ANOMALY DETECTION

Lesson 6: Supervised Anomaly Detection

Learning objectives

You will be able to:

- Describe supervised anomaly detection
- Implement cost-sensitive learning
- Apply adaptive resampling and boosting methods
- Use Python* to perform supervised anomaly detection

Supervised anomaly detection

Introduction

- In previous lessons, we discussed various approaches to unsupervised anomaly detection. That is, how to detect anomalies when you don't have any additional information about the data.
- One issue that arises with unsupervised anomaly detection is that you often label noise as anomalies because you don't know any better.
- But what if you do know better? More specifically, what if you have examples of anomalies and normal points that you could encounter?

Supervised anomaly detection

Knowledge is power

- If you have (domain-specific) information about your anomalies, use it.
- If you are really lucky, you might have a simple selection criterion:
  - All heights above 6’ 5” are anomalies
- More often, information consists of examples of normal data and anomalies
- Extra information usually improves anomaly detection accuracy significantly

Whenever possible, use supervised methods.

Supervised anomaly detection: a classification problem

A special case of classification

- Examples of anomalies and normal points = training data
- Unlabeled points = test data
- Therefore, the many classification techniques available (supervised machine learning) can be used for anomaly detection

Supervised anomaly detection: a classification problem

A classification problem with specific challenges

- Class imbalance
  - Anomalies by definition are rare, so there will be few examples
- Contaminated normal data
  - Only anomalies labeled; normal class contaminated by unlabeled anomalies
- Partially labeled data (“semi-supervised anomaly detection”)
  - Typical case: only normal class labeled

Class imbalance

An illustration of the problem

- Consider screening test that uses x-ray scans to detect a rare cancer
- For a typical population tested, 99% of patients are healthy and 1% has cancer
- We want develop an algorithm to classify the scans as normal or anomalous
- Naturally, we want our algorithm to performance as well as possible

Must evaluate algorithm performance with care

Class imbalance

A useless anomaly detector

- Label all scans as normal without any analysis
- Confusion matrix (for 100 scans):
- Accuracy = (TP+TN) / Total

TP = true positive; TN = true negative

Total = all data

- Accuracy = (0 + 99)/100 = 99%
- Very high accuracy, but never find a sick patient

Predicted

True	Normal	Anomaly
Normal	99	0
Anomaly	1	0

Class imbalance: effective anomaly detection

Evaluate the algorithm appropriately

- Overall accuracy is not a useful measure by which to judge the algorithm when the anomalies are a small fraction of the overall data
- Typically, it is more costly to misclassify an anomalous point than normal data*
  - For the cancer example: a false positive (normal point misclassified) will lead to additional, diagnostic tests, which hopefully will correct the error
  - A false negative (anomalous point misclassified) will lead to overlooking the disease at an early, treatable stage and perhaps ignoring it until it is too late to treat
- This cost should be included when evaluating the effectiveness of the algorithm

Class imbalance: effective anomaly detection

Algorithm should take into account cost of making a mistake (misclassification)

- Use a cost-weighted approach when implementing algorithm
- Two main ways to do so:
  - Cost-sensitive learning
  - Adaptive resampling

Cost-sensitive learning

- Classifier is trained using a weighted accuracy over the various classes
- Consider two classes—normal (denoted by 0) and anomalies (denoted by 1)
- In this case minimize the weighted accuracy given by:

$$J = c_0 n_0 + c_1 n_1$$

c_i is cost of misclassifying a point from class *i* (with *i = 0, 1)*

n_i is number of misclassified instances from class *i*

- The objective function above can be generalized to multiclass systems (different types of anomaly)

Cost-sensitive learning

- The costs for each classes is often domain-dependent and should be specified as part of the inputs for the algorithm
- However, if no information about the costs is available, you have to make an educated guess
- One possibility is $c_i = 1/N_i$ where N_i is the number of points in class *i*
  - The weighted accuracy then depends on the fraction of misclassifications for each class offsetting the class imbalance
- See Python* notebook for an example

$$J = \frac{n_o}{N_0} + \frac{n_1}{N_1}$$

Adaptive resampling

Another way to address class imbalance

- Non-uniform sampling of the training set to favor the rare, anomaly class
  - Oversample anomalies
  - Undersample normal data
  - Or do both
- Sampling probabilities are chosen proportional to misclassification costs
  - Oversampling is done with replacement
  - Undersampling can be done with or without replacement
- Algorithm is trained on resampled dataset

Adaptive resampling

The case for undersampling

- Undersampling often works better than oversampling
- Typically, use most or all of anomalies and a small number of normal data points
- As a result:
  - Small training dataset, so training is fast
  - Since the training dataset is small, can construct multiple training datasets and average over the results leading to a better anomaly detector
- However, the effectiveness of undersampling is limited by how sensitive the classifier is to discarding normal data. When the classification model is mainly dependent on the anomaly examples, undersampling works well*

Adaptive resampling vs. cost-sensitive learning

- Adaptive resampling (AS): select some of labeled data using cost-based weights to create a training dataset and then treat all points in this sub-dataset equally
- Cost-senstive learning (CSL): use all of the labeled data as a training dataset and use cost-based weights for classification
- If the cost functions are the same, the two approaches should produce similar results. However:
  - CSL keeps all the data, so more accurate if you analyze the data just once
  - AS more efficient because it works with smaller datasets (for undersampling)
  - If use AS to average over many training datasets, usually better than CSL

Boosting methods

Learning from your mistakes

- The process of converting a family of weak learners into a strong learner
- Weak learner = a classifier only a little better than random guessing
- Strong learner = a classifier highly correlated with the correct classification
- The weak learners are trained sequentially, each trying to correct the mistakes of its predecessor

AdaBoost

A popular boosting algorithm

- Learning occurs through a weighted-error approach
- For each round:
  - Weight is increased for previously misclassified examples
  - Weight is increased for correctly classified ones
- Predictions for test data are obtained from a confidence-weighted majority vote of the learners from all rounds
- Typically used with decision trees, but works with any weak learner
- Can be modified to include cost-sensitive learning

CONCLUSION

Use Python* for anomaly detection

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

- See notebook entitled *Supervised_Anomaly_Detection_student.ipynb*

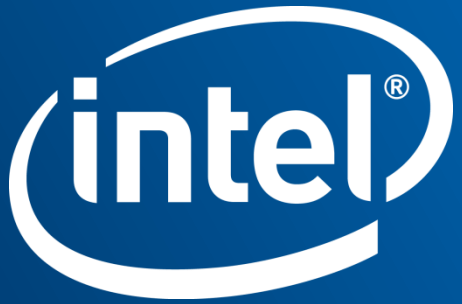
Learning objectives recap

In this session you learned how to:

- Describe supervised anomaly detection
- Implement cost-sensitive learning
- Apply adaptive resampling and boosting methods
- Use Python* to perform supervised anomaly detection

References

- *Outlier Analysis* by C.C. Aggarwal (Springer 2013)
- *Introduction to AdaBoost* by N. Nikolaou
- *Supervised Machine Learning: A Review of Classification Techniques* by S.B. Kotsiantis, *Informatica* **31** 249-268 (2007)


intel®