method concensus

February 28, 2017

```
In [5]: # import and set options
        %matplotlib inline
        import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import scipy.stats as stats
        import os
        from collections import Counter
        import utils
```

0.1 Method Concensus

Cancer driver genes through the pattern of somatic mutations may show several signals of positive selection. It therefore is likely that several computational methods may find a gene as significant. Further, there is a correlation between the number of methods which agree a gene is a significant cancer driver gene and the likelihood a gene is found in the cancer gene census (Tamborero *et al.*). With this observation, the fraction of predicted driver genes by each method which agrees with additional methods serves as a proxy for positive predictive value. This metric also tends to reduce the underestimation of positive predictive value, which occurs when utilizing the cancer gene census.

0.1.1 Load data

We first load in the significant genes for each method (in our case q<=.1).

```
In [ ]: # load config file
        config = utils.load_config('config.yaml')

        # make output directory
        if not os.path.exists('output'): os.mkdir('output')

        # get the significant genes for each method
        signif_dict = utils.fetch_significant_genes('example_data/pancan',  # data
                                                    .1,  # q-value threshold
                                                    config)
        num_methods = len(signif_dict)
```

```
        # eliminate any excluded methods.
        # in this case, this is OncodriveFM as
        # the latest version OncodriveFML is already included
        if utils.is_valid_config(config, 'exclude', 'method_overlap'):
            exclude_methods = config['exclude']['method_overlap']
            meth_list = list(signif_dict.keys())
            for meth in meth_list:
                if meth in exclude_methods:
                    del signif_dict[meth]
```

0.1.2 Method overlap functions

The following functions count the overlap between methods on predicted driver genes.

```
In [29]: def gene_overlap_count(cts, num_methods):
             """Count the number of methods that agree on each significant gene."""
             # now count the number of overlaps
             ovlp_genes = {}
             for nmeth in range(1, num_methods+1):
                 mygenes = [x for x in cts if cts[x]==nmeth]
                 ovlp_genes[nmeth] = mygenes

             # format the results to a dataframe
             max_genes = max(map(len, ovlp_genes.values()))
             for i in range(1, num_methods+1):
                 list_len = len(ovlp_genes[i])
                 ovlp_genes[i] = ovlp_genes[i] + [None]*(max_genes-list_len)
             overlap_df = pd.DataFrame(ovlp_genes)

             return overlap_df


         def method_overlap_count(signif_genes, gene_counts):
             """Calculate the ammount of overlap with other methods."""
             output_list = []
             for method in signif_genes:
                 # list of num ovlps
                 method_cts = [gene_counts[g]-1 for g in signif_genes[method]]

                 # get overlap counts
                 num_uniq = len([x for x in method_cts if x==0])
                 num_one = len([x for x in method_cts if x==1])
                 num_two = len([x for x in method_cts if x==2])
                 num_three = len([x for x in method_cts if x>=3])
                 num_total = len(method_cts)

                 # append result
                 tmp_list = [method, num_total, num_uniq, num_one, num_two, num_thr
```

```
            output_list.append(tmp_list)

        # format into dataframe
        header_names = ['Method', 'Total', 'predicted by 1 method', 'two metho
                        'three methods', 'at least four methods']
        output_df = pd.DataFrame(output_list, columns=header_names)

        return output_df
```

0.1.3 Calculate overlap

The next step is to calculate the overlap of significant genes for each method.

```
In [33]: # count how many time each gene is significant
         gene_cts = Counter([g for method in signif_dict for g in signif_dict[metho
         gene_overlap_df = gene_overlap_count(gene_cts, num_methods)
         gene_overlap_df.to_csv('output/gene_overlap_counts.txt', sep='\t', index=F

         # calculate the number of overlaps for each method
         method_ovlp_df = method_overlap_count(signif_dict, gene_cts)
         method_ovlp_df
```

Out[33]:

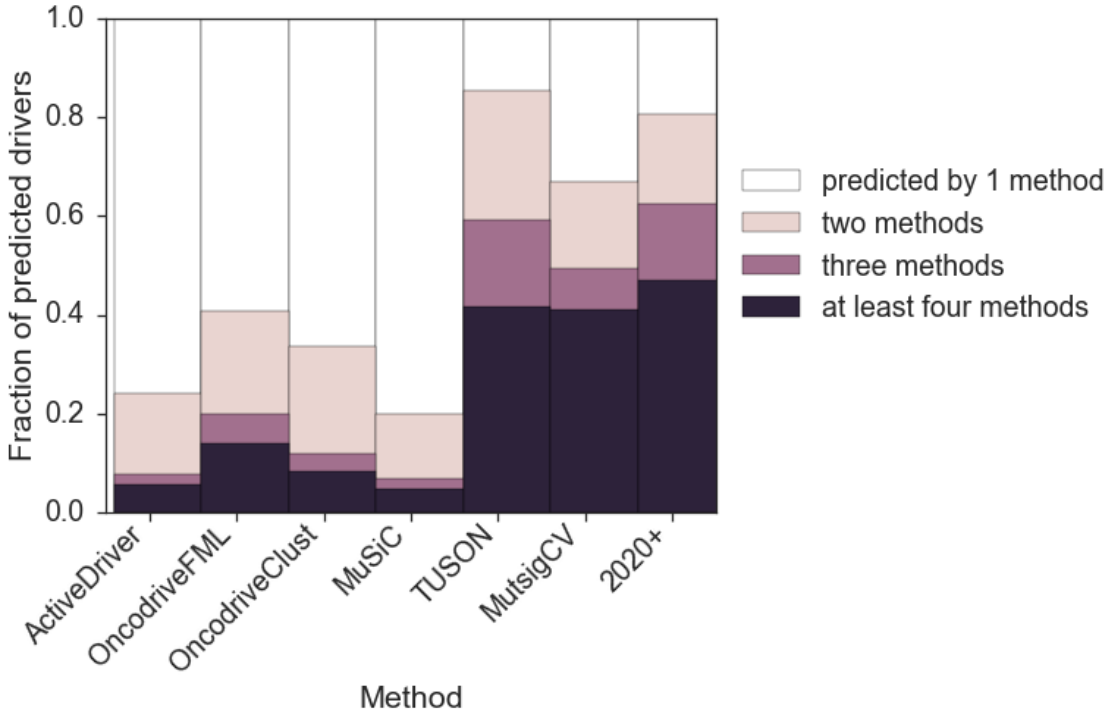
	Method	Total	predicted by 1 method	two methods	three metho
0	ActiveDriver	417	316	69	
1	OncodriveClust	586	389	127	2
2	TUSON	243	35	64	4
3	OncodriveFML	679	402	141	4
4	MuSiC	1975	1582	257	4
5	MutsigCV	158	52	28	1
6	2020+	208	40	38	3

	at least four methods
0	23
1	49
2	101
3	94
4	95
5	65
6	98

0.1.4 Plot results

```
In [41]: def plot_method_overlap(overlap_df, custom_order=None):
             """Plot the fraction overlap of predicted driver genes with other meth
             """
             # calculate the fractions
             overlap_df['predicted by 1 method'] = 1.0
             mycols = ['two methods', 'three methods', 'at least four methods']
             overlap_df.loc[:, mycols] = overlap_df.loc[:, mycols].astype(float).di
```

```
    overlap_df['three methods'] = overlap_df['three methods'] + overlap_df
    overlap_df['two methods'] = overlap_df['two methods'] + overlap_df['th

    # order methods in increasing order
    if custom_order is None:
        custom_order = overlap_df.sort_values('two methods')['Method'].tol

    colors = ['white'] + sns.cubehelix_palette(3)[:3]
    for i, col in enumerate(['predicted by 1 method',
                             'two methods', 'three methods', 'at least fou
        with sns.axes_style('ticks', rc={'xtick.major.pad':-1.0}), sns.plo
            sns.barplot('Method', col, data=overlap_df,
                        color=colors[i], label=col, order=custom_order,)

            # Finishing touches
            lgd = plt.legend(bbox_to_anchor=(1, .75), loc='upper left',
                             ncol=1,)
            plt.ylim((0, 1))
            plt.ylabel('Fraction of predicted drivers')
            plt.gca().set_xticklabels(custom_order, rotation=45, ha='right
            fig = plt.gcf()
            fig.set_size_inches(7, 7)

            # set bar width to 1
            for container in plt.gca().containers:
                plt.setp(container, width=1)
            # remove extra ticks
            plt.gca().get_xaxis().tick_bottom()
            plt.gca().get_yaxis().tick_left()

            # change tick padding
            plt.gca().tick_params(axis='x', which='major', pad=0)

    plt.tight_layout()

order = ['ActiveDriver', 'OncodriveFML', 'OncodriveClust',
         'MuSiC', 'TUSON', 'MutsigCV', '2020+']
plot_method_overlap(method_ovlp_df.copy(), custom_order=order)
```


1.0
0.8
0.6
0.4
0.2
0.0
Fraction of predicted drivers
predicted by 1 method
two methods
three methods
at least four methods
ActiveDriver
OncodriveFML
OncodriveClust
MuSiC
TUSON
MutsigCV
2020+
Method