Analysis of the false positive rates of the Nelson rules

July 31, 2015

1 Background

The Nelson rules are a set of rules to determine if a measured variable is out of control.
More information on the wikipedia page

2 Setup

We will generate a set of pseudo-random numbers with a mean of 0 and a standard deviation of 1. We can then see how frequently each rule is triggered from our series.

```
In [7]: from IPython.display import HTML
        from random import normalvariate
        from pandas import DataFrame

        def generate_sequence(length):
            return [normalvariate(0,1) for i in xrange(length)]

        def simulate(rule, n, sequence_length):
            failures = 0
            for iteration in xrange(n):
                if rule(generate_sequence(sequence_length)):
                    failures += 1
            return failures

        def failure_proportion(rule, sequence_length):
            return simulate(rule, 10000, sequence_length) / 10000.0

        def failure_table(rule):

            columns = ["Sequence length", "Failure rate"]
            data = []
            for sequence_length in [10,20,50,100,200,500]:
                failures = failure_proportion(rule, sequence_length) * 100
                data.append([sequence_length, "%2.2f%%" % failures])
            return DataFrame(data, columns=columns)
```

3 Analysis

A simulation of each rule over different sequence lengths will be run 10,000 times. If the rule is triggered at least once in a sequence, that will count as one failure. Rules broken multiple times will not count as multiple failures.

3.1 Rule 1

One point is more than 3 standard deviations from the mean.

```
In [8]: def rule_1(sequence):
            return any(map(lambda value: abs(value) > 3, sequence))

        failure_table(rule_1)
```

Out[8]:

	Sequence length	Failure rate
0	10	2.86%
1	20	5.31%
2	50	12.49%
3	100	22.66%
4	200	41.40%
5	500	74.14%

3.2 Rule 2

Nine (or more) points in a row are on the same side of the mean.

```
In [9]: def rule_2(sequence):
            over_mean = map(lambda value: value > 0, sequence)
            under_mean = map(lambda value: value <= 0, sequence)
            for i in range(len(sequence) - 9):
                if all(over_mean[i:i + 9]) or all(under_mean[i:i + 9]):
                    return True
            return False

        failure_table(rule_2)
```

Out[9]:

	Sequence length	Failure rate
0	10	0.34%
1	20	2.39%
2	50	8.13%
3	100	17.21%
4	200	32.27%
5	500	62.30%

3.3 Rule 3

Six (or more) points in a row are continually increasing (or decreasing).

```
In [10]: def rule_3(sequence):
             pairs = zip(sequence, sequence[1:])
             going_up = map(lambda (previous, current): previous < current, pairs)
             for i in range(len(going_up) - 6):
                 if sum(going_up[i:i+6]) == 6:
                     return True
             return False

         failure_table(rule_3)
```

Out[10]:

	Sequence length	Failure rate
0	10	0.05%
1	20	0.21%

```
2                50        0.77%
3               100        1.57%
4               200        3.43%
5               500        8.11%
```

3.4 Rule 4

Fourteen (or more) points in a row alternate in direction, increasing then decreasing.

```
In [11]: def rule_4(sequence):
             pairs = zip(sequence, sequence[1:])
             going_up = map(lambda (previous, current): previous < current, pairs)
             alternating = map(lambda (previous, current): previous != current, zip(going_up, going_up[
             for i in range(len(alternating) - 12):
                 if sum(alternating[i:i+12]) == 12:
                     return True
             return False

         failure_table(rule_4)
```

```
Out[11]:    Sequence length Failure rate
         0               10        0.00%
         1               20        1.28%
         2               50        6.19%
         3              100       13.26%
         4              200       27.57%
         5              500       56.50%
```

3.5 Rule 5

Two (or three) out of three points in a row are more than 2 standard deviations from the mean in the same direction.

```
In [17]: def rule_5(sequence):
             for i in range(len(sequence) - 3):
                 triplet = sorted(sequence[i:i + 3])
                 if triplet[0] < -2 and triplet[1] < -2:
                     return True
                 if triplet[1] > 2 and triplet[2] > 2:
                     return True
             return False

         failure_table(rule_5)
```

```
Out[17]:    Sequence length Failure rate
         0               10        1.46%
         1               20        3.91%
         2               50        8.71%
         3              100       17.05%
         4              200       32.49%
         5              500       62.64%
```

3.6 Rule 6

Four (or five) out of five points in a row are more than 1 standard deviation from the mean in the same direction.

```
In [13]: def rule_6(sequence):
             for i in range(len(sequence) - 5):
                 quint = sorted(sequence[i:i + 5])
                 if all(map(lambda value: value < -1, quint[:4])):
                     return True
                 if all(map(lambda value: value > 1, quint[1:])):
                     return True
             return False

         failure_table(rule_6)
```

Out[13]:

	Sequence length	Failure rate
0	10	1.83%
1	20	5.38%
2	50	14.80%
3	100	28.19%
4	200	50.50%
5	500	82.25%

3.7 Rule 7

Fifteen points in a row are all within 1 standard deviation of the mean on either side of the mean.

```
In [14]: def rule_7(sequence):
             for i in range(len(sequence) - 15):
                 block_of_15 = sorted(sequence[i:i + 15])
                 if all(map(lambda value: abs(value) < 1, block_of_15)):
                     return True
             return False

         failure_table(rule_7)
```

Out[14]:

	Sequence length	Failure rate
0	10	0.00%
1	20	0.79%
2	50	3.79%
3	100	8.85%
4	200	18.18%
5	500	40.28%

3.8 Rule 8

Eight points in a row exist with none within 1 standard deviation of the mean and the points are in both directions from the mean.

```
In [15]: def rule_8(sequence):
             for i in range(len(sequence) - 8):
                 block_of_8 = sorted(sequence[i:i + 8])
                 if all(map(lambda value: abs(value) > 1, block_of_8)):
                     return True
             return False

         failure_table(rule_8)
```

Out[15]:

	Sequence length	Failure rate
0	10	0.02%

1	20	0.13%
2	50	0.31%
3	100	0.74%
4	200	1.15%
5	500	3.24%

3.9 Combined rules

In [18]:
```
def all_rules(sequence):
    for rule in [rule_1, rule_2, rule_3, rule_4, rule_5, rule_6, rule_7, rule_8]:
        if rule(sequence):
            return True
    return False

failure_table(all_rules)
```

Out[18]:

	Sequence length	Failure rate
0	10	6.16%
1	20	16.53%
2	50	41.74%
3	100	68.94%
4	200	91.58%
5	500	99.74%

In [ ]: