DATA QUALITY TESTING with LSTM Autoencoder

Name: Shlok Gopalbhai Gondalia
Class: CS498
Email: shlok@rams.colostate.edu
Semester: Spring 2020

Quality Testing for Time Series Data

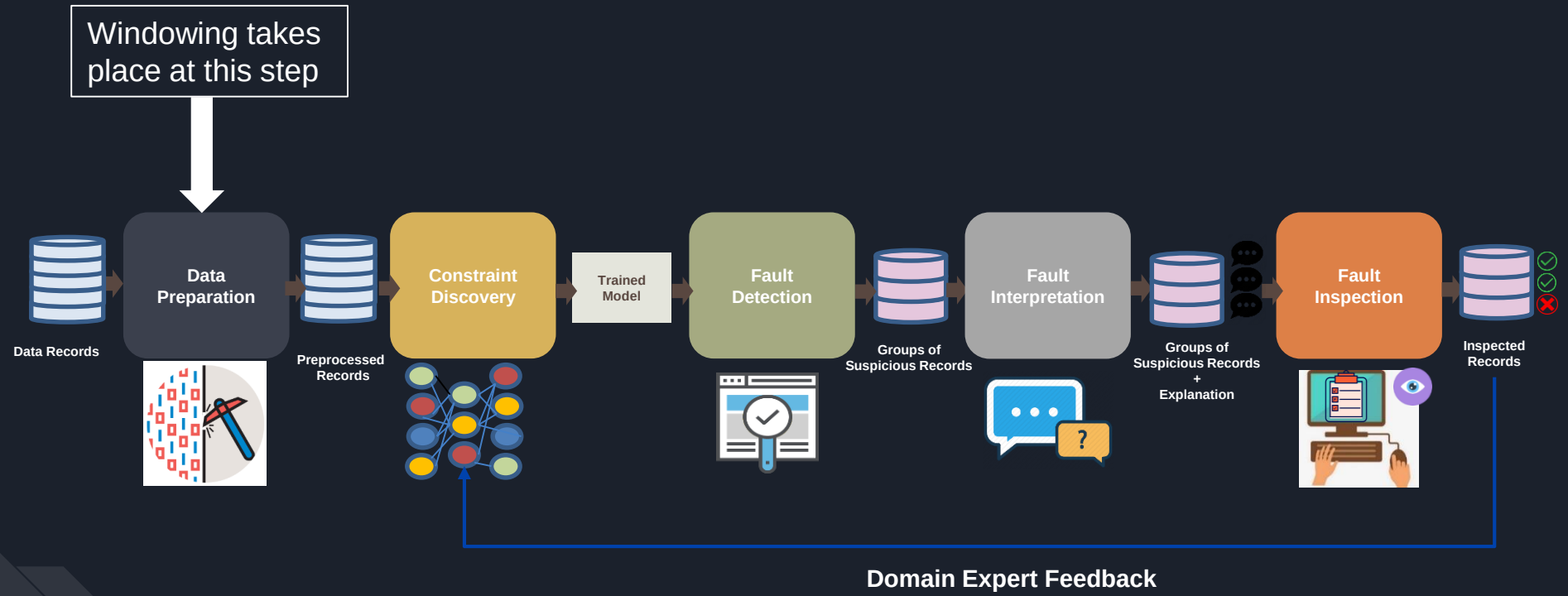
Problem with existing approaches

- Can generate false alarms while reporting constraint violations
- Use brute force windowing
  - It can result in missing constraints if the window size is small
  - It can increase the computational complexity of a network if the window size is large
- Don't provide explanations behind the constraint violation

Our Goal

- Develop an automated data quality test approach
  - Uses an LSTM autoencoder
  - Works with both univariate and multivariate time series
- Discover complex constraints within
  - Attributes of a record
  - Multiple records in a sequence
- Report as suspicious those sequences/data which violate the constraints

ADQuaTe3 Overview



My Role as the Evaluator

- Evaluate the effectiveness and performance of the approach after the Fault Inspection phase
- Compare our approach with the best existing approach using the metrics
  - F1_T Value
  - Time Taken for the dataset to run
  - Growth Rates of F1_T Value
- Use Mutation Analysis to insert artificial faults in datasets to evaluate the detection ability of the approaches (Slide 7)

F1_T Value

$$F1\_T = 2 \times \frac{(precision\_T \times recall\_T)}{(precision\_T + recall\_T)}$$

where:

$$precision\_T = \frac{TPR\_T}{(TPR\_T + FPR\_T)},$$

$$recall\_T = \frac{TPR\_T}{(TPR\_T + FNR\_T)},$$

$$FPR\_T = \frac{FP\_T}{N_T},$$

$$TPR\_T = \frac{TP\_T}{P_T},$$

$$FNR\_T = 1 - TPR\_T, \text{ and}$$

$$TNR\_T = 1 - FPR\_T.$$

- FP_T => No. of valid time series incorrectly detected
- N_T => No. of actual valid time series
- TP_T => No. of faulty time series detected
- P_T => No. of actual faulty time series

$$F1\_TGR = \left(\frac{F1\_T_{NR}}{F1\_T_1}\right)^{\frac{1}{NR}} - 1$$

Mutation analysis

- Seed different types of faults into randomly selected records in the datasets
- Faults are based on mutation operators M1 to M5 with the goal of violating constraints over the features

Mutation Operators:

- M1 => Add noise
- M2 => Horizontal shift
- M3 => Vertical shift
- M4 => Re-scale
- M5 => Add dense noise

Datasets Used For Results

- Two different datasets
  - Yahoo synthetic servers (univariate)
  - Shuttle dataset (multivariate)
- Ran the mutated datasets using ADQuoTe3 with both
  - Brute force windowing
  - Autocorrelation windowing

Completed Tasks

- Improved python script **`testScriptTuningLSTMAutoencoder.py`** to find the best brute force windowing
  - Implemented automated brute force window finding
  - Before it was done manually
- Wrote python scripts to
  - Generate line plots between F1_T values and no. of runs
  - Calculate the total time taken by a dataset to run
  - Calculate F1_TGR
- Plotted line graphs for the F1_T value

Modified Python Script

testScriptTuningLSTMAutoencoder.py

- Finds the best window size based on the highest F1_T value
  - Loop runs a dataset for a range of window size (10 to 50)
  - Prints the result

New Python Scripts

- Rewrote the script (**`testScriptPlotTrend.py`**) from last semester
  - Generates the line plot for a single original or mutated dataset
- Wrote a new script called **`testScriptMutation.py`**
  - Generates the line plot for a group of mutated datasets

- Both these plots are used for comparing F1_T values for these two approaches
- Both these plots have same x and y axes

testScriptPlotTrend.py

- Creates a line plot from a single dataset using the defined window size and auto window size
- Now uses SQL Query to parse the F1_T values of a particular dataset from the results
- X-axis shows the number of times the dataset is run
- Y-axis is F1_T Values
- Calculates the time taken by a dataset to run
- Calculates the F1_T growth rate

testScriptMutation.py

- Works the same way as **`testScriptPlotTrend.py`** with mutation
- Mutations:
  - Takes a group of mutated datasets instead of single to plot
  - Calculates the average time taken for the group of dataset
    - Tells us about the efficiency of each approach
  - Calculates the average F1_TGR
    - Tells us how fast a model is improving in each run
  - Creates a .csv file with Time Taken and F1_TGR values

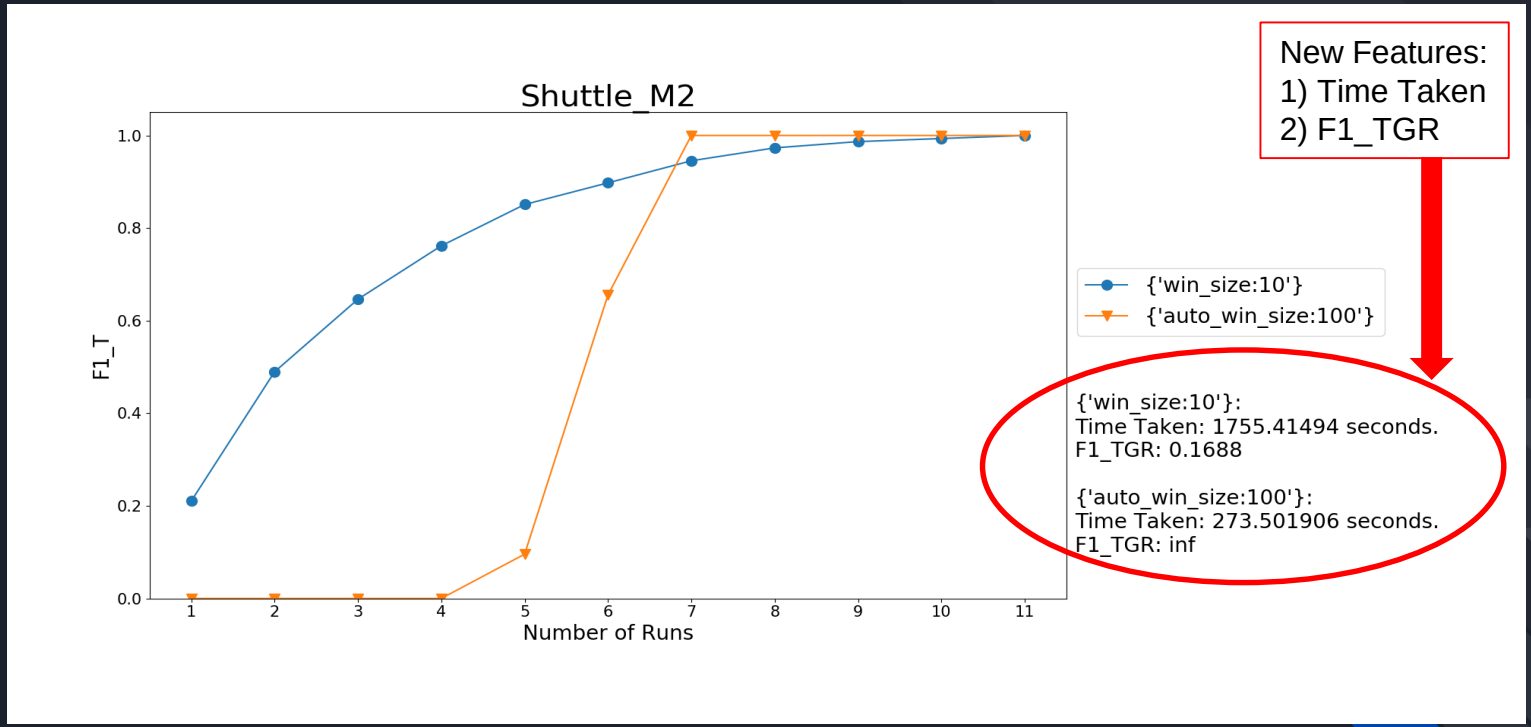
testScriptMutation.py
Sample CSV File

M4_Datasets_best_win_size => brute force windowing
M4_Datasets_auto_win_size => autocorrelation windowing

Attributes/Datasets	M4_datasets_best_win_size	M4_datasets_auto_win_size
Average Time Taken in Seconds	714.0229413	151.4716103
Average Growth Rates	0.007562277	0.044868257

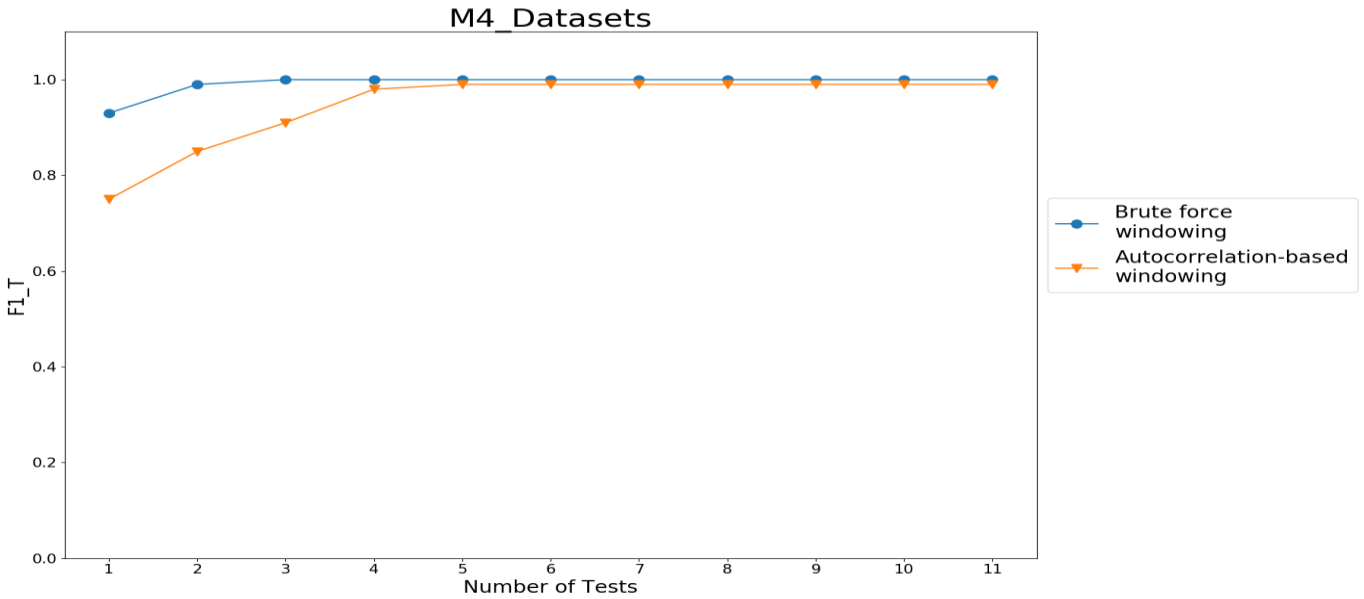
- M4_Datasets ran around 4.7 times faster in autocorrelation windowing than in brute force windowing
- For M4_Datasets, model improved around 5.9 times faster in autocorrelation windowing than in brute force windowing

testScriptPlotTrend.py: Sample Plot



- Shuttle_M2 is 99% effective and is around 6.4 times faster for autocorrelation windowing

testScriptMutation.py: Sample Plot



- M4_Datasets are around 98% effective in autocorrelation windowing

Plots help with Visualization

- Clearly see the difference between the two approaches.
- X-axis represents the number of runs and Y-axis represents the F1_T values
  - It makes it easy to compare each run with the F1_T value
- It helps us to see how effective the new approach is in finding faulty records

Results

- ADQuaTe3 approach is around 98% effective compared to the brute force windowing
- We can see that by comparing the blue and the orange lines from the graph
- This new approach is also approximately 3 times faster than brute force windowing approach
- Growth Rate (F1_TGR) also tells us about the accuracy of this approach and how it improves after retraining the learning model

Reflection: What can I improve?

- Enhance writing and presentation skills
- Get more in-depth knowledge about ADQuaTe
- Learn how to read a research paper systematically

Next steps

- Read different research papers related to our work
- Understand the details of the approach and tool implementation to interpret the results better