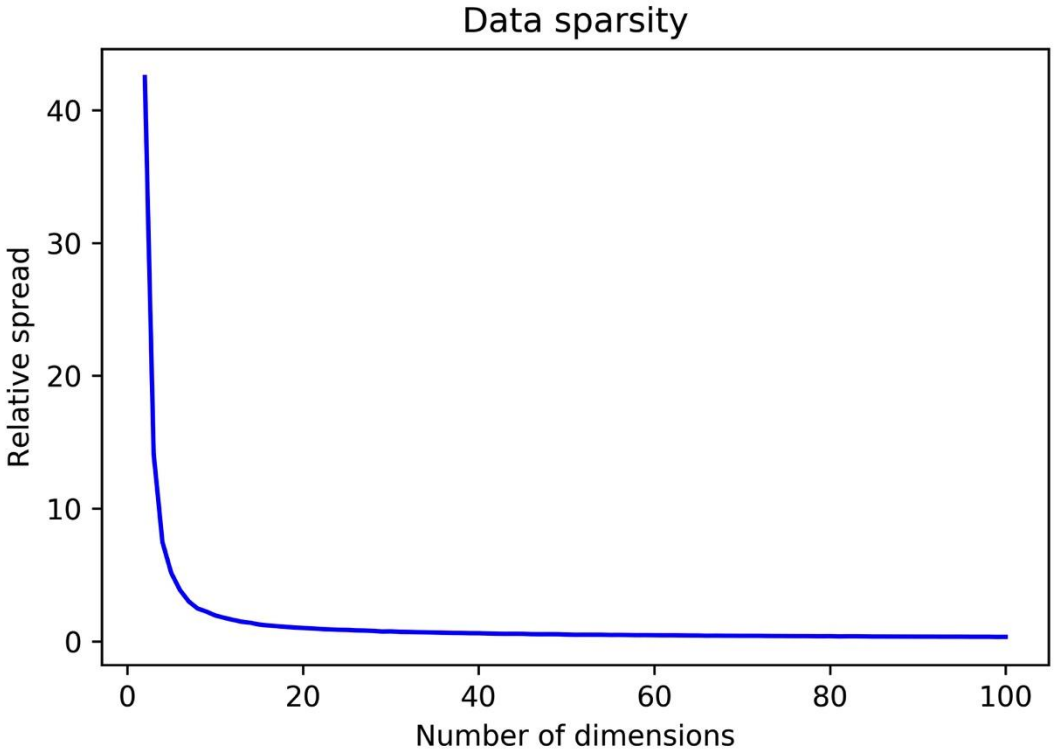
These problems plague other algorithms too (not just anomaly detection)

Data sparsity

- All pairs of points are almost equidistant in high dimensions
- More precisely, for a randomly chosen point, the relative difference between the furthest-neighbor distance r_{max} and nearest-neighbor distance r_{min} goes to zero as the dimensionality d grows:

$$\lim_{d \to \infty} \left(\frac{r_{max} - r_{min}}{r_{min}} \right) = 0$$

- In other words, for a given query point in high dimensions, almost all points are likely to lie in a thin shell around the query point



Relative spread

$$= \left(\frac{r_{\max} - r_{\min}}{r_{\min}} \right)$$

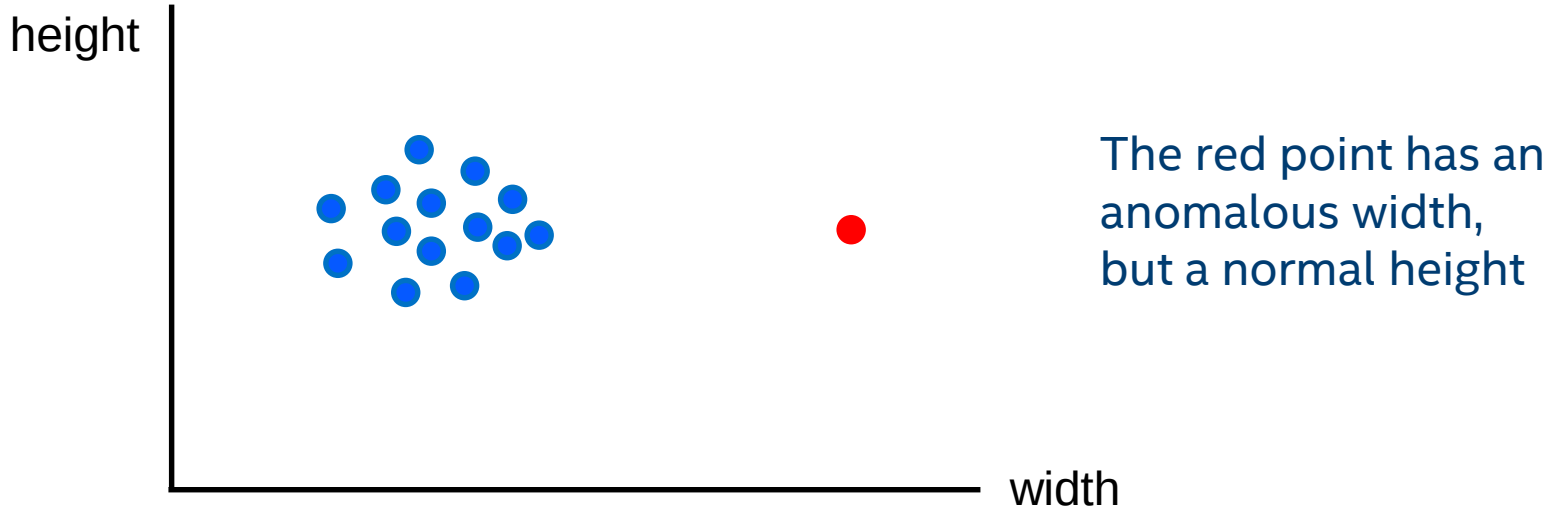
Data sparsity

- As a result, distance-based anomaly detection methods become challenging
- Consider the following definition for detecting an anomaly:
  - A point *P* in a dataset *T* is an anomaly if at least a fraction *f* of the points in *T* lies greater than distance *r* from *P*
- As the dimensionality increases, it becomes increasingly harder to choose the threshold distance *r* because of the shell effect
  - If *r* is a little bigger than the shell radius, no points are anomalies
    If *r* is a little smaller than the shell radius, all points are anomalies
- Therefore, *r* must be chosen with unrealistic accuracy to detect a reasonable number of anomalies

Relevant vs. irrelevant features

Anomalies are usually different only in some features (dimensions)

- These features are called “relevant”



The red point has an anomalous width, but a normal height

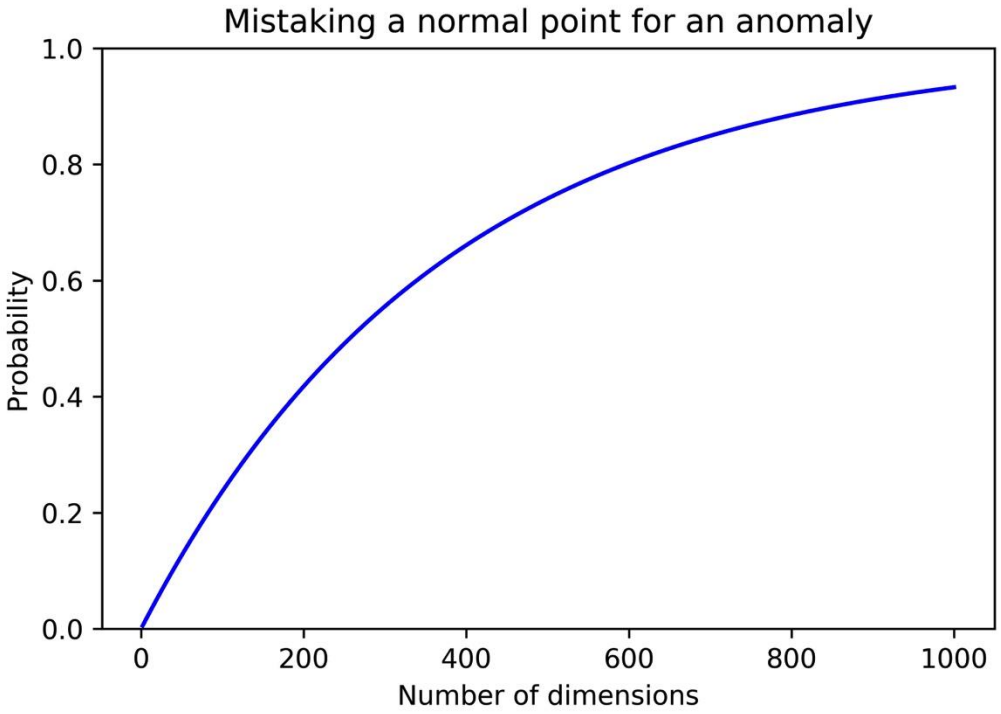
Relevant vs. irrelevant features

Irrelevant features add noise making it harder to detect anomalies

- Consider data that consists of some anomalies together with normal (non-anomalous) data that is generated from a Gaussian distribution
- For each normal data point there is a small probability of getting a result that is far from the mean, which would make it look like an anomaly
- Let's assume that a normal point looks like an anomaly if it is more than three standard deviations away from the mean

Relevant vs. irrelevant features

- For a Gaussian distribution, the probability of being within three standard deviations from the mean is $p = 0.9973$
- The probability of looking like an anomaly is $q = 1 - p = 0.0027$
- So for $d = 1$, the probability of mistaking a normal point for an anomaly is very small
- What happens as d increases?

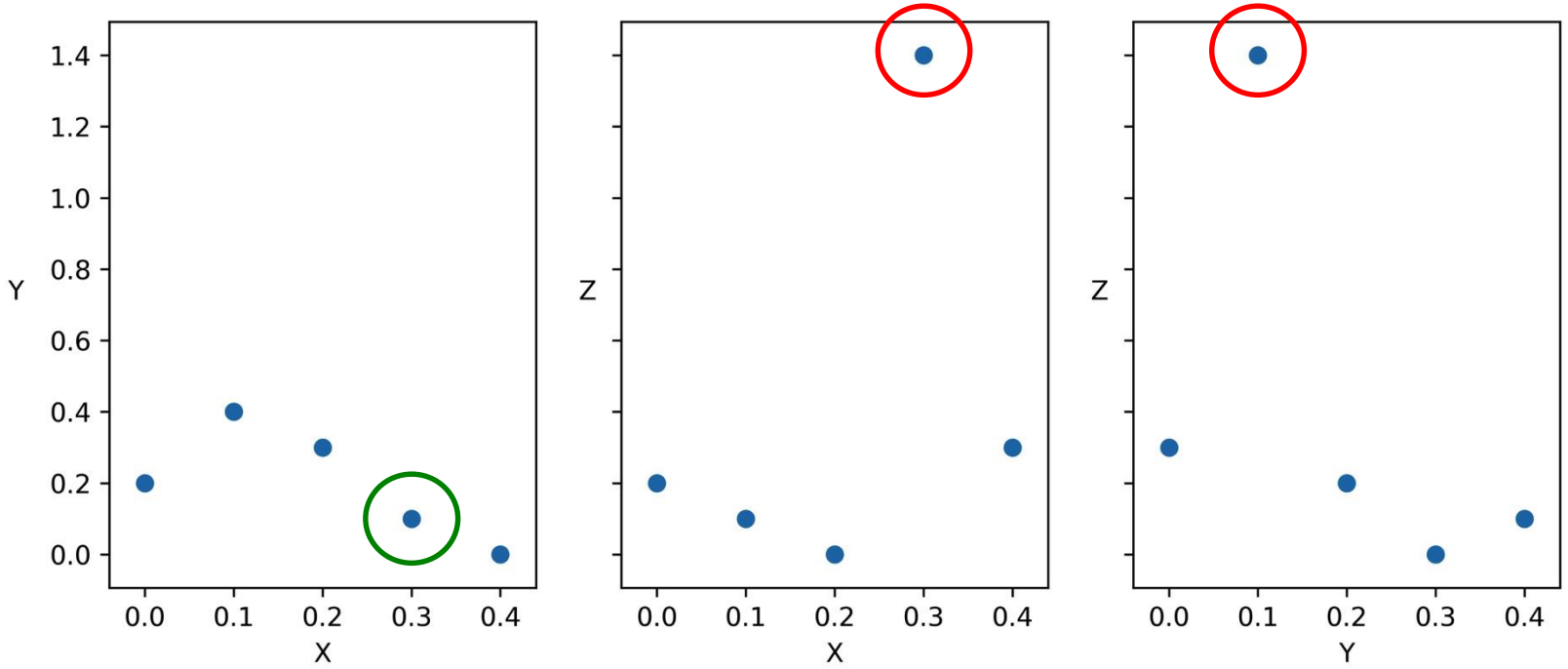

Mistaking a normal point for an anomaly
1.0
0.8
0.6
0.4
0.2
0.0
Probability
0
200
400
600
800
1000
Number of dimensions

Relevant vs. irrelevant features

- For d =1000, the probability that the point won't look like an anomaly in any of the dimensions is p^d =0.0670.
- In other words, it is very likely that is some dimensions, what is supposed to be a normal point will appear to be an anomaly
- Similarly, it is unlikely that the anomalies will have anomalous values in all dimensions. In some (or even many) dimensions, they will look like normal data
- Anomalies are embedded in lower-dimensional subspaces ("relevant features")
- To complicate matters further, different anomalies may be embedded in different subspaces

Relevant vs. irrelevant features

- Because of the potential confusion between anomalies and normal data, it is not appropriate simply to use all dimensions to detect anomalies. Such an approach—for example, a distance metric that includes all dimensions available—will lead to poor results
- The challenge is to find the relevant subspaces (features) that lead to an effective anomaly detection algorithm


Y
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
X
Z
0.0
0.1
0.2
0.3
0.4
X
Z
0.0
0.1
0.2
0.3
0.4
Y

Combinatorial explosion

Exhaustive search is impractical

- Imagine looking for a relevant subspace in $d = 1000$ using a grid
- Divide each dimension into 10 bins
- Have to scan 10^{1000} cells
- Also need $>10^{1000}$ data points to reach any statistically valid conclusions

Subspace search

A note of caution

- Comparing a data point with many different subspaces can introduce statistical bias
- Multiple comparisons problem: can find a subspace that looks good by chance
- Scoring problem: can inappropriately favor a subspaces by using a biased metric (e.g., using a distance-based score with subspaces of different dimensionality)

The subspace method

A few approaches to find appropriate subspaces

- PCA (principal component analysis)
- Grid-based search (beware: combinatorial explosion possible!)
- Cluster-based statistics
- Random sampling of subspaces
  - feature bagging

Whatever method you use, remember: detecting anomalies is only a first step. Explaining why they are anomalies is the main goal. If you can't explain your findings, your subspace method is unhelpful.

Feature bagging

An ensemble method

- An approach for combining outliers from many different subspaces
- Perform anomaly detection in random subsets of the attributes (=subspaces)
- Combine all results to achieve an overall ranking or score

Feature bagging

The implementation for *d*-dimensional data

- An iterative process. For iteration *s:*
- Select an integer *m* at random from (d/2)* to (d-1)
- Select *m* features at random without replacement from the dataset producing an *m*-dimensional dataset $T_{s,\,m}$
- Apply the anomaly detection algorithm A_s to $T_{s,\,m}$ to score each data point
- Combine scores from the different iterations to get an overall result for each point

Feature bagging

Comparing apples with apples

- Since each *m*-dimensional subspace contains a different number of features, the scores must be normalized to make a meaningful comparison
- If normalization is done properly, it allows for extra freedom: the anomaly detection algorithm A_s can be different for each iteration
- The simplest approach, however, is to use the same anomaly detection algorithm for all iterations that returns a normalized score

Feature bagging

Two ways to combine scores from different iterations

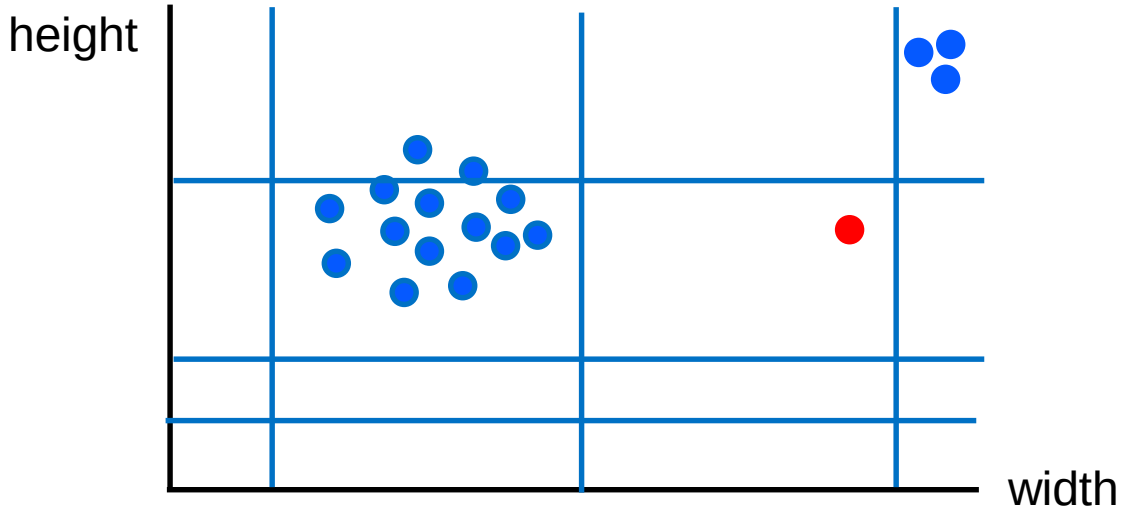
- Breadth-first: The top-ranked anomalies from all iterations are ranked first, followed by the second-ranked outliers (with repetitions removed), etc.
  - Different tie-breaking procedures between anomalies within each rank may lead to slightly different results.
- Cumulative-sum: For each point, the anomaly scores from the different iterations are added together. The overall score is determined by the cumulative sum.

Even though each iteration may give not-so-great results, the combination provides for good anomaly detection

Isolation forests

Introduction

- An ensemble method: an isolation forest is a combination of isolation trees
- In an isolation tree, the data is repeated divided with axis-parallel cuts at randomly chosen positions for randomly selected features.
- The goal is to divide the data into nodes with fewer and fewer points until a singleton node containing one point is reached (the “leaf” of the tree).
- We expect the tree branches containing anomalies to be less deep than those for normal points, because anomalies are located in sparse regions.



Anomaly (red point) is isolated

Isolation forests: how to create a tree

The basic algorithm

- Start with a root node containing all the points in the dataset
- Create a list *L* of nodes for further splitting
- Initially, the only member of *L* is the root node
- The steps on the next slide are repeated to create the isolation tree until the candidate list *L* is empty

Isolation forests: how to create a tree

- Choose a node N from L at random and remove it from L
- Choose a feature f at random; value of feature at a point is denoted by x_f
- Split the data in N into two parts N_1 *and* N_2 at a random value v along the feature
  - For all points in N_1: $x_f \leq v$
  - For all points in N_z: $x_f > v$
  - v is chosen uniformly at random between the minimum and maximum values of x_f for points in N
- If N_1 and N_2 contain more than one point then add them to L. Otherwise, designate the node(s) with only one point as a leaf
- Algorithm ends when have only leaves and no further splitting of nodes is possible

Isolation forests

Scoring

- Anomalies are typically isolated into leaves in fewer splitting's than normal points*
- Therefore, length of path from the root to the tree (number of splitting's) is used as the score—anomalies have small scores
- Repeat algorithm multiple times and average scores to get final result

Note: the isolation forest implementation in scikit-learn uses a different scoring convention from that found elsewhere (see Python* notebook accompanying lecture).

Isolation forests

Improving the basic algorithm

- If tree is grown to full height, the algorithm is parameter-free, but it can take a long time to run
- For greater efficiency, can stop when node has *n* (> 1) points
  - Use an extrapolation formula to estimate final length
- Can also split data into train and test samples and construct tree with smaller training set

CONCLUSION

Use Python* for anomaly detection

Next up is a look at applying these concepts in Python*

- See notebook entitled *High_Dimensional_Anomaly_Detection_student.ipynb*

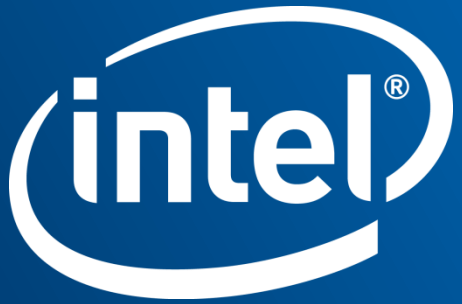
Learning objectives recap

In this session you learned how to:

- Understand high dimensional anomaly detection
- Apply the subspace method with feature bagging
- Apply isolation forests
- Use Python* to perform anomaly detection on high-dimensional data

References

- *Outlier Detection in High-Dimensional Data* by A. Zimek, E. Schubert, H.-P. Kriegel (2013)
- *Outlier Detection for High Dimensional Data* by C. C. Aggarwal, P.S. Yu (2001)


intel®