A step by step guide to HW4

Daniel Barbará

Goals

- Implement the StrOUD algorithm, using LOF as the strangeness function
- Handle Signal data collected from a device that controls a centrifuge (see below)o Design and Engineer Features from the Data.
- Find anomalous and normal data in the test set

StrOUD (Strangeness-based Outlier Detection)

- Doesn’t create a model per se.
- Uses transduction instead of induction
  - That means it doesn’t try to estimate a global $P(N|\mathbf{x})$
  - But rather check for fitness in a sample distribution
  - The sample is the model
- Uses the measure of strangeness for fitness
  - Strangeness is a function that measures how unique a point is with respect to the sample distribution

Strangeness in HW4

- We will use the Local Outlier Factor as strangeness (LOF)
- **Do not use the python implementation of LOF!**
  - It does implement the whole anomaly detection algorithm by choosing a threshold.
  - It's NOT what we want to do!
- Instead code your own LOF function. Two choices:
  - Use the definition on the book.
  - Use the definition on the slides/tutorial in the class website.
- LOF measures the density of a point as compared to the density of its closest ***k*** neighbors (parameter).

Parameter *k*

- As always, you have to tune it.
- Cross validation in AD is complicated (but possible)
- Alternatively, split data and use several random samples as baseline (training)

How to: step 1

- Create your baseline (training data)
    - It has to be ALL normal
    - For that, take data (signals) from folders ModeA, ModeB, ModeC, and ModeD.
        - Some of the files in there are empty. Just ignore them.
        - Each file is a signal (data point) with a lot of features.
        - Each feature is a measurement or sample of the signal.
    - Since there are enough signals, you can create one or more random samples (useful for validation)
    - Run **each** signal through the Fast Fourier Transform function in python (FFT)
        - That produces a vector of the same number of features that the original signal has.
    - Keep the transformed data as your baseline sample.
    - Your sample is the baseline or training data.

Step 2

- Compute the LOF of each point in the baseline with respect to the other points in the baseline.
- Put those LOF measures in a list and **sort that list** in ascending order.
- Keep that list: that is your **strangeness training list.**

Step 3

- Now produce a test set by taking random signals (files) from ModeA, ModeB, ModeC, and ModeD folders (normal) and adding random signals (files) from ModeM (anomalies).
- Keep the list balanced
- Make sure you use data from ModeA, ModeB, ModeC, and ModeD that you didn't use in the training.
- As before, run each signal through FFT and keep the transformed points.
- This is your test data. Notice that since you know which points are normal and which are anomalies, you can form a list of labels. Your test data and your labels have to be in the same order.
- **Note:** since you took a sample of the normal data to create the training and a sample of normal/abnormal data to create the test, you have the ability to run multiple experiments. Useful for validation.

Step 4

- For each data point in the test set, compute the LOF **with respect to the points in the training set.**
- Form a list of LOF (strangeness values) for the test set. DO NOT SORT THIS ONE! Preserve the order you had in the test set.

Step 5

- Take your test strangeness list and for each measure on it:
  - Find its place on the **strangeness training list. That is find how many measures in the training strangeness list are higher or equal to the one in the test strangeness list you are considering.**
  - Note: since you are placing the test strangeness list value in the training strangeness list, there will be at least one that is equal to it.
  - Call that number **b.**
  - Divide **b by N+1,** where **N** is the size of your training strangeness list.
  - Call that ratio the p-value of that test point.
  - Take the test point strangeness out of the list and proceed with the next one.
  - At the end, you have a list of test p-values in the same order of the test points.

Step 6

- With the list of test p-values, you can compute the AUC of the corresponding ROC.
  - To build the ROC, consider confidence levels from 0 to 1.0 at chosen intervals
  - For each confidence level, compare each p-value with 1-confidence level.
  - If p-value < 1-confidence level, the test point is an anomaly at that confidence level. Otherwise, it’s normal.
  - Compare the predictions to your truth (test labels) and you can compute the False Positive rate and True Positive rate at that confidence level (one point of the ROC).
  - With the ROC, you can compute the AUC.
  - Hint: python has pre-built ROC and AUC calls. Figure them out or do your own.

Step 7

- For different training/test sets compute AUC (you can try cross validation, you just have to be clever about it).
- Compare AUC for different choices of ***k.***
- Pick a winner ***k*** and use it for your next step.
- Report results in the submission.
- You must do this in some form. Otherwise, points will be taken off.

Step 8 (final)

- Using a training set, compute the training strangeness list, using the chosen ***k***.
- Using the **provided test set of signals:**
  - FFT them
  - Compute the p-value of each test point as indicated before.
  - Submit the list of p-values to Miner2
- Things to watch:
  - Careful how you read the signals. Python will not read them in the given order unless you force it.
  - The Miner2 has the truth list **in that order.** If you scramble it, it will most likely give you a 0.5 score.
  - The Miner2 computes AUC from the p-values you provide. No need to do it yourself.
  - Only 5 submissions per 24-hour period.

THE END