# Programming Assignment #1

***COEN 281 Pattern Recognition and Data Mining***
***Department of Computer Engineering***
***Santa Clara University***

Dr. Ming-Hwa Wang — Winter Quarter 2019
Phone: (408) 805-4175 — Email address: m1wang@scu.edu
Course website: http://www.cse.scu.edu/~mwang2/mining/
Office Hours: Tuesday & Thursday 9:00am-9:30am

***Due date***: Midnight January 20, 2019

## *Isolation Forest* (200 points)

Please implement an anomaly detection program which reports top-*k* anomalies from *n* *d*-dimensional input CSV format data using isolation forest in C/C++/Java. The term isolation means "separating an instance from the rest of the instances". Since anomalies are "few and different" and therefore they are more susceptible to isolation. In a data-induced random tree, partitioning of instances are repeated recursively until all instances are isolated. This random partitioning produces noticeable shorter paths for anomalies since a) the fewer instances of anomalies result in a smaller number of partitions – shorter paths in a tree structure, and b) instances with distinguishable attribute-values are more likely to be separated in early partitioning.

The anomaly score $s(x, n) = 2^{-E(h(x))/c(n)}$ where $c(n) = 2H(n-1) - 2(n-1)/n$ is the average of $h(x)$ given $n$, $H(i)$ is the harmonic number as $\ln(i) + 0.5772156649$ (Euler's constant), and $h(x)$ is the path length. $E(h(x))$ is the average $h(x)$ of from a collection of isolation trees. $0 < s \le 1$ for $0 < h(x) \le n - 1$. If $s$ close to 1, then they are definitely anomalies, if $s$ much smaller than 0.5, they are normal instances, and if all instances return $s \approx 0.5$, then the entire sample doesn't really have any anomaly.

The algorithms are listed below:

```
Algorithm 1: iForest(X, t, ψ)
Inputs: X - input data, t - number of trees, ψ – subsampling size
Output: a set of t iTrees
 1: Initialize Forest
 2: set height limit l = ceiling(log₂ψ)
 3: for i = 1 to t do
 4:   X' ← sample(X, ψ)
 5:   Forest ← Forest ∪ iTree(X', 0, l)
 6: end for
 7: return Forest

Algorithm 2: iTree(X, e, l)
Inputs: X - input data, e - current tree height, l – height limit
Output: an iTree
 1: if e ≥ l or |X| ≤ 1 then
 2:   return exNode{Size ← |X|}
 3: else
 4:   let Q be a list of attributes in X
 5:   randomly select an attribute q ∈ Q
 6:   randomly select a split point p from max and min values of
     attribute q in X
 7:   X_l ← filter(X, q < p)
 8:   X_r ← filter(X, q ≥ p)
 9:   return inNode{Left ← iT ree(X_l, e + 1, l),
10:                 Right ← iTree(X_r, e + 1, l),
11:                 SplitAtt ← q,
12:                 SplitValue ← p}
13: end if

Algorithm 3: PathLength(x, T, e)
Inputs : x - an instance, T - an iTree, e - current path length;
         to be initialized to zero when first called
Output: path length of x
 1: if T is an external node then
 2:   return e + c(T.size) {c(.) is defined in Equation 1}
 3: end if
 4: a    T.splitAtt
 5: if x_a < T.splitValue then
 6:   return PathLength(x, T.left, e + 1)
 7: else {x_a ≥ T.splitValue}
 8:   return PathLength(x, T.right, e + 1)
 9: end if
```

You can create your own test cases by modifying /home/mwang2/bin/nDdata.py and put under /home/<login>/test, and then run with:

```
$ perl Autotest 1 -t /home/<login>/test
```

***Student Name:***
***SSN/ID:***
***Score:***

- Correctness and boundary condition (60%):
- Whitespace and free format compliance (5%):
- Compiling without warning/error (5%):
- Error Handling (5%):
- Modular design, file/directory organizing, showing input, documentation, coding standards (20%):
- Automation (5%):

Subtotal

- Late penalty (20% per day):
- Special service penalty (5%):

***Total score:***