percolator q-values

November 20, 2015

1 Percolator q-values are computed as described in the manuscript

Here we inspect the output of Percolator on an arbitrary file.

The output was obtained in the following way.

1. An X!Tandem output file was converted to tsv using `tandem2pin`:

   ```
   tandem2pin -o test_in.tsv -P DECOY_  output.t.xml
   ```

2. Percolator was run using the following command:

   ```
   percolator -B dec.pin test_in.tsv > targ.pin
   ```

   So, decoy peptides are listed in `dec.pin` and target peptides are listed in `targ.pin`.

   `pin` files are not valid TSV files because the last column (`proteinIds`) contains a variable amount of proteins separated with a tab, so, in order to be able to parse these files, they were preprocessed to remove everything to the right of the first protein ID:

```
cut -f 1-6 dec.pin > dec_cut.pin
cut -f 1-6 targ.pin > targ_cut.pin
```

The scores and q-values calculated by Percolator are intact. This is a recent version of Percolator, so it uses "target-decoy competition" by default to calculate q-values, according to the help message.

Below, we analyze the resulting files.

```
In [1]: # set up the necessary libraries
        import pandas as pd
        %pylab --no-import-all inline
        import seaborn
        from pyteomics import pylab_aux as pa, auxiliary as aux
```

```
Populating the interactive namespace from numpy and matplotlib
```

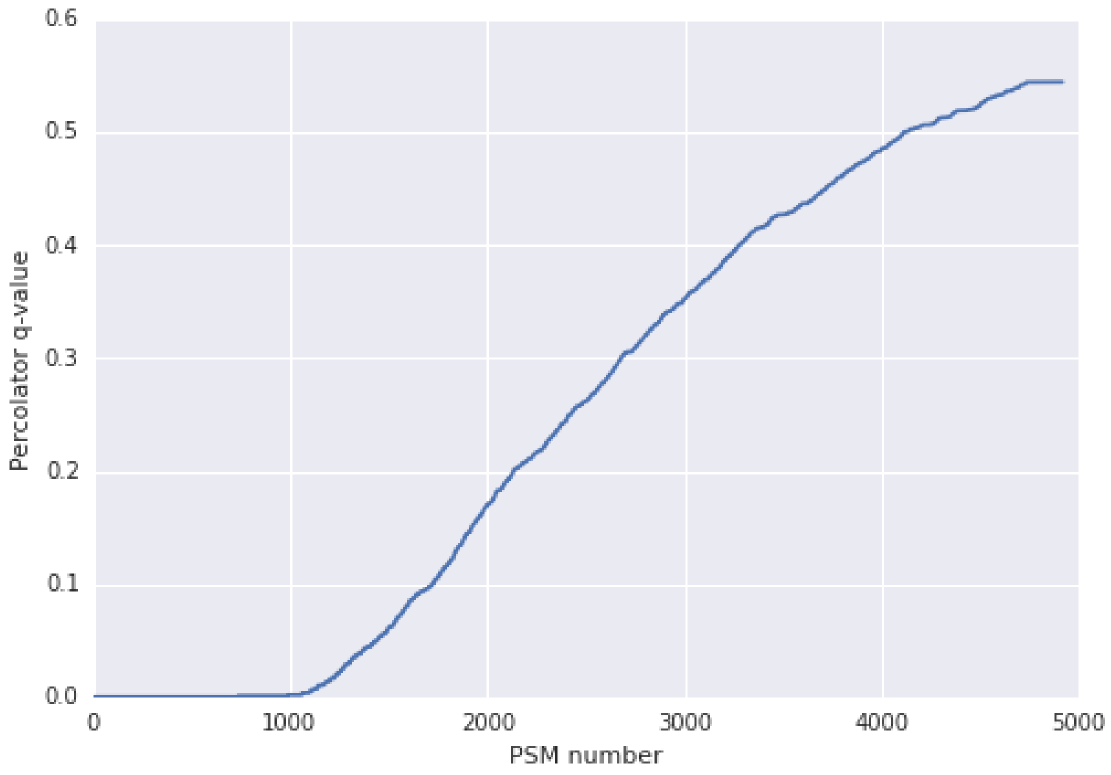
```
In [2]: # read the peptide tables
        targ = pd.read_table('/tmp/targ_cut.pin')
        dec = pd.read_table('/tmp/dec_cut.pin')
```

```
In [3]: # label the peptides as target and decoy, then concatenate tables together
        targ['decoy'] = False
        dec['decoy'] = True
        psms = pd.concat([targ, dec])
        print('There are {} peptides in the table.'.format(psms.shape[0]))
        # take a look at the top rows of the table
        psms.iloc[:5, 1:]
```

```
There are 4916 peptides in the table.

Out[3]:     score  q-value  posterior_error_prob                        peptide  \
        0  6.33811        0          1.598270e-13     K.SCVEEPEPEPEAAEGDGDK.K
        1  5.40133        0          8.943880e-12      R.QAHLCVLASNCDEPMYVK.L
        2  5.38035        0          9.787290e-12      R.DYLDFLDDEEDQGIYQSK.V
        3  5.34931        0          1.118340e-11        K.QLQQAQAAGAEQEVEK.F
        4  5.23310        0          1.842490e-11  K.AAEAAAAPAESAAPAAGEEPSK.E

                                           proteinIds  decoy
        0  sp|P51858|HDGF_HUMAN-Hepatoma-derived-growth-f...  False
        1  sp|P25398|RS12_HUMAN-40S-ribosomal-protein-S12...  False
        2  sp|P25205|MCM3_HUMAN-DNA-replication-licensing...  False
        3  sp|P39748|FEN1_HUMAN-Flap-endonuclease-1-OS=Ho...  False
        4  sp|P80723|BASP1_HUMAN-Brain-acid-soluble-prote...  False
```

```
In [4]: # sort by Percolator score. We use q-value as the secondary search key,
        # so that in case of equal scores
        # peptides are in the same order as Percolator put them
        psms.sort_values(['score', 'q-value'], ascending=[False, True], inplace=True)

        # plot Percolator q-value
        pylab.plot(psms['q-value'])
        pylab.xlabel('PSM number')
        pylab.ylabel('Percolator q-value')
```

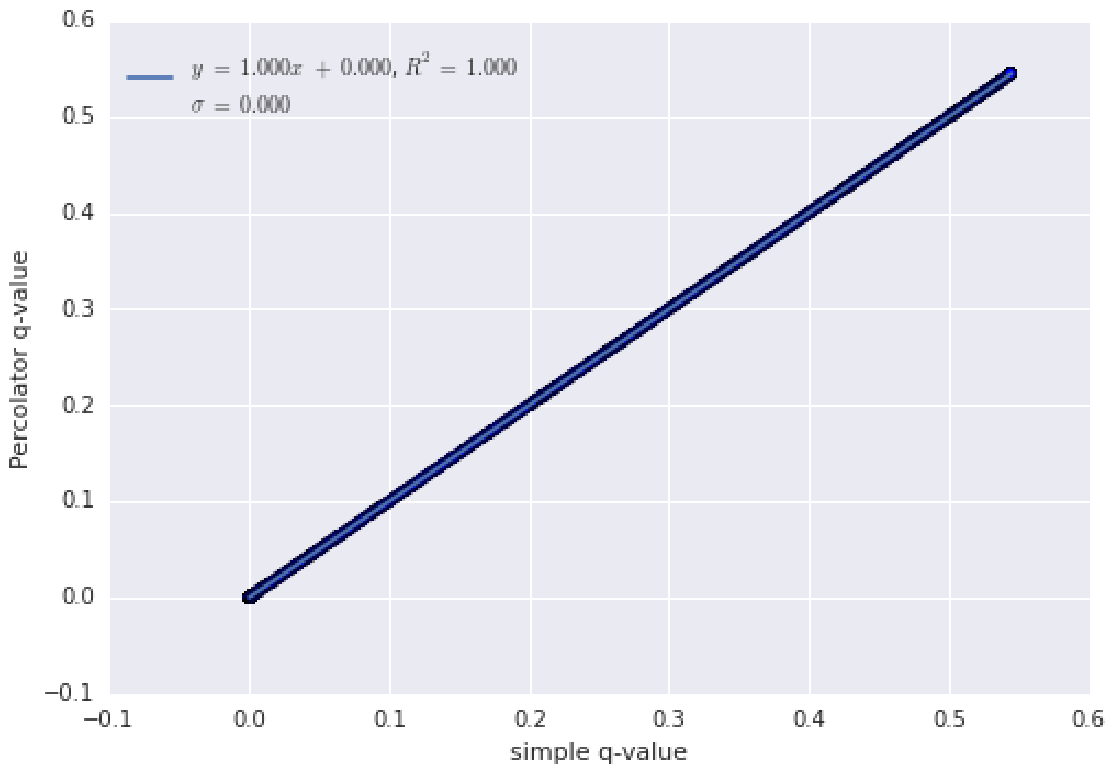
```
Out[4]: <matplotlib.text.Text at 0x7f1c59b61d30>
```



```
In [5]: # Compute "simple" q-values using the procedure described in the manuscript.
        # In short, this means:
        # (1) calculate (# of decoys in top i peptides) / (# of targets in top i PSMs)
        # (2) monotonize using Eq. 6 in revised manuscript
        # This procedure is already implemented in Pyteomics, so we just use it.
        q = aux.qvalues(psms, key='q-value', is_decoy='decoy', formula=1)
        psms['q'] = q['q'] # psms['q'] now has the "simple" q-values
```

```
In [6]: # compare two versions of q-value by plotting them against one another; they look the same
        pa.scatter_trend(psms['q'], psms['q-value'])
        pylab.xlabel('simple q-value')
        pylab.ylabel('Percolator q-value')
```

```
Out[6]: <matplotlib.text.Text at 0x7f1c59a89128>
```



```
In [7]: # confirm that all q-values are indeed the same within floating point accuracy
        np.allclose(psms['q'], psms['q-value'])
```

```
Out[7]: True
```

We can also do the same analysis on Percolator results obtained using `qvality` for q-value calculation. In this case, we surely do not expect these q-values to match "simple" q-values.

We ran the following:

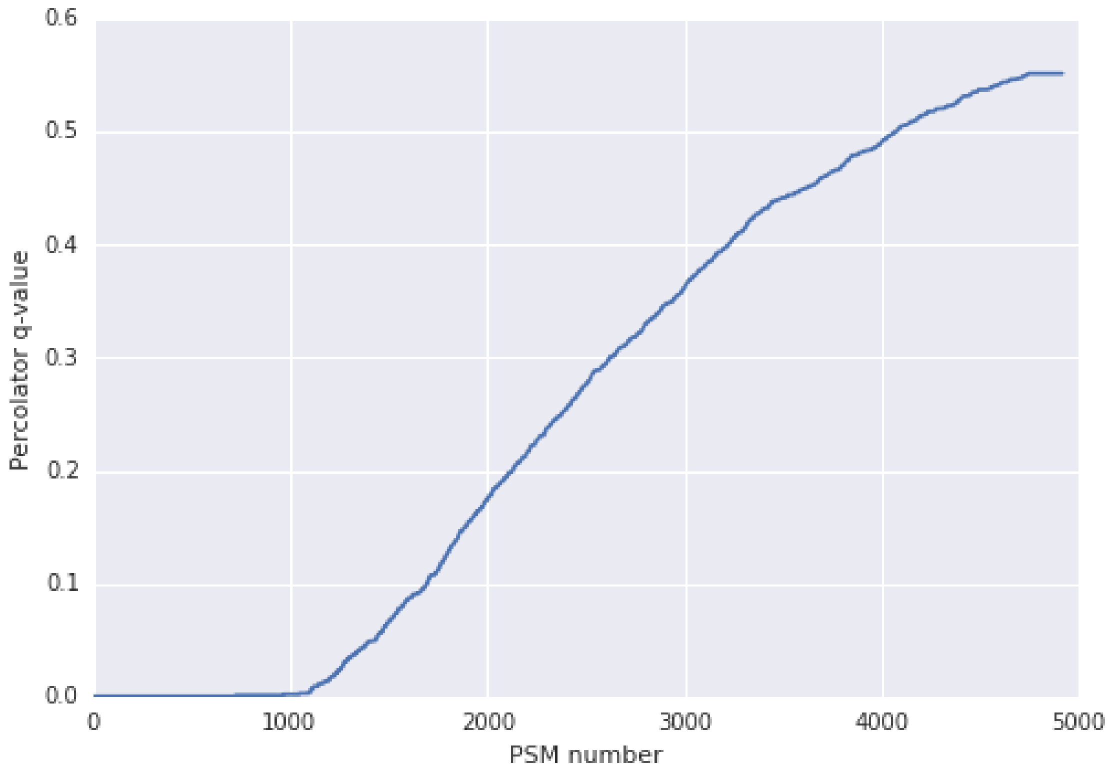
```
percolator -y -B dec_y.pin test_in.tsv > targ_y.pin
```

Then performed the `cut` operation on the `pin` files to get `targ_y_cut.pin` and `dec_y_cut.pin`. The rest of the code is the same as above.

```
In [8]: targ = pd.read_table('/tmp/targ_y_cut.pin')
        dec = pd.read_table('/tmp/dec_y_cut.pin')
        targ['decoy'] = False
        dec['decoy'] = True
        psms = pd.concat([targ, dec])
        print('There are {} peptides in the table.'.format(psms.shape[0]))
        psms.sort_values(['score', 'q-value'], ascending=[False, True], inplace=True)
        pylab.plot(psms['q-value'])
        pylab.xlabel('PSM number')
        pylab.ylabel('Percolator q-value')
```

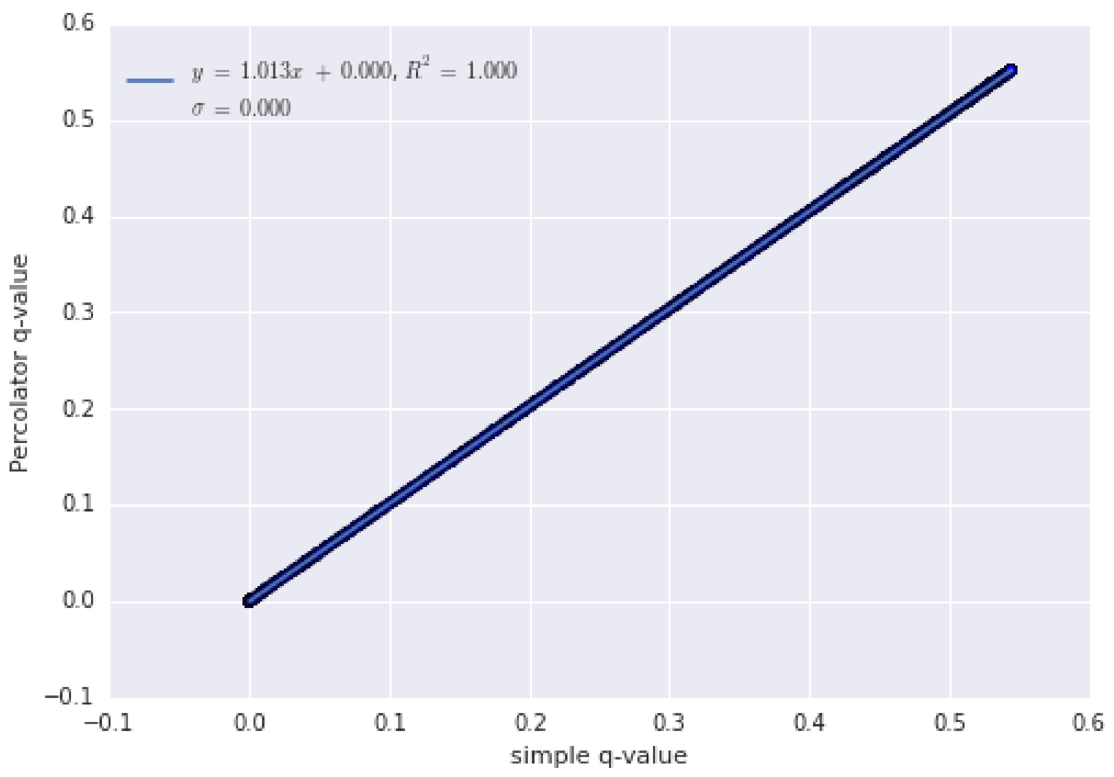
```
There are 4916 peptides in the table.
```

```
Out[8]: <matplotlib.text.Text at 0x7f1c5997ac50>
```



```
In [9]: q = aux.qvalues(psms, key='q-value', is_decoy='decoy', formula=1)
        psms['q'] = q['q']
        pa.scatter_trend(psms['q'], psms['q-value'])
        pylab.xlabel('simple q-value')
        pylab.ylabel('Percolator q-value')
        np.allclose(psms['q'], psms['q-value'])
```

```
Out[9]: False
```



Thus, monotonization done by Percolator *in default mode* matches Eq. 6 in the revised manuscript and the definition by Kall et al.

When using `qvality`, q-values are calculated differently, and the results presented in the manuscript do not apply. It's not clear why they are directly proportional to the "simple" q-values, but that is beyond the topic.