**Concept Change Detection – Write-Up**

Bin Gao, Bin Yan

# Part I Numeric change detection

Numeric change detection uses the sliding window size of 50, and increments the sliding window by one number at a time. The algorithm will detect the first position that falls out of the calculated confidence interval. If the detected change persists for at least window size, it’s confirmed to be a change and the first position read that causes the statistics to fall out of the calculated confidence is outputted. This is based on the assumption that there is only one change in a test file.

If the file doesn’t have enough numbers after the first detected position to confirm that change has persisted for at least window size, the program will check if the change has persisted for more than half of the window size at the end of execution. If so, the suspected position is outputted, since the test file doesn’t have enough data to overthrow the suspected change, and it’s based on the assumption that there is only one change in a test file. Files that are sufficient long should not have such problem.

Two confidence intervals are calculated in the algorithm, the mean confidence interval based on the t-test and the variance confidence interval based on the “Bayesian perspective”, <http://hdl.handle.net/1877/438>. Since mean change will always cause variance change, but variance change may not cause mean change. At any moment of the execution, if the program finds both potential mean change and potential variance change, mean detection will dominate the execution, but variance detection will still be running in the background. It means if the program finds both mean change and variance change at the end of execution, mean change position will be outputted.

Because large variance has a large confidence interval, and small variance has a small confidence interval, detecting variance going up is usually faster than detecting variance going down. It means detection of variance going down has some delays. Therefore, when variance moving down is detected, the program uses a backtrace procedure to pinpoint the more accurate position of variance moving down. The backtrace procedure basically uses the same algorithm to detect the variance moving up in the reverse direction, since detecting variance moving up is more accurate.

# Part II Categorical change detection

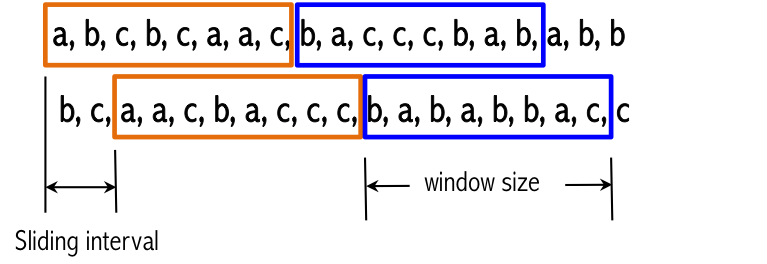
**1 Statistics**

* Binomial distributions: **Chi-square test**   
  (Also tried binom\_test[[1]](#footnote-1). The results were the same.)
* Multinomial distributions: **Chi-square test with Yates’ correction for continuity** (When the frequency of any element is 0, not valid, switch back to Chi-square)

The statistics used for Categorical data are Chi-square test and Chi-square test with Yates’ correction for continuity[[2]](#footnote-2). First, the frequencies of each element in two sliding windows are calculated. Then two arrays of frequencies are input in to scipy.stats.chisquare(). If the p value falls below the threshold, then a change has occurred. However, there is a problem with this statistical test. The Chi-square test is invalid when the observed or expected frequencies in each category are too small. A typical rule is that all of the observed and expected frequencies should be at least 5. For multinomial distributions (data with three or more categories), it is very likely that the frequency in one category is below 5. Therefore, for multinomial distributions, Chi-square test with Yates’ correction for continuity is used.

**2 Implementation**

***Window size and*** ***sliding interval***



Window size = 40

Sliding interval = 5 to detect a change; Sliding interval = 1 when locate the exact change position

***Thresholds***

We have to choose a very small p value (p = 0.0005) in order to avoid false positive. However, when locate the exact position of change, we combine the results with a larger threshold (p=0.05).

***Locating the change***

Step 1: Use a small threshold (p=0.0005) to detect whether a change has occurred, mark the location between two sliding windows as *start.*

Step 2: Move backward for two more window sizes from the *start*, use a larger threshold (p=0.05) and smaller sliding interval (size=1) to find the exact the location, mark the end of the second window as *end*.

Step 3: Exact location = (start + end)/2

Due to the space limitation, details cannot be described here. If necessary, please contact us for more details regarding the logic and intuition behind this implementation.

***Data stored in memory***

The size of data sample that needs be to stored in memory = 3 × window size

**3 Test Files**

Use np.random.choice(['i','j','k'], size=139, replace=True, p=[0.1,0.8,0.1] to generate test files.

**4 Results and Discussion**

We tested on 18 files. We are able to (1) detect whether there is a change or not with a success rate of 90%; (2) Locate the change position within ±10 for five out of seven testing files.

We had to fine-tune the window size, sliding interval and p value threshold to avoid false positive as well as to match the change location. The results are very sensitive to these parameters. In general, Chi-square is somewhat inaccurate when expected numbers are small. It is suggested that the sample size should be at least 1000[[3]](#footnote-3). As a consequence, there is possibility of overfit.

1. See the commented code: p = binom\_test(freq, n=None, p=freq\_old[0]/sum(freq\_old)) [↑](#footnote-ref-1)
2. http://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.stats.chi2\_contingency.html [↑](#footnote-ref-2)
3. http://www.biostathandbook.com/small.html [↑](#footnote-ref-3)