CoMap evaluation

October 2, 2015

In [1]:
```
%matplotlib inline
import os
from collections import OrderedDict
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import yaml
from pathlib import Path
from data import CodesInDbs, Mappings, Databases
from IPython.display import Latex
import normalize

pd.set_option('display.max_colwidth', 100)

sns.set_style('whitegrid')
#sns.set_context("poster")
#plt.rcParams['figure.figsize'] = (4, 3)
plt.rc("savefig", dpi=150)

measures_palette = sns.color_palette('Set1', n_colors=2, desat=.5)
measures_palette.reverse()

def graded_recall_palette(n_colors):
    palette = sns.color_palette("Blues", n_colors=n_colors, desat=.6)
    palette.reverse()
    return palette

def graded_precision_palette(n_colors):
    palette = sns.color_palette("Reds", n_colors=n_colors, desat=.6)
    palette.reverse()
    return palette

def mystyle(palette=None, xrot=0, ha='center', ylim=(0,1), ylabel=None, savefig=None):
    class C:
        def __enter__(self):
            if palette is not None:
                palette.__enter__()
        def __exit__(self, exc_type, value, traceback):
            if palette is not None:
                palette.__exit__(exc_type, value, traceback)
            if exc_type is None:
                sns.despine(left=True)
```

```
                    plt.grid(False, axis='x')
                    if plt.gca().legend_:
                        plt.legend(loc=2, bbox_to_anchor=(1, 1))
                    plt.gca().get_lines()[0].set_visible(False)
                    plt.gca().set_ylim(*ylim)
                    plt.xticks(rotation=xrot, ha=ha)
                    if ylabel is not None:
                        plt.ylabel(ylabel)
                    if savefig:
                        plt.savefig('{}-{}'.format(PROJECT, savefig))
            return C()

        pd.set_option('display.notebook_repr_html', True)
        def _repr_latex_(self):
            #return r"\begin{center}%s\end{center}" %
            return self.to_latex()
        pd.DataFrame._repr_latex_ = _repr_latex_  # monkey patch pandas DataFrame

        PROJECT = os.getenv('COMAP_PROJECT')
        print("PROJECT:", PROJECT)
```

```
PROJECT: safeguard
```

```
In [2]: with open('../projects/{}/variations.yaml'.format(PROJECT)) as f:
            variations = yaml.load(f)

        with open('../projects/{}/config.yaml'.format(PROJECT)) as f:
            config = yaml.load(f)
            databases = Databases.of_config(config)
            coding_systems = config['coding-systems']

        with open('../projects/{}/events.yaml'.format(PROJECT)) as f:
            events = yaml.load(f)
            event_names = {}
            for event in events:
                casedef = yaml.load(open('../projects/{}/case-definitions/{}.yaml'.format(PROJECT, even
                event_names[event] = casedef['name']

        with open('../projects/{}/mappings.yaml'.format(PROJECT)) as f:
            mappings = Mappings.of_raw_data(yaml.load(f), events, databases)
            mappings = normalize.mappings(mappings, databases)

        with open('../codes-in-dbs.json') as f:
            codes_in_dbs = CodesInDbs.of_data(json.load(f))

        with open('../{}.code-stats.csv'.format(PROJECT)) as f:
            code_stats = pd.read_csv(f)

        def database_label(database):
            return database
            #return "{} ({})".format(database, databases.coding_system(database))

        def measure_label(measure):
            return {
                "recall": "Sensitivity",
```

```
            "precision": "PPV", # "Positive predictive value",
        }[measure]

    def event_label(event):
        return event_names[event]
```

In [3]:
```
df0 = pd.read_csv('../{}.evaluations.csv'.format(PROJECT))
df = df0['variation event database recall precision'.split()]
df.head()
```

Out[3]:

	variation	event	database	recall	precision
0	max-recall	is	IPCI	1.000000	1.000000
1	max-recall	is	GePaRD	0.909091	0.666667
2	max-recall	is	CPRD	0.850000	0.850000
3	max-recall	is	Medicare	1.000000	0.833333
4	max-recall	ap	IPCI	1.000000	1.000000

1 Notes

Should exclusion codes from the reference be generated? No. Exclusion codes are often added database specifically, where the codes are *not* represented in the case definition.

2 Coding systems

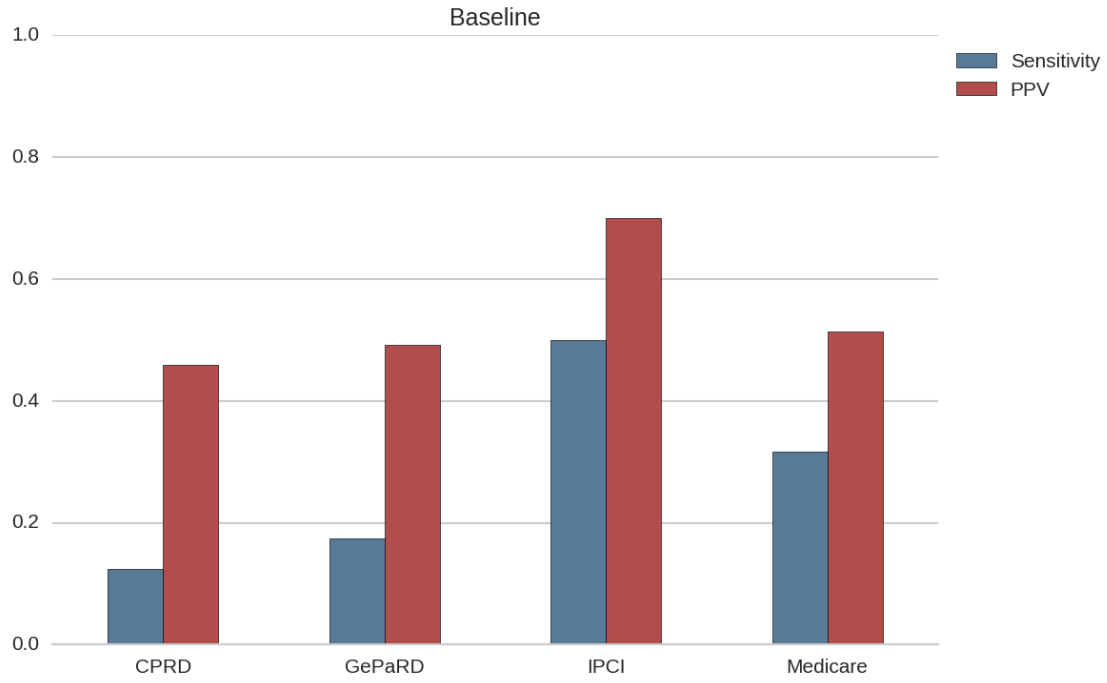
In [4]:
```
pd.DataFrame([
        (database, databases.coding_system(database))
        for database in databases.databases()
    ], columns=("Database", "Coding system")).set_index("Database")
```

Out[4]:

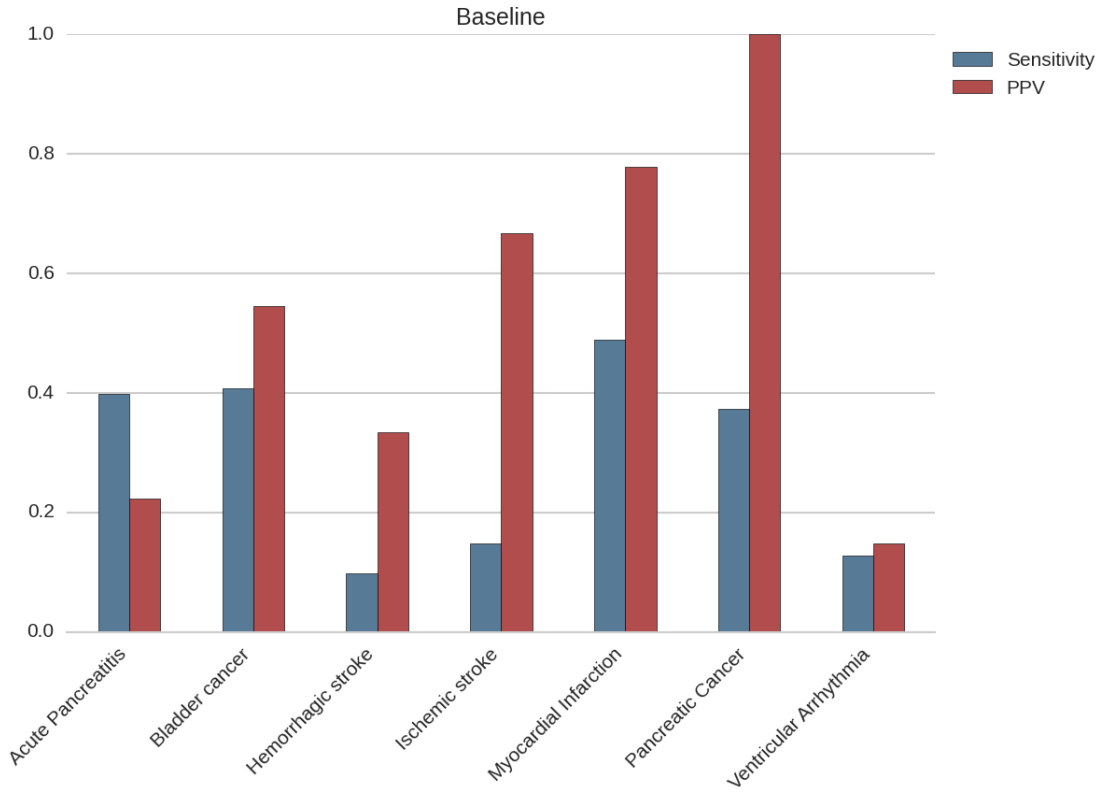
Database	Coding system
Medicare	ICD9CM
IPCI	ICPC2EENG
CPRD	RCD2
GePaRD	ICD10CM

3 Baseline

In [5]:
```
averages_compare = pd.DataFrame([
        df[df.variation == 'baseline'].groupby('database').recall.mean(),
        df[df.variation == 'baseline'].groupby('database').precision.mean(),
    ])
averages_compare.index = averages_compare.index.map(measure_label)
averages_compare.columns = averages_compare.columns.map(database_label)
with mystyle(measures_palette, savefig='baseline-performance-by-db.pdf'):
    averages_compare.T.plot(kind='bar', title='Baseline')
```




```
In [6]: averages_compare = pd.DataFrame([
            df[df.variation == 'baseline'].groupby('event').recall.mean(),
            df[df.variation == 'baseline'].groupby('event').precision.mean(),
        ])
        averages_compare.index = averages_compare.index.map(measure_label)
        averages_compare.columns = averages_compare.columns.map(event_label)
        with mystyle(measures_palette, xrot=45, ha='right', savefig='baseline-performance-by-event.pdf')
            averages_compare.T.plot(kind='bar', title='Baseline')
```

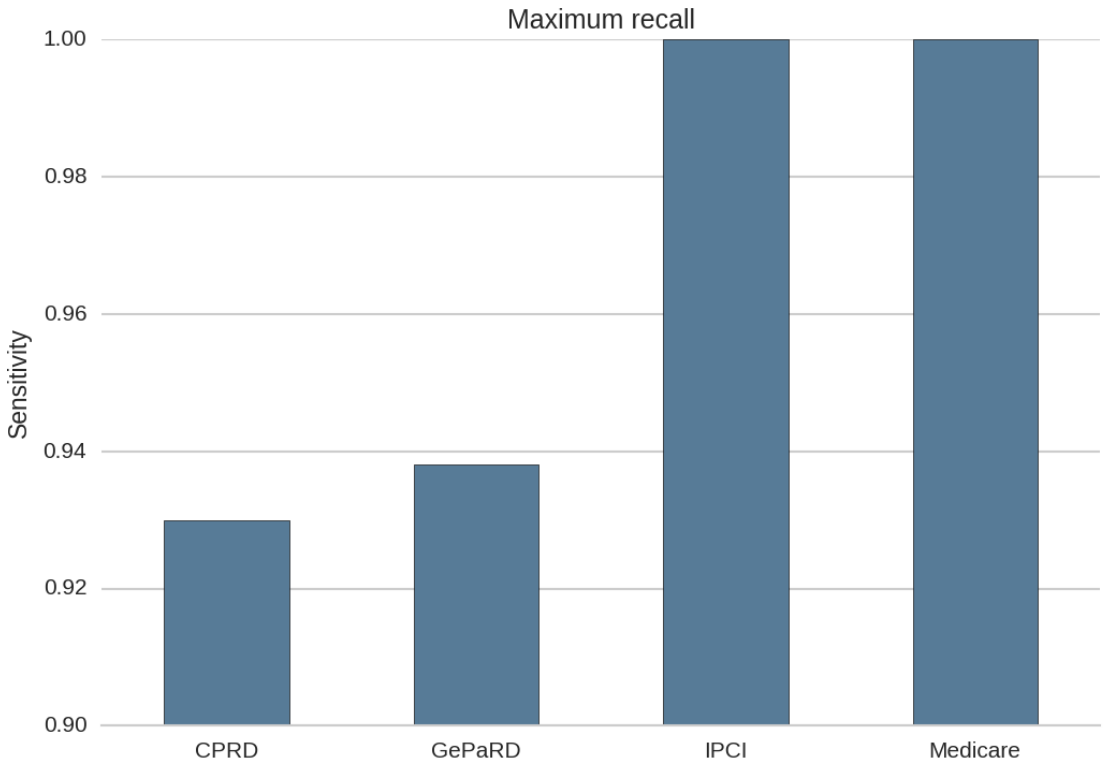


4 Max-recall

```
In [7]: averages_compare = pd.DataFrame([
            df[df.variation == 'max-recall'].groupby('database').recall.mean(),
        ])
        averages_compare.index = [measure_label('recall')]
        averages_compare.columns = averages_compare.columns.map(database_label)
        with mystyle(measures_palette, ylim=(.9,1), savefig='max-recall-by-db.pdf'):
            averages_compare.T.plot(kind='bar', legend=False, title='Maximum recall')
            plt.ylabel(measure_label('recall'))
        averages_compare
```

Out[7]:

	CPRD	GePaRD	IPCI	Medicare
Sensitivity	0.929871	0.938033	1	1



4.1 Reasons for imperfect sensitivity

```
In [8]: stats = DataFrame()
        stats['In mapping'] = code_stats[code_stats.InMapping]\
            .groupby('Database').Code.count()
        stats['Not in maximum recall'] = code_stats[code_stats.InMapping & ~code_stats.InDnf]\
            .groupby('Database').Code.count()
        stats.fillna(0, inplace=True)
        stats['%'] = (stats['Not in maximum recall'] / stats['In mapping']).map("{:.2%}".format)
        #stats['Not in maximum recall but in database'] = code_stats[code_stats.InMapping & code_stats.
        #    .groupby('Database').Code.count()
        stats.fillna(0, inplace=True)
        stats
```

Out[8]:

	In mapping	Not in maximum recall	%
Database			
CPRD	229	14	6.11%
GePaRD	72	3	4.17%
IPCI	7	0	0.00%
Medicare	50	0	0.00%

```
In [9]: max_recall_fn = df0[(df0.variation == 'max-recall') & (df0.recall < 1)][["database", "fn"]]
        max_recall_fn.database = max_recall_fn.database.map(database_label)
        max_recall_fn.fn = max_recall_fn.fn.fillna('[]').map(json.loads)
```

```
max_recall_fn = max_recall_fn.groupby('database').fn.sum().to_frame()
max_recall_fn.fn = max_recall_fn.fn.map(set).map(', '.join)
max_recall_fn.index.name = 'Database'
max_recall_fn.columns = ['False negatives of maximum recall']
max_recall_fn
```

Out[9]:

Database	False negatives of maximum recall
CPRD	G60X., BBd9., G61X., G6W.., 7L1H7, G6X.., ByuE0, G64z4, BBaz., ByuE., BBa.., 1O0.., 7L1H6, BA0z.
GePaRD	I46.0, I64, I21.9

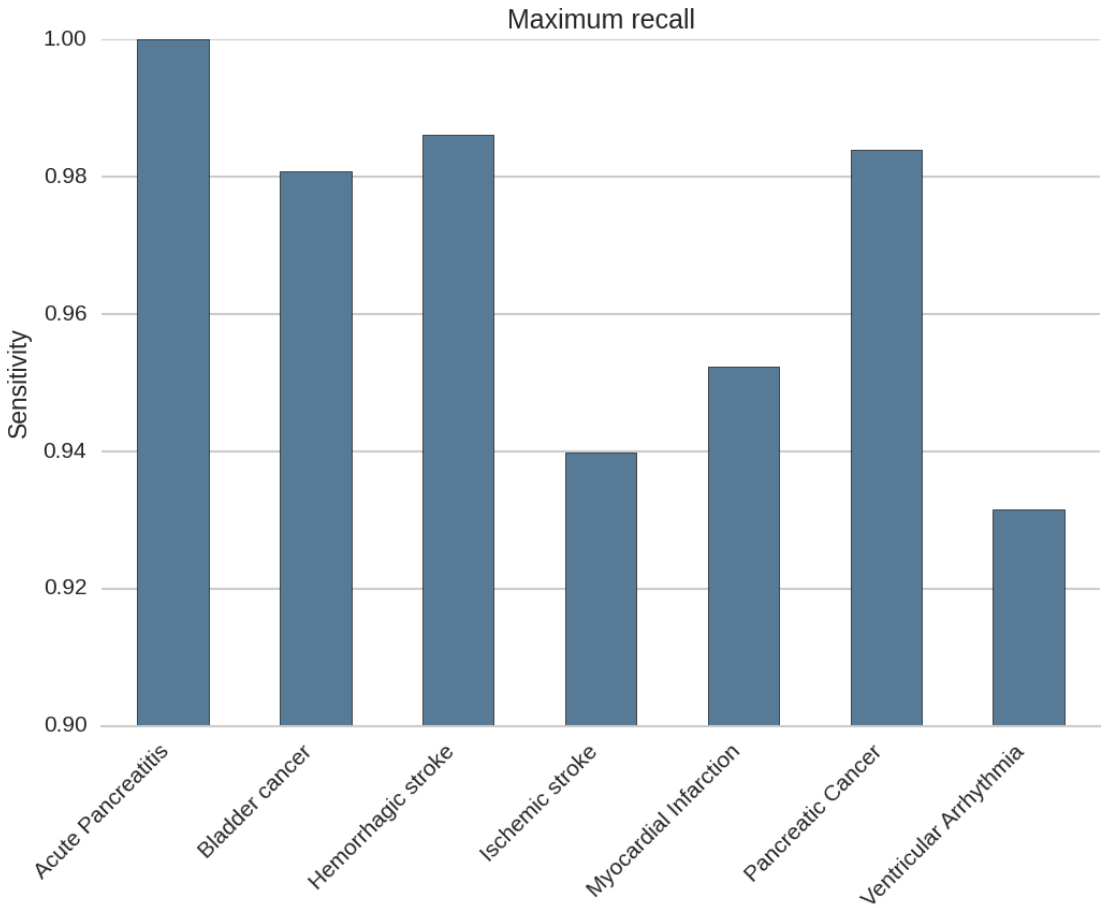
`CPRD`: READ2 codes from the reference are mapped to READ CTV3 codes that are not in UMLS, for example `7L1H6` (READ2) -> `XaM3E`, `XaPuP`, `7L1H6`, `7L1h6`.

`GePaRD`: Only 3 codes are missing, but those FN have large influence on sensitivity in mappings with few codes.

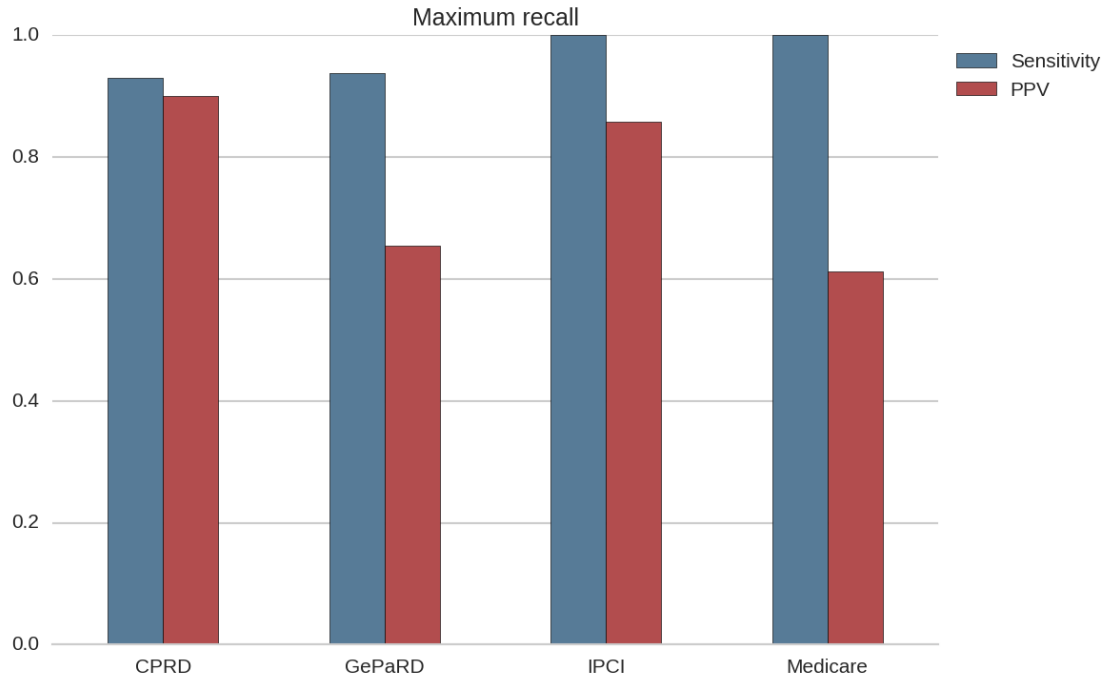
In [10]:
```
averages = df[df.variation == 'max-recall'].groupby('event').recall.mean()
averages.name = measure_label('recall')
averages.index = averages.index.map(event_names.get)
with mystyle(measures_palette, xrot=45, ha='right', ylim=(0.9,1), savefig='max-recall-by-event
    averages.plot(kind='bar', legend=False, title="Maximum recall")
    plt.ylabel(measure_label('recall'))
averages.to_frame()
```

Out[10]:

	Sensitivity
Acute Pancreatitis	1.000000
Bladder cancer	0.980769
Hemorrhagic stroke	0.986111
Ischemic stroke	0.939773
Myocardial Infarction	0.952381
Pancreatic Cancer	0.983945
Ventricular Arrhythmia	0.931481

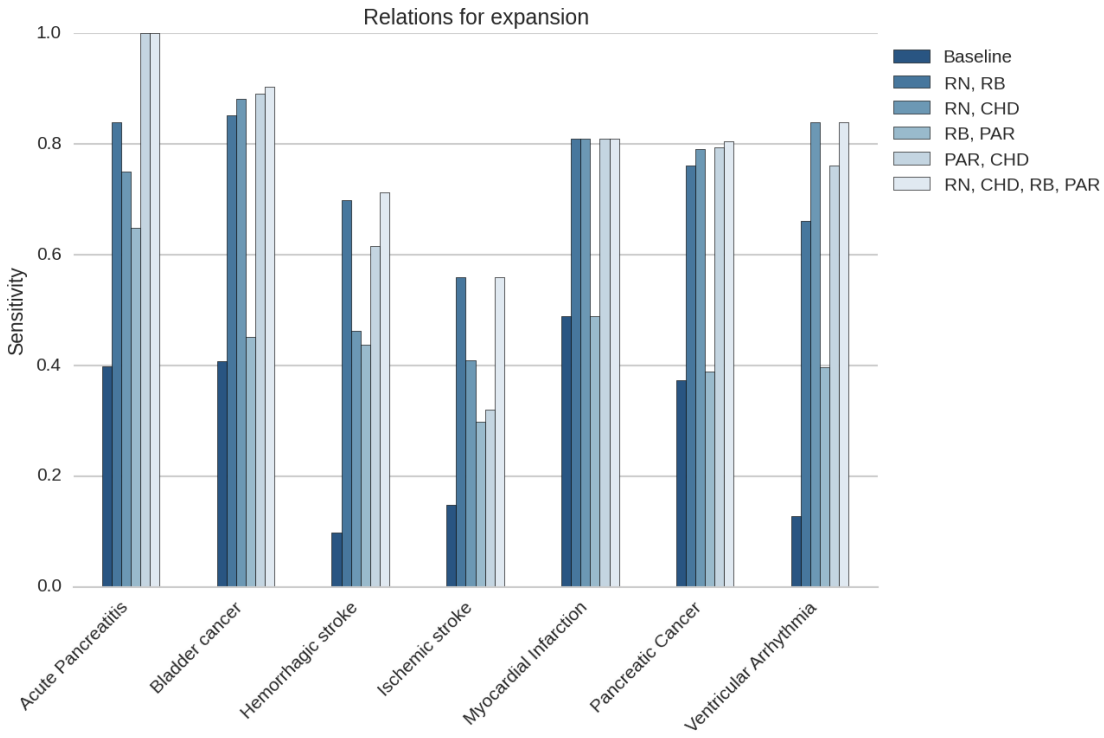



```
In [11]: averages_compare = pd.DataFrame([
             df[df.variation == 'max-recall'].groupby('database').recall.mean(),
             df[df.variation == 'max-recall'].groupby('database').precision.mean(),
         ])
         averages_compare.index = averages_compare.index.map(measure_label)
         averages_compare.columns = averages_compare.columns.map(database_label)
         with mystyle(measures_palette, savefig='max-recall-performance-by-db.pdf'):
             averages_compare.T.plot(kind='bar', title='Maximum recall')
```

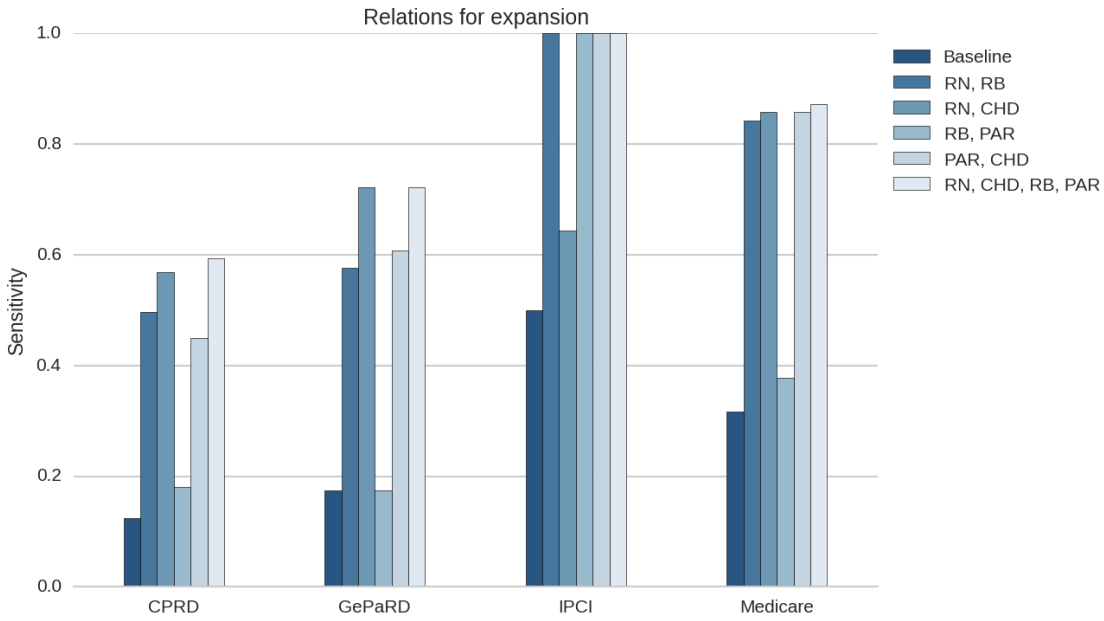


5 Compare relations for expansion

```
In [12]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-RB.expand', 'RN, RB'),
             ('1-RN-CHD.expand', 'RN, CHD'),
             ('1-RB-PAR.expand', 'RB, PAR'),
             ('1-PAR-CHD.expand', 'PAR, CHD'),
             ('1-RN-CHD-RB-PAR.expand', 'RN, CHD, RB, PAR'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').recall.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(event_names.get)
         averages_compare.index = compare_variations.values()

         with mystyle(graded_recall_palette(len(compare_variations)), xrot=45, ha='right', savefig='rela
             averages_compare.T.plot(kind='bar', title="Relations for expansion")
             plt.ylabel(measure_label('recall'))
```

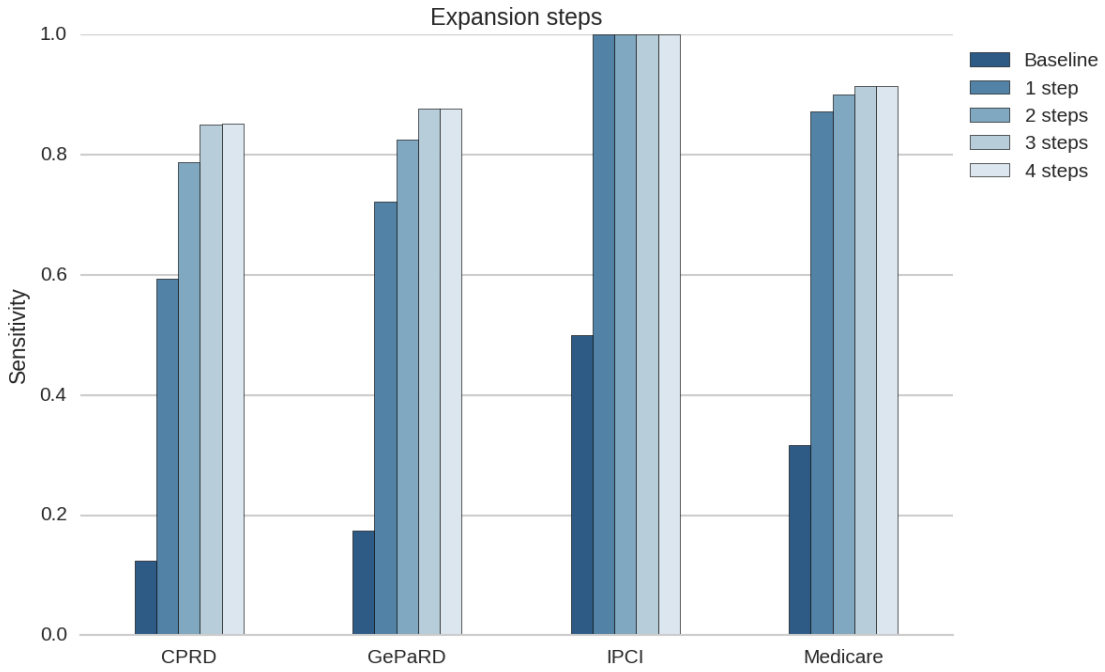



```
In [13]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-RB.expand', 'RN, RB'),
             ('1-RN-CHD.expand', 'RN, CHD'),
             ('1-RB-PAR.expand', 'RB, PAR'),
             ('1-PAR-CHD.expand', 'PAR, CHD'),
             ('1-RN-CHD-RB-PAR.expand', 'RN, CHD, RB, PAR'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('database').recall.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(database_label)
         averages_compare.index = compare_variations.values()
         with mystyle(graded_recall_palette(len(compare_variations)), savefig='relations-recall-by-db.p
             averages_compare.T.plot(kind='bar', title="Relations for expansion")
             plt.ylabel(measure_label('recall'))
```



6 Increasing sensitivity with more expansion steps

```
In [14]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-CHD-RB-PAR.expand', '1 step'),
             ('2-RN-CHD-RB-PAR.expand', '2 steps'),
             ('3-RN-CHD-RB-PAR.expand', '3 steps'),
             ('4-RN-CHD-RB-PAR.expand', '4 steps'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('database').recall.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(database_label)
         averages_compare.index = compare_variations.values()

         with mystyle(graded_recall_palette(len(compare_variations)), savefig='steps-recall-by-db.pdf'):
             averages_compare.T.plot(kind='bar', title="Expansion steps")
             plt.ylabel(measure_label('recall'))
```



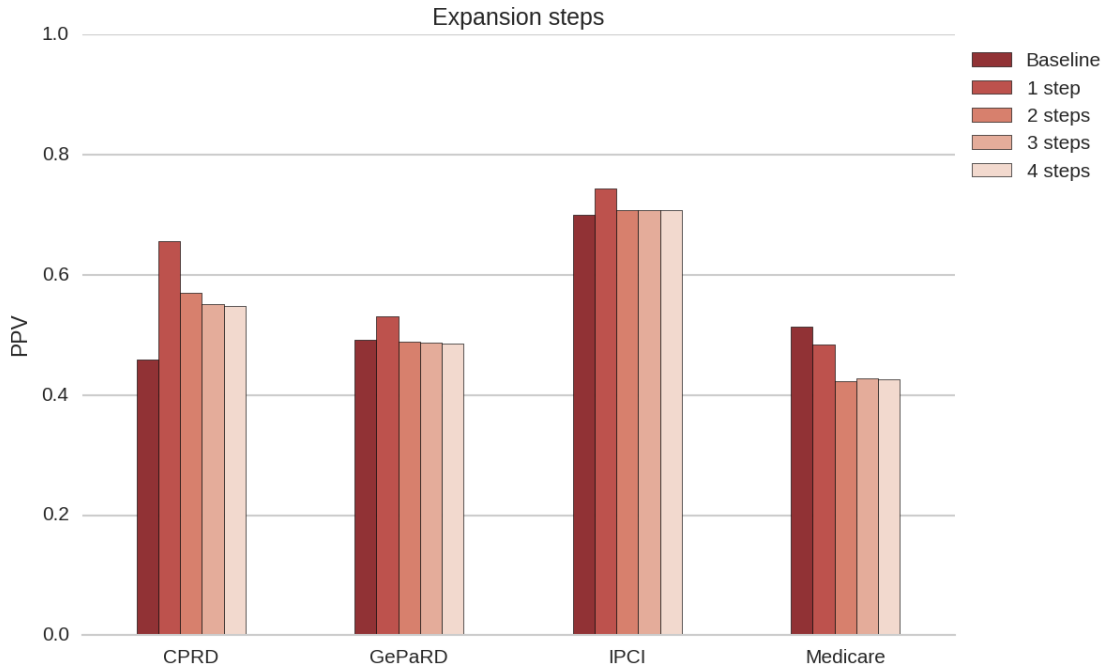
6.1 Reasons for low performance in IPCI when including exclusion codes

Exclusion codes are not in the evaluation any more. See note above.

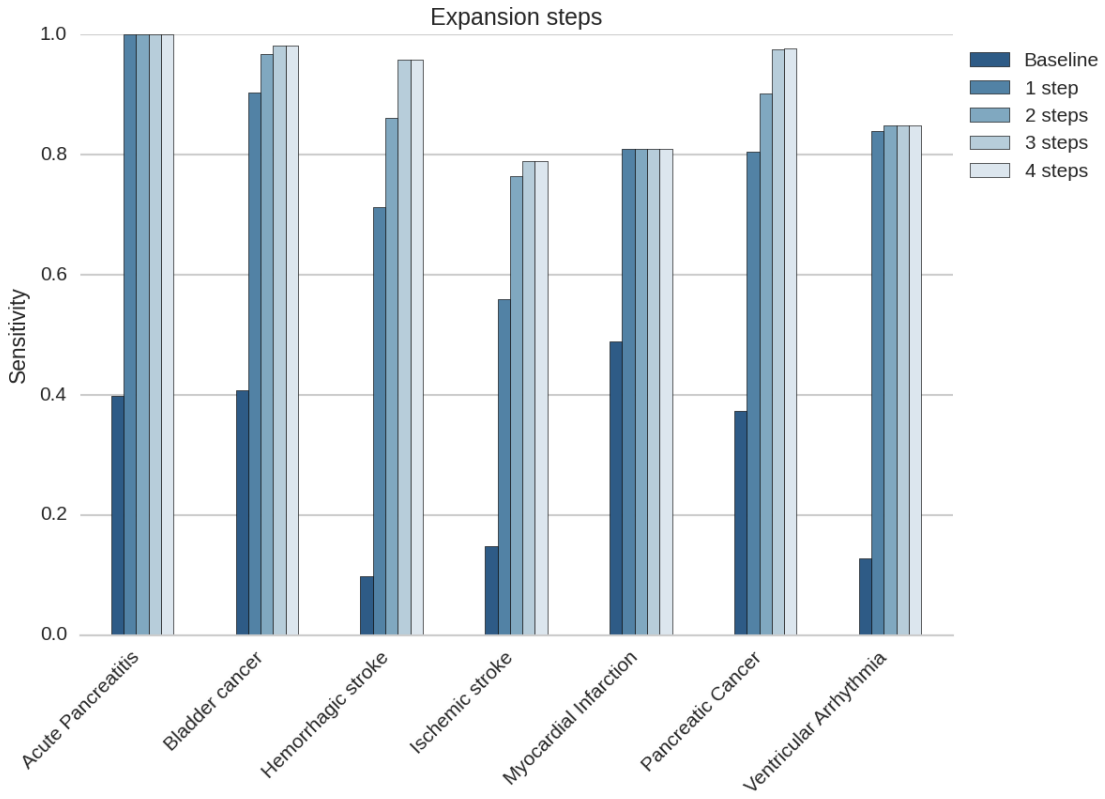
The IPCI mapping contains *very* broad codes that are refined with additional terms. For example

- `K24` (Fear of heart attack)
- `K90` (stroke)
- `K93` (Pulmonary embolism)
- `D70` (Dementia) OR "dementia" AND "infarct"
- `U14` (Kidney symptom/complaint) OR "nier" AND "infarct"

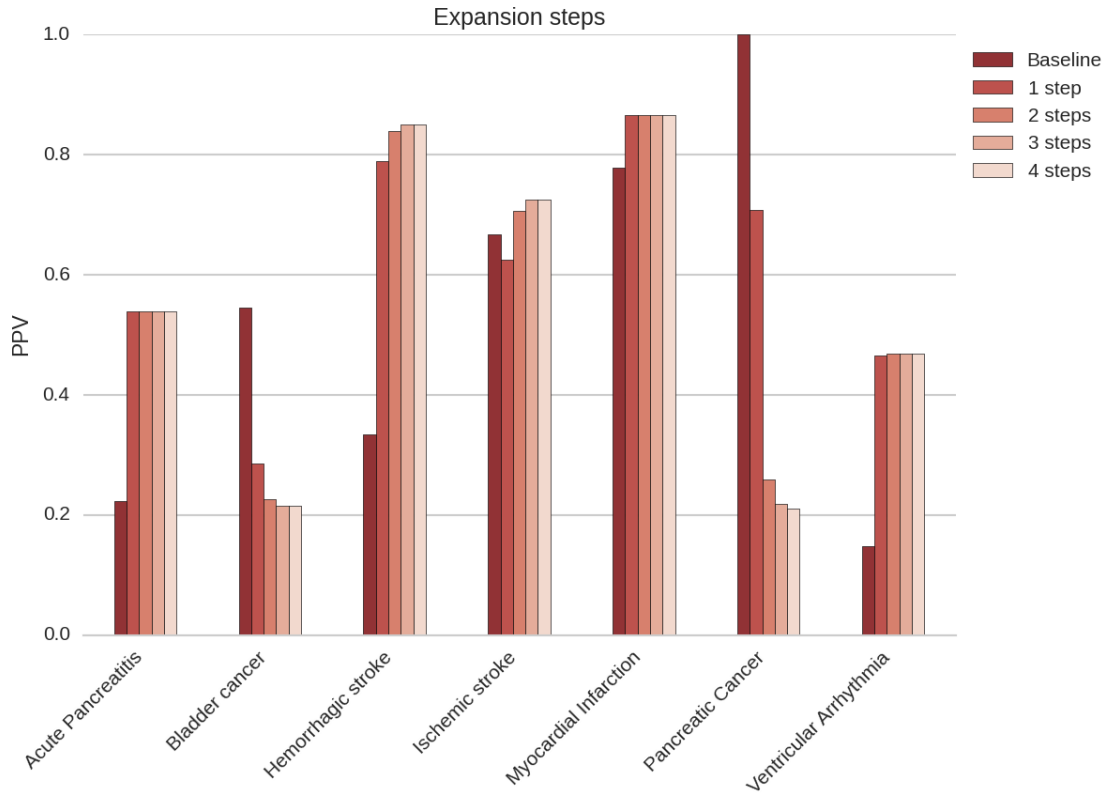
```
In [15]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-CHD-RB-PAR.expand', '1 step'),
             ('2-RN-CHD-RB-PAR.expand', '2 steps'),
             ('3-RN-CHD-RB-PAR.expand', '3 steps'),
             ('4-RN-CHD-RB-PAR.expand', '4 steps'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('database').precision.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(database_label)
         averages_compare.index = compare_variations.values()

         with mystyle(graded_precision_palette(len(compare_variations)), savefig='steps-precision-by-db
             averages_compare.T.plot(kind='bar', title="Expansion steps")
             plt.ylabel(measure_label('precision'))
```



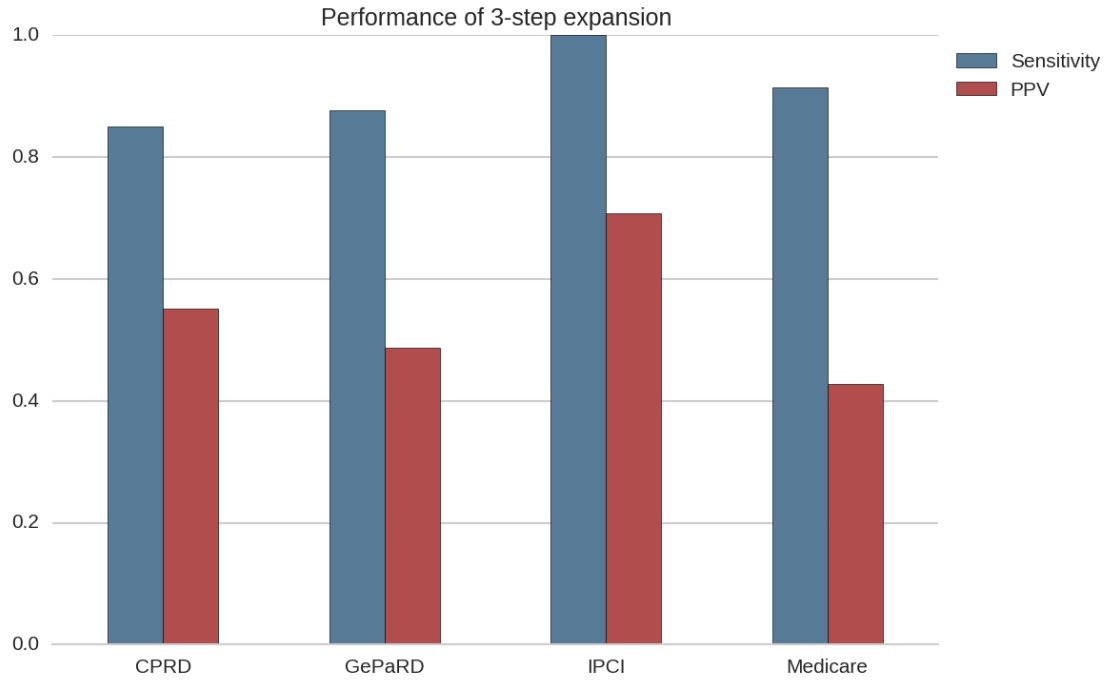

```
In [16]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-CHD-RB-PAR.expand', '1 step'),
             ('2-RN-CHD-RB-PAR.expand', '2 steps'),
             ('3-RN-CHD-RB-PAR.expand', '3 steps'),
             ('4-RN-CHD-RB-PAR.expand', '4 steps'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').recall.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(event_names.get)
         averages_compare.index = compare_variations.values()

         with mystyle(graded_recall_palette(len(compare_variations)), xrot=45, ha='right', savefig='step
             averages_compare.T.plot(kind='bar', title="Expansion steps")
             plt.ylabel(measure_label('recall'))
```




```
In [17]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('1-RN-CHD-RB-PAR.expand', '1 step'),
             ('2-RN-CHD-RB-PAR.expand', '2 steps'),
             ('3-RN-CHD-RB-PAR.expand', '3 steps'),
             ('4-RN-CHD-RB-PAR.expand', '4 steps'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').precision.mean()
             for variation in compare_variations
         ], index=compare_variations)
         averages_compare.columns = averages_compare.columns.map(event_names.get)
         averages_compare.index = compare_variations.values()

         with mystyle(graded_precision_palette(len(compare_variations)), xrot=45, ha='right', savefig='
             averages_compare.T.plot(kind='bar', title="Expansion steps")
             plt.ylabel(measure_label('precision'))
```

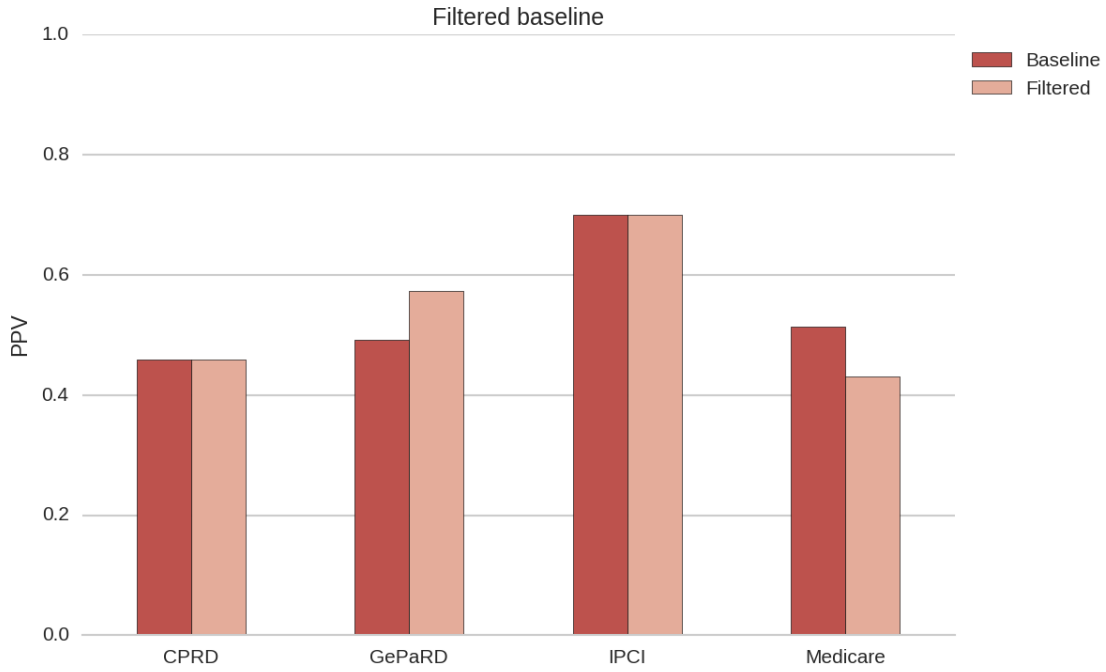



```
In [18]: measures = ['recall', 'precision']
         averages_compare = pd.DataFrame([
             df[df.variation == '3-RN-CHD-RB-PAR.expand'].groupby('database')[measure].mean()
             for measure in measures
         ], index=map(measure_label, measures))
         averages_compare.columns = averages_compare.columns.map(database_label)
         #averages_compare.index = compare_variations.values()

         with mystyle(measures_palette, savefig='expansion3-performance-by-db.pdf'):
             averages_compare.T.plot(kind='bar', title="Performance of 3-step expansion")
```

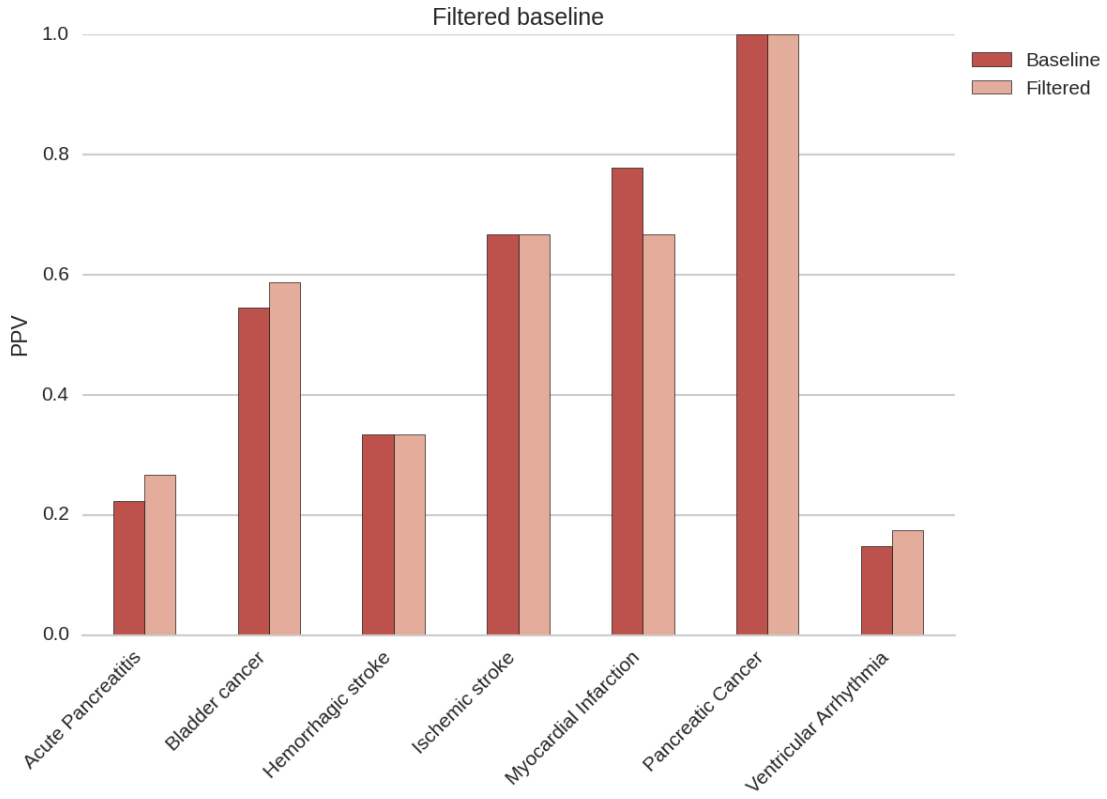


7 Removing unused codes

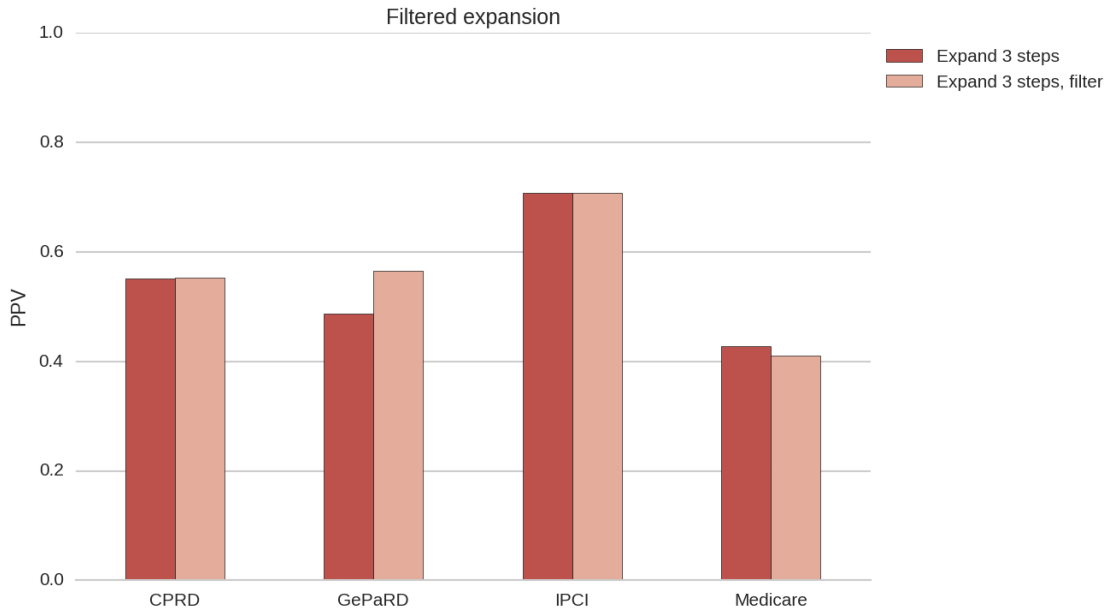
```
In [19]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('baseline.filter-gen', 'Filtered'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('database').precision.mean()
             for variation in compare_variations
         ], index = compare_variations.values())
         averages_compare.columns = averages_compare.columns.map(database_label)

         with mystyle(graded_precision_palette(len(compare_variations)), savefig='filtered-baseline-pre
             averages_compare.T.plot(kind='bar', title="Filtered baseline")
             plt.ylabel(measure_label('precision'))
```



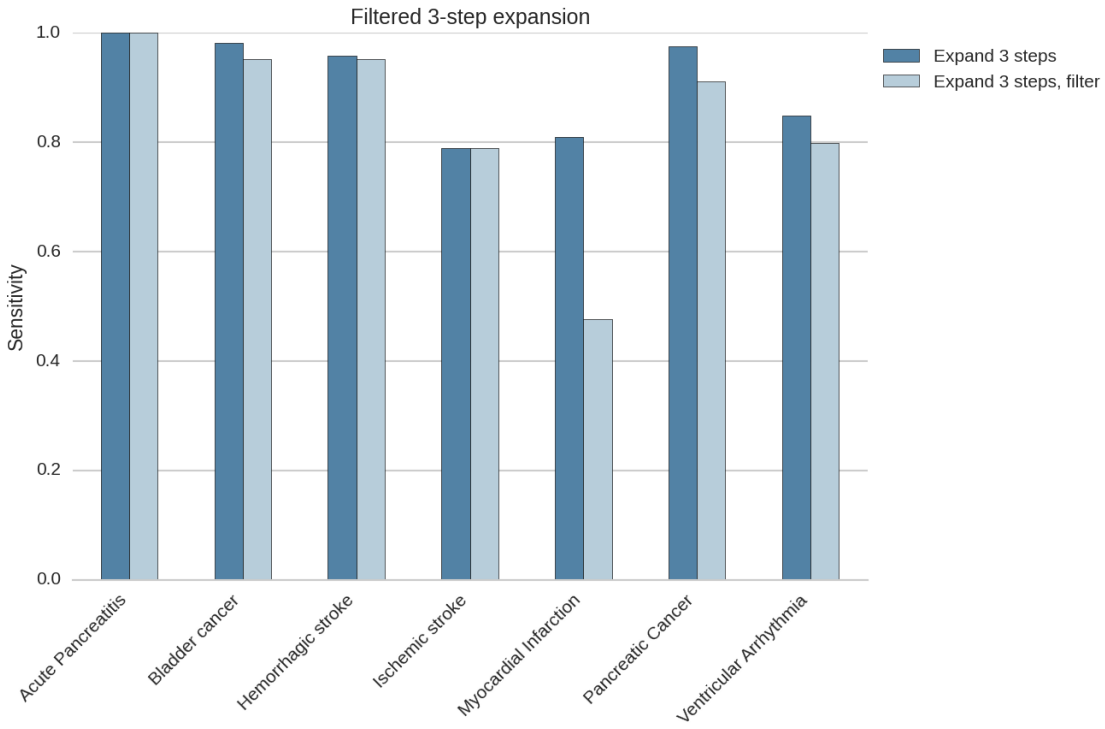

```
In [20]: compare_variations = OrderedDict([
             ('baseline', 'Baseline'),
             ('baseline.filter-gen', 'Filtered'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').precision.mean()
             for variation in compare_variations
         ], index = compare_variations.values())
         averages_compare.columns = averages_compare.columns.map(event_label)

         with mystyle(graded_precision_palette(len(compare_variations)), xrot=45, ha='right', savefig='
             averages_compare.T.plot(kind='bar', title="Filtered baseline")
             plt.ylabel(measure_label('precision'))
```



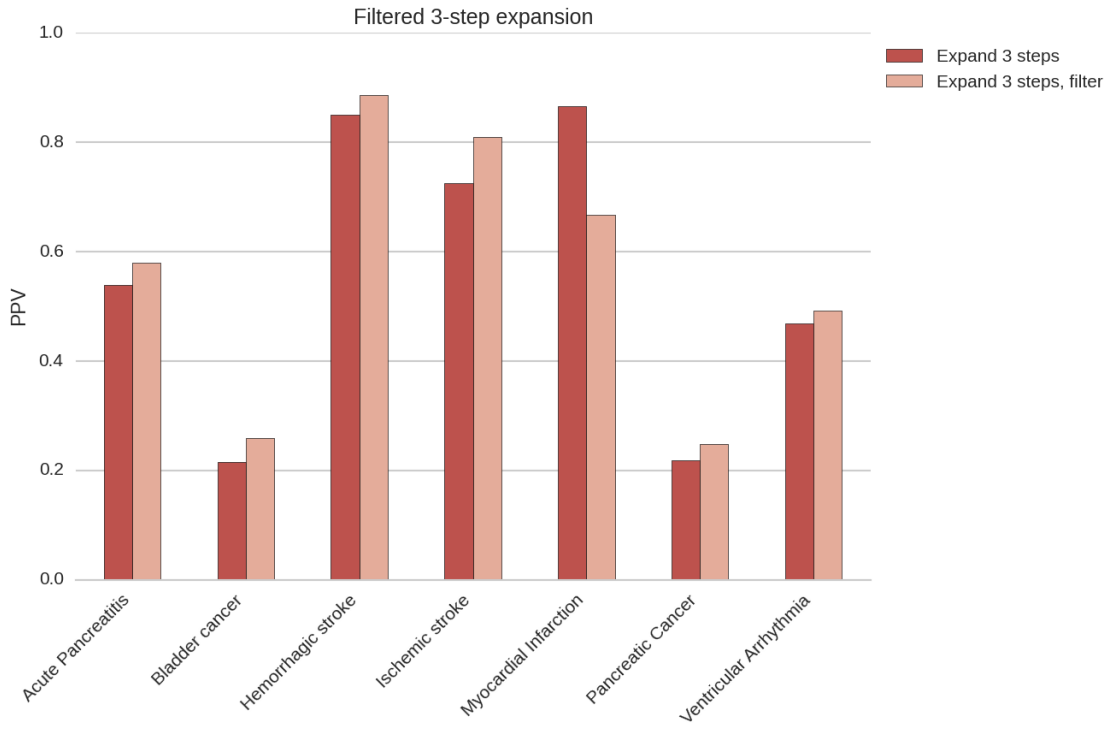

```
In [21]: compare_variations = OrderedDict([
             ('3-RN-CHD-RB-PAR.expand', 'Expand 3 steps'),
             ('3-RN-CHD-RB-PAR.expand.filter-gen', 'Expand 3 steps, filter'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('database').precision.mean()
             for variation in compare_variations
         ], index = compare_variations.values())
         averages_compare.columns = averages_compare.columns.map(database_label)

         with mystyle(graded_precision_palette(len(compare_variations)), savefig='filtered-expansion3-p
             averages_compare.T.plot(kind='bar', title="Filtered expansion")
             plt.ylabel(measure_label('precision'))
```



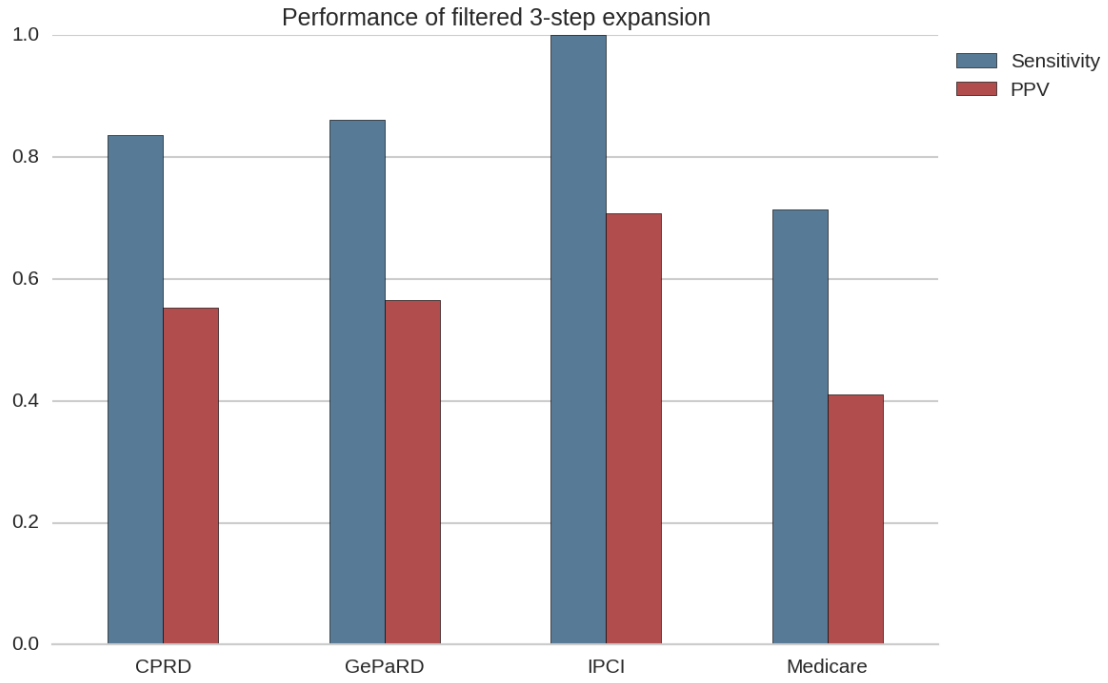

```
In [22]: compare_variations = OrderedDict([
             ('3-RN-CHD-RB-PAR.expand', 'Expand 3 steps'),
             ('3-RN-CHD-RB-PAR.expand.filter-gen', 'Expand 3 steps, filter'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').recall.mean()
             for variation in compare_variations
         ], index = compare_variations.values())
         averages_compare.columns = averages_compare.columns.map(event_names.get)

         with mystyle(graded_recall_palette(len(compare_variations)), xrot=45, ha='right', savefig='fil
             averages_compare.T.plot(kind='bar', title="Filtered 3-step expansion")
             plt.ylabel(measure_label('recall'))
```




```
In [23]: compare_variations = OrderedDict([
             ('3-RN-CHD-RB-PAR.expand', 'Expand 3 steps'),
             ('3-RN-CHD-RB-PAR.expand.filter-gen', 'Expand 3 steps, filter'),
         ])
         averages_compare = pd.DataFrame([
             df[df.variation == variation].groupby('event').precision.mean()
             for variation in compare_variations
         ], index = compare_variations.values())
         averages_compare.columns = averages_compare.columns.map(event_names.get)

         with mystyle(graded_precision_palette(len(compare_variations)), xrot=45, ha='right', savefig='
             averages_compare.T.plot(kind='bar', title="Filtered 3-step expansion")
             plt.ylabel(measure_label('precision'))
```




```
In [24]: measures = ['recall', 'precision']
         averages_compare = pd.DataFrame([
             df[df.variation == '3-RN-CHD-RB-PAR.expand.filter-gen'].groupby('database')[measure].mean()
             for measure in measures
         ], index=map(measure_label, measures))
         averages_compare.columns = averages_compare.columns.map(database_label)
         #averages_compare.index = compare_variations.values()

         with mystyle(measures_palette, savefig='filtered-expansion3-performance-by-db.pdf'):
             averages_compare.T.plot(kind='bar', title="Performance of filtered 3-step expansion")
```



The drop in PPV for Myocardial infarction is caused by the mapping to codes `410.*` (Acute myocardial infarction) in Medicare which is *not* used in the ARS database.

8 Codes in reference mappings, not in databases

Codes that might be removed from the TP when filtering.

In [25]:
```
stats = DataFrame()
stats['In ref'] = code_stats[code_stats.InMapping]\
    .groupby('Database').Code.count()
stats['Not in DB'] = code_stats[code_stats.InMapping & ~code_stats.InDatabase]\
    .groupby('Database').Code.count()
stats.fillna(0, inplace=True)
stats['%'] = (stats['Not in DB'] / stats['In ref']).map("{:.2%}".format)
stats['Codes'] = code_stats[code_stats.InMapping & ~code_stats.InDatabase]\
    .groupby('Database').Code.aggregate(lambda vs: ', '.join(set(vs)))
stats
```

Out[25]:

Database	In ref	Not in DB	%	Codes
CPRD	229	5	2.18%	Gyu60, ByuF., 8531., B91zz, BB46.
GePaRD	72	1	1.39%	C25
IPCI	7	0	0.00%	NaN
Medicare	50	14	28.00%	410.6, 410.9, 410.4, 410.5, 410.1, 188, 410.3, 410.7, 410.0, 157, 410, 410.2, 41

In [26]: