Stage 3 - Entity Mathcing

In [1]:
```
import pandas as pd
import py_entitymatching as em
```

1. Data

The type of entity we want to match is company. The two tables are Forbes lists crawled from Forbes (http://forbes.com/) and NASDAQ company information downloaded from NASDAQ (http://www.nasdaq.com/). See detailed information about our data below.

In [2]:
```
data_dir = './dataset/structured_data/'
```

1.1 Forbes data

We collected company information from multiple ranking lists from Forbes website. The total tuples within this company infomation list is 3,110.

The attributes used in this list are: company name, country, industry, sales, profits, assets, market value, employee number.

In [3]:
```
forbes_filename = 'forbes_all.csv'
forbes_df = pd.read_csv(open(data_dir + forbes_filename, encoding = "I
SO-8859-1"))
print("# tuples:", len(forbes_df))
```

```
# tuples: 3110
```

Here is some sample tuples.

In [4]:
```
forbes_df[:5]
```

Out[4]:

	id	Company	Country	Industry	Sales (M)	Profits (M)	Assets (M)	Market Value (M)	Employee
0	1	77 Bank	Japan	Banks	853	165	69100	1400	-
1	2	Abu Dhabi Commercial Bank	United Arab Emirates	Banks	2800	1300	62100	11000	-
2	3	Abu Dhabi Islamic Bank	United Arab Emirates	Banks	1600	434	24300	3800	-
3	4	Agricultural Bank of China	China	Banks	131900	28800	2739800	152700	-
4	5	Ahli United Bank	Bahrain	Banks	1400	524	34000	4200	-

NASDAQ data

We collected company stock information from NASDAQ websites. The total number of tuples found is 4,714. Attributes used in this list are: symbol, name, lastsale, marketcap, country, IPOyear, sector, industry, summary quote.

In [5]:
```
nasdaq_filename = 'nasdaq.csv'
nasdaq_df = pd.read_csv(data_dir + nasdaq_filename)
print("# tuples:", len(nasdaq_df))
```

```
# tuples: 4714
```

Here is some sample tuples.

In [6]:
```
nasdaq_df[:5]
```

Out[6]:

	id	Symbol	Name	LastSale	MarketCap	Country	IPOyear	Sector	In
0	1	FLWS	1-800 FLOWERS.COM, Inc.	10.15	665.525939	United States	1999	Consumer Services	O St
1	2	PIH	1347 Property Insurance Holdings, Inc.	8.20	48.845481	United States	2014	Finance	Pr Ca
2	3	TURN	180 Degree Capital Corp.	1.45	44.811103	United States	n/a	Finance	Fi Se
3	4	FCCY	1st Constitution Bancorp (NJ)	18.50	148.505827	United States	n/a	Finance	Sa In
4	5	SRCE	1st Source Corporation	46.95	1262.613016	United States	n/a	Finance	M

We are to make use of the fields of name, industry, country and market value. For the market value attribute, we have market value in the Forbes list and market cap in the NASDAQ list. Even though there are different terms, usually they represents the value of each company and have similar values. Thus, we treat them as comparable attributes and use them all as market value.

In [7]:
```
all_fields = ['Name', 'Industry', 'MarketValue', 'Country']
```

In [8]:
```
forbes = forbes_df.rename(columns={'Company':'Name', 'Market Value (M)
':'MarketValue'})
```

In [9]:
```
nasdaq = nasdaq_df.rename(columns={'industry':'Industry', 'MarketCap':
'MarketValue'})
```

Sampling

For the ease of exploration and developement.

In [10]:
```
nasdaq_sample = nasdaq[:200]
nasdaq_sample.to_csv(data_dir + 'nasdaq_sample.csv')
```

In [11]:
```
forbes_sample = forbes[:100]
forbes_sample.to_csv(data_dir + 'forbes_sample.csv')
```

In [ ]:

Stage 3 - Entity Mathcing

In [1]:
```
from collections import Counter
import os

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd

import py_entitymatching as em

IS_DEVELOPING = False

data_dir = './dataset/structured_data/'
A_filename = 'forbes_sample.csv' if IS_DEVELOPING else 'forbes_all.csv
'
B_filename = 'nasdaq_sample.csv' if IS_DEVELOPING else 'nasdaq.csv'
blocked_filename = 'blocked_sample.csv' if IS_DEVELOPING else 'blocked
.csv'
```

2 Blocking

In [2]:
```
A = pd.read_csv(data_dir + A_filename , encoding = "ISO-8859-1")
B = pd.read_csv(data_dir + B_filename , encoding = "ISO-8859-1")
```

In [3]:
```
A = A.rename(columns={'Company':'Name', 'Market Value (M)':'MarketValu
e', 'Profits (M)':'Profits' ,'Sales (M)':'Sales', 'Assets (M)':'Assets
'})
B = B.rename(columns={'industry':'Industry', 'MarketCap':'MarketValue'
})
em.set_key(A, 'id')
em.set_key(B, 'id')
A.to_csv(data_dir +"forbes_all_rename.csv")
B.to_csv(data_dir +"nasdaq_rename.csv")
```

In [4]:
```
A
all_fields_A = ['Name', 'Country' , 'Industry', 'Sales', 'Profits', 'A
ssets', 'MarketValue', 'Employee']
all_fields_A = ['Country' , 'Industry', 'MarketValue', 'Name' ]
```

In [5]:
```
B
all_fields_B = ['Symbol', 'Name', 'Country' , 'Sector' , 'Industry', '
LastSale', 'MarketValue']
all_fields_B = ['Name', 'Country' , 'Sector' , 'Industry','MarketValue
']
```

2.1 Overlap blocker to initial the blocks.

Using overlap blocker.

1. We first tokenize the names then use overlap size of 1 to block candidates.
2. After the first attemp, we found there are some common stop words can be useful to eliminate impossible pairs, then we added these common stop words into the block. eg., of, property, gas, oil.

The number of pairs after this first step is 40,908.

In [6]:
```
ob = em.OverlapBlocker()
```

Rule out common words.

In [7]:
```
ob.stop_words.extend(w.lower() for w in [
    'of',
    'property', 'holdings', 'holding', 'inc', 'bancorp', 'Bancorporati
on', 'plc',
    'Corporation', 'Corp', 'Group', 'Company',
    'Advanced',
    'Insurance', 'Bank', 'Financial', 'Pharmaceuticals', 'Pharma', 'Th
erapeutics',
    'Systems', 'Technologies', 'Technology', 'Express', 'Air', 'Securi
ties', 'Security'
    'Mining', 'Energy', 'Electric', 'Gas', 'Oil', 'Power'])
ob.stop_words = list(set(ob.stop_words))
print(*ob.stop_words, sep=', ')
```

```
holding, corporation, technology, inc, air, will, energy, plc, with,
be, systems, technologies, an, and, a, for, that, electric, gas, on,
group, was, by, power, is, to, at, were, it, company, therapeutics,
he, property, as, the, advanced, bank, oil, in, holdings, of, from,
bancorporation, its, corp, securitymining, are, financial, pharmaceu
ticals, securities, has, express, bancorp, pharma, insurance
```

In [8]:
```
## jz: not yet sure how to combine multiple occurrence of same compani
es, e.g. "Zions Bancorp" vs "Zions Bancorp."
# AA = ob.block_tables(A, A, 'Name', 'Name', word_level=True, overlap_
size=1, rem_stop_words=True,
#                       l_output_attrs=all_fields,
#                       r_output_attrs=all_fields,
#                       show_progress=False)
# print(len(AA))
# AA.head()
```

In [9]:
```
# Specify the tokenization to be 'word' level and set overlap_size to
be 3.
block_overlap = ob.block_tables(A, B, 'Name', 'Name', word_level=True,
overlap_size=1, rem_stop_words=True,
                    l_output_attrs=all_fields_A,
                    r_output_attrs=all_fields_B,
                    show_progress=False)
print('# pairs:', len(block_overlap))
# Display first 5 tuple pairs in the candidate set.
block_overlap.head(300)
```

```
# pairs: 40908
```

Out[9]:

	_id	ltable_id	rtable_id	ltable_Country	ltable_Industry	ltable_MarketValue	ltable
0	0	3073	3	United States	Utilities: Gas and Electric	8343	Pinna West
1	1	260	3	Thailand	Banks	1300	Thana Capit
2	2	2533	3	Bermuda	Property & Casualty Insurance	8600	Arch (Group
3	3	1164	3	United States	Diversified Financials	-	Capit Group
4	4	388	3	JAPAN	-	298	M&A Partn
5	5	334	3	Hong Kong	-	537	Empe Capit Group
6	6	2579	3	United States	Real Estate	6200	Ameri Capit Agen

7	7	228	3	Malaysia	Banks	4900	RHB (
8	8	2581	3	United States	Real Estate	9500	Annal Capit Mana
9	9	1080	3	United States	Consumer Financial Services	39200	Capit Finan
10	10	2586	3	China	Real Estate	3600	Beijin Capit Devel
11	11	1184	3	Bermuda	Diversified Insurance	5100	Axis (Holdi
12	12	2676	7	Japan	Rental & Leasing	3800	Centu Tokyo Leasi
13	13	1142	7	Taiwan	Diversified Chemicals	4100	Far Ea New (
14	14	1463	7	United States	Entertainment	-	Twent Centu
15	15	312	7	PAKISTAN	-	11	Centu Insura
16	16	925	10	United States	Conglomerates	102200	3M
17	17	2513	12	United States	Pipelines	3616	Enabl Midst Partn
18	18	388	12	JAPAN	-	298	M&A Partn
19	19	1962	12	Switzerland	Investment Services	10700	Partn Group Holdi
							Enter

20	20	2515	12	United States	Pipelines	-	Produ Partn
21	21	3087	12	United States	Wholesalers: Diversified	459	Globa Partn
22	22	2512	12	United States	Pipelines	8813	Bucke Partn
23	23	2449	12	United States	Petroleum Refining	892	Calun Speci Produ Partn
24	24	1427	12	United States	Energy	1705	Ferrel Partn
25	25	2517	12	United States	Pipelines	15671	Mage Midst Partn
26	26	1423	12	United States	Energy	806	Crest Equity Partn
27	27	682	12	United States	Beverages	11547	Coca- Europ Partn
28	28	3093	12	United States	Wholesalers: Diversified	805	NGL I Partn
29	29	1734	14	United States	Homebuilders	1458	Merita Home
...	...	...	...	...	...	...	...
270	270	2110	42	United States	Managed Health Care	3700	Molin Healtl
271	271	1679	42	United States	Health Care: Pharmacy and	3816	Envis Healtl

					Other Services		Holdi
272	272	2657	44	United States	Real Estate	10500	SL Gr Realty
273	273	1026	44	Japan	Construction Services	11200	Daito Cons
274	274	2659	44	Japan	Real Estate	15200	Sumit Realty
275	275	196	44	Philippines	Banks	5800	Metro Bank
276	276	2631	44	United States	Real Estate	11700	Kimc
277	277	2603	44	United States	Real Estate	12600	Digita Trust
278	278	2668	44	United States	Real estate	17807	Vorna Realty
279	279	142	44	Nigeria	Banks	2300	Guara Trust
280	280	2607	44	United States	Real Estate	14700	Essex Prope Trust
281	281	1936	44	Netherlands	Investment Services	15700	HAL T
282	282	212	44	United States	Banks	16600	North Trust
283	283	250	44	Japan	Banks	13000	Sumit Mitsu
284	284	284	44	Hong Kong	-	4738	Anxin
285	285	2667	44	United States	Real Estate	18000	Vorna Realty
286	286	1691	45	United States	Healthcare Services	10700	Quest Diagn
					Oil Services &		Natio

287	287	2375	47	United States	Equipment	11000	Oilwe
288	288	202	47	United Arab Emirates	Banks	12800	Natio Bank Dhab
289	289	203	47	Canada	Banks	12000	Natio Bank Cana
290	290	204	47	Greece	Banks	3000	Natio Bank Greec
291	291	269	47	United Arab Emirates	Banks	2800	Union Natio Bank
292	292	206	47	Saudi Arabia	Banks	22400	Natio Comr Bank
293	293	2192	47	China	Mining, Crude-Oil Production	-	China Natio Offsh
294	294	17	47	Saudi Arabia	Banks	5500	Arab Natio Bank
295	295	2450	47	China	Petroleum Refining	-	China Natio Petro
296	296	2392	47	China	Other Industrial Equipment	990	China Natio Mater
297	297	1753	47	United States	Hotels, Casinos, Resorts	1357	Penn Natio Gami
298	298	3039	47	United States	Transportation	-	Schn Natio
							Punja Natio

299	299	223	47	India	Banks	2700	Bank

300 rows × 12 columns

In [10]:
```
# try out blocker over a single pair
# ob.block_tuples(A.ix[60], B.ix[0], l_overlap_attr='Name', r_overlap_attr='Name',
#                 rem_stop_words=True, word_level=True, overlap_size=1)
```

2.2 Equivalence attribute blocker

We applied equivalence blocker on the previous overlap blocker results using same country name. This further cut down the number of pairs to 22,721.

In [11]:
```
## debug and improve the blocker
## This may take some trial and run...
## Block on the same country
cb = em.AttrEquivalenceBlocker()
block_country = cb.block_candset(block_overlap, 'Country', 'Country')
print('# pairs:', len(block_country))
```

```
0%                          100%
[##############################] | ETA: 00:00:01 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00
Total time elapsed: 00:00:00

# pairs: 22721
```

In [12]:
```
block_country.head()
```

Out[12]:

	_id	ltable_id	rtable_id	ltable_Country	ltable_Industry	ltable_MarketValue	ltable_N
0	0	3073	3	United States	Utilities: Gas and Electric	8343	Pinnacle West Ca
3	3	1164	3	United States	Diversified Financials	-	Capital Group C
6	6	2579	3	United States	Real Estate	6200	Americar Capital Agency
8	8	2581	3	United States	Real Estate	9500	Annaly Capital Manager
9	9	1080	3	United States	Consumer Financial Services	39200	Capital C Financial

2.3 Hashing blocker

The following step we applied hashing blocker to the company names:

1. If the first word in the company name is some common key words, eg. China, First, United, then we compare the second word.
2. If the first word is not in the kw list, we just compare the first and second letters in the first words from the two candidates.

After this step, we have number of candiate pairs to be 2,054.

In [13]:
```
## Hash Blocker
def hash_blocker(ltuple, rtuple):
    kw = ['China', 'American','United','First']
    # NOTE
    # some companies started with these common keywords, thus to do ha
shing more efficiently, we compare the second word.

    l_sp = ltuple['Name'].split()
    r_sp = rtuple['Name'].split()
    if l_sp[0] in kw or r_sp[0] in kw:
        if l_sp[0] == r_sp[0]:
            if  l_sp[1][0:2] != r_sp[1][0:2]:
                return True
            else:
                return False
        else:
            return True
    elif ltuple['Name'][0:2] != rtuple['Name'][0:2]:
        #print (ltuple['Name'][0],rtuple['Name'][0])
        return True
    else:
        return False
def hash_all (ltuple,rtuple):
    return False
```

In [14]:
```
hb = em.BlackBoxBlocker()
hb.set_black_box_function(hash_blocker)
```

In [15]:
```
block_hash = hb.block_candset(block_country)
print('# pairs:', len(block_hash))
block_hash.head(30)
```

```
0%                            100%
[##############################] | ETA: 00:00:06 | ETA: 00:00:04 | E
TA: 00:00:04 | ETA: 00:00:04 | ETA: 00:00:04 | ETA: 00:00:03 | ETA:
00:00:03 | ETA: 00:00:03 | ETA: 00:00:02 | ETA: 00:00:02 | ETA: 00:0
0:02 | ETA: 00:00:02 | ETA: 00:00:02 | ETA: 00:00:02 | ETA: 00:00:01
| ETA: 00:00:01 | ETA: 00:00:01 | ETA: 00:00:01 | ETA: 00:00:01 | ET
A: 00:00:01 | ETA: 00:00:01 | ETA: 00:00:01 | ETA: 00:00:00 | ETA: 0
0:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00
:00 | ETA: 00:00:00 | ETA: 00:00:00 | ETA: 00:00:00

# pairs: 2054

Total time elapsed: 00:00:03
```

Out[15]:

	_id	ltable_id	rtable_id	ltable_Country	ltable_Industry	ltable_MarketValue	ltabl
16	16	925	10	United States	Conglomerates	102200	3M

30	30	2217	15	United States	Miscellaneous	139	A-Ma Prec Meta
34	34	829	16	United States	Chemicals	798	A. Sc
39	39	2464	23	United States	Pharmaceuticals	64900	Abbo Labo
40	40	2465	24	United States	Pharmaceuticals	99400	AbbV
41	41	2842	26	United States	Specialty Retailers: Apparel	2132	Aber Fitch
226	226	1254	38	United States	Diversified Outsourcing Services	1812	ABM Indu
328	328	1408	48	United States	Electronics, Electrical Equip.	9553	Acui
433	433	2576	62	United States	Recreational Products	25500	Activ Blizz
443	443	1408	65	United States	Electronics, Electrical Equip.	9553	Acui
449	449	2448	70	United States	Petroleum Refining	169	Adar Reso Ener
467	467	2448	71	United States	Petroleum Refining	169	Adar Reso Ener
493	493	2448	72	United States	Petroleum Refining	169	Adar Reso Ener
538	538	2790	79	United States	Software & Programming	47400	Adol Syst
545	545	2878	82	United States	Specialty Stores	11600	Adva Parts
546	546	2126	82	United States	Media	-	Adva Publ
					Semiconductors		

625	625	2757	87	United States	and Other Electronic Components	2261	Adva Micr
634	634	2731	88	Taiwan	Semiconductors	8300	Adva Sem
664	664	1014	95	United States	Construction Services	4700	AEC Tech
665	665	1438	95	United States	Engineering, Construction	4699	AEC
674	674	2105	105	United States	Managed Health Care	40200	Aetn
677	677	1902	107	United States	Investment Services	9600	Affili Man Grou
678	678	2029	108	United States	Life & Health Insurance	28500	Aflac
706	706	987	110	United States	Construction and Farm Machinery	4098	AGC
708	708	1376	113	United States	Electronics	13700	Agile Tech
815	815	2811	124	United States	Specialized Chemicals	31900	Air P Cher
862	862	2168	130	United States	Metals	737	AK S Hold
867	867	886	131	United States	Computer Services	9000	Akar Tech
868	868	521	139	United States	Airline	9500	Alasl Grou
871	871	521	140	United States	Airline	9500	Alasl Grou

In [22]:
```
block_hash.to_csv(data_dir+"blocking_results.csv")
```

2.4 Debug the blocker results.

We used the debug_blocker function to generate the 200 most likely matched pairs from the eliminated pairs, and the results are printed below. As we examined, most of them are different entities. This means we have reasonable blocking results. And most of the eliminated pairs are not likely to be matching pairs.

In [16]:
```
corres = [('Name', 'Name'), ('Country', 'Country'),('Industry','Indust
ry')]
dbg = em.debug_blocker(block_hash,A,B)
dbg
```

Out[16]:

	_id	similarity	ltable_id	rtable_id	ltable_Name	ltable_Country	ltable_Industry	lt
0	0	0.500000	1087	3137	Jaccs	Japan	Consumer Financial Services	6
1	1	0.500000	2614	3802	General Growth Properties	United States	Real Estate	2
2	2	0.461538	2326	1105	Oil & Gas Development	Pakistan	Oil & Gas Operations	4
3	3	0.461538	2326	713	Oil & Gas Development	Pakistan	Oil & Gas Operations	4
4	4	0.461538	2327	4703	Oil & Natural Gas	India	Oil & Gas Operations	2
5	5	0.461538	2326	4703	Oil & Gas Development	Pakistan	Oil & Gas Operations	4
6	6	0.461538	2327	1105	Oil & Natural Gas	India	Oil & Gas Operations	2
7	7	0.461538	2327	713	Oil & Natural Gas	India	Oil & Gas Operations	2

8	8	0.461538	2326	3735	Oil & Gas Development	Pakistan	Oil & Gas Operations	4
9	9	0.461538	2327	3735	Oil & Natural Gas	India	Oil & Gas Operations	2
10	10	0.454545	2333	3127	Petroleum Traders Corporation	United States	Oil & Gas Operations	-
11	11	0.454545	2333	4602	Petroleum Traders Corporation	United States	Oil & Gas Operations	-
12	12	0.454545	2333	270	Petroleum Traders Corporation	United States	Oil & Gas Operations	-
13	13	0.454545	2333	39	Petroleum Traders Corporation	United States	Oil & Gas Operations	-
14	14	0.454545	2333	1632	Petroleum Traders Corporation	United States	Oil & Gas Operations	-
15	15	0.444444	2069	467	Bank of America	United States	Major Banks	1
16	16	0.444444	2094	3307	PNC Financial Services	United States	Major Banks	4
17	17	0.444444	2071	471	Bank of Montreal	Canada	Major Banks	4
18	18	0.444444	2069	471	Bank of America	United States	Major Banks	1
19	19	0.444444	2290	1008	CNOOC	China	Oil & Gas Operations	5
20	20	0.444444	2070	467	Bank of China	China	Major Banks	1
21	21	0.444444	2094	1511	PNC Financial Services	United States	Major Banks	4

22	22	0.444444	2070	464	Bank of China	China	Major Banks	14
23	23	0.444444	2071	466	Bank of Montreal	Canada	Major Banks	4
24	24	0.444444	2070	466	Bank of China	China	Major Banks	14
25	25	0.444444	2070	471	Bank of China	China	Major Banks	14
26	26	0.444444	2101	758	TD Bank Group	Canada	Major Banks	8
27	27	0.444444	2071	467	Bank of Montreal	Canada	Major Banks	4
28	28	0.444444	2071	464	Bank of Montreal	Canada	Major Banks	4
29	29	0.444444	2092	2893	National Australia Bank	Australia	Major Banks	5
...	...	...	...	...	...	...	...	...
170	170	0.375000	2631	4136	Kimco Realty	United States	Real Estate	1
171	171	0.375000	2095	1661	Regions Financial	United States	Major Banks	1
172	172	0.375000	2102	1510	US Bancorp	United States	Major Banks	7
173	173	0.375000	2095	880	Regions Financial	United States	Major Banks	1
174	174	0.375000	2102	3230	US Bancorp	United States	Major Banks	7

175	175	0.375000	2095	775	Regions Financial	United States	Major Banks	1
176	176	0.375000	2095	4093	Regions Financial	United States	Major Banks	1
177	177	0.375000	2102	3303	US Bancorp	United States	Major Banks	7
178	178	0.375000	2102	2057	US Bancorp	United States	Major Banks	7
179	179	0.375000	2667	4136	Vornado Realty	United States	Real Estate	1
180	180	0.375000	2617	1776	GPT Group	Australia	Real Estate	6
181	181	0.375000	2606	200	Equity Residential	United States	Real Estate	2
182	182	0.375000	2601	3802	Damac Properties	United Arab Emirates	Real Estate	4
183	183	0.375000	2578	4384	Aldar Properties	United Arab Emirates	Real Estate	6
184	184	0.375000	2102	3304	US Bancorp	United States	Major Banks	7
185	185	0.375000	2647	2427	Realogy Holdings	United States	Real estate	5
186	186	0.375000	2102	1185	US Bancorp	United States	Major Banks	7
187	187	0.375000	2102	1638	US Bancorp	United States	Major Banks	7
188	188	0.375000	2095	4323	Regions Financial	United States	Major Banks	1
189	189	0.375000	2095	2700	Regions Financial	United States	Major Banks	1

190	190	0.375000	2591	1776	CBRE Group	United States	Real Estate	1
191	191	0.375000	2626	1102	Henderson Land	Hong Kong	Real Estate	2
192	192	0.375000	2102	2925	US Bancorp	United States	Major Banks	7
193	193	0.375000	2665	4427	Vereit	United States	Real Estate	8
194	194	0.375000	2095	3977	Regions Financial	United States	Major Banks	1
195	195	0.375000	2601	4384	Damac Properties	United Arab Emirates	Real Estate	4
196	196	0.375000	2102	1637	US Bancorp	United States	Major Banks	7
197	197	0.375000	2588	1102	British Land	United Kingdom	Real Estate	1
198	198	0.375000	2667	1882	Vornado Realty	United States	Real Estate	1
199	199	0.375000	2605	3802	Emaar Properties	United Arab Emirates	Real Estate	1

200 rows × 12 columns

2.5 Generat 300 samples from the final blocking results.

We generated 5 small samples sets with about 50 pairs in each. After screening these small samples sets, we found reasonable number of matching pairs in each sample set. Thus, we used these final blocking results to generate 300 samples to label and then used as the development stage dataset.

In [17]:
```
sample_table = em.sample_table(block_hash,300,verbose=True)
```

Finally save the results from blocking step. Then manually label these 300 tuple pairs. In these labeled sample set G, we have 160 positive matching tuples and 140 negative matching tuples.

In [18]:
```
label_table = em.label_table(sample_table, label_column_name='is_match
')
```

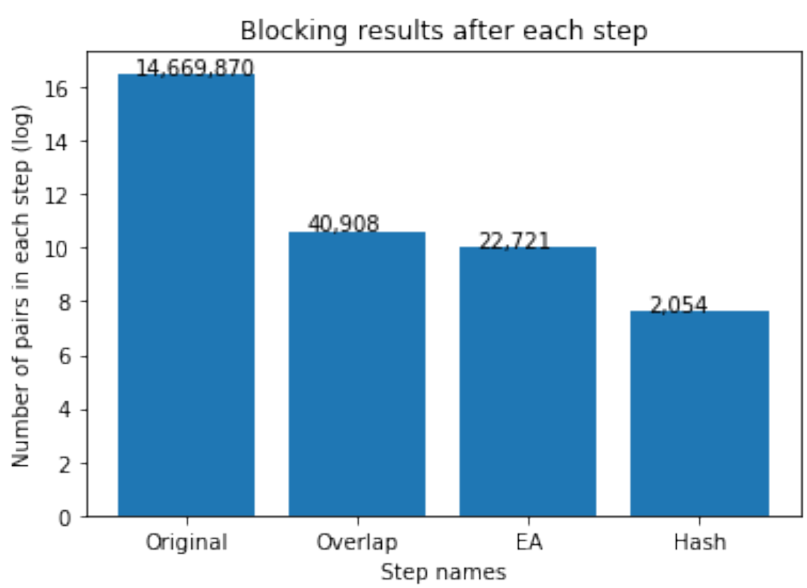
```
Column name (is_match) is not present in dataframe
```

In [19]:
```
label_table.to_csv(data_dir + blocked_filename)
# After save to the blocked file, we labeled them and rename it to "go
lden_label.csv"
```

2.6 Blocking summary

After the hierachical blocking steps, we finally reduced the possible matching pairs from 4717*3110 to 2054.

In [20]:
```
plt.bar([1,2,3,4],[np.log(4717*3110),np.log(40908), np.log(22721),np.l
og(2054) ]);
plt.ylabel("Number of pairs in each step (log)");
plt.xlabel("Step names");
plt.xticks([1,2,3,4],["Original","Overlap","EA","Hash"]);
plt.title("Blocking results after each step");
plt.text(0.7,np.log(4717*3110),"14,669,870");
plt.text(1.7,np.log(40908),"40,908");
plt.text(2.7,np.log(22721),"22,721");
plt.text(3.7,np.log(2054),"2,054");
```



The blocking power is: 1-2054/14669870 = 99.99%

Stage 3 - Entity Matching

In [1]:
```
from collections import Counter

import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd

import py_entitymatching as em

IS_DEVELOPING = False

data_dir = './dataset/structured_data/'
A_filename = 'forbes_sample.csv' if IS_DEVELOPING else 'forbes_all_rename.csv'
B_filename = 'nasdaq_sample.csv' if IS_DEVELOPING else 'nasdaq_rename.csv'
blocked_filename = 'blocked_sample.csv' if IS_DEVELOPING else 'blocked.csv'
labeled_filename = 'labeled_sample.csv' if IS_DEVELOPING else 'golden_label.csv'

all_fields = ['Name',"Country", 'Industry', "MarketValue"]

# Set the seed value for reproducibility
seed = 0
```

3 Mathcing

In [2]:
```
A = pd.read_csv(data_dir + A_filename,  encoding = "ISO-8859-1" )
A = A[['id'] + all_fields]
em.set_key(A, 'id')
B = pd.read_csv(data_dir + B_filename,  encoding = "ISO-8859-1")
B = B[['id'] + all_fields]
em.set_key(B, 'id')
# Load the pre-labeled data
S = em.read_csv_metadata(data_dir + labeled_filename,
                         key='_id',
                         ltable=A, rtable=B,
                         fk_ltable='ltable_id', fk_rtable='rtable_id',
encoding = "ISO-8859-1")
```

```
Metadata file is not present in the given path; proceeding to read t
he csv file.
```

3.1 Split training set ¶

Split the labeled data into development set and evaluation set. Use the development set to select the best learning-based matcher.

In [3]:
```
# Split S into I an J
IJ = em.split_train_test(S, train_proportion=0.6, random_state=seed)
I = IJ['train']
J = IJ['test']
print(Counter(I.is_match))
print(Counter(J.is_match))
```

```
Counter({1: 100, 0: 80})
Counter({0: 60, 1: 60})
```

Save split I, J into files.

In [4]:
```
I.to_csv(data_dir +"I.csv")
J.to_csv(data_dir +"J.csv")
```

3.2 Creating features

In [5]:
```
# Generate a set of features
F = em.get_features_for_matching(A, B)
```

In [6]:
```
F.feature_name
```

Out[6]:
```
0                                id_id_exm
1                                id_id_anm
2                           id_id_lev_dist
3                            id_id_lev_sim
4                Name_Name_jac_qgm_3_qgm_3
5            Name_Name_cos_dlm_dc0_dlm_dc0
6            Name_Name_jac_dlm_dc0_dlm_dc0
7                            Name_Name_mel
8                       Name_Name_lev_dist
9                        Name_Name_lev_sim
10                           Name_Name_nmw
11                            Name_Name_sw
12          Country_Country_jac_qgm_3_qgm_3
13      Country_Country_cos_dlm_dc0_dlm_dc0
14      Country_Country_jac_dlm_dc0_dlm_dc0
15                     Country_Country_mel
16                Country_Country_lev_dist
17                 Country_Country_lev_sim
18                     Country_Country_nmw
19                      Country_Country_sw
20        Industry_Industry_jac_qgm_3_qgm_3
21    Industry_Industry_cos_dlm_dc0_dlm_dc0
22    Industry_Industry_jac_dlm_dc0_dlm_dc0
23                   Industry_Industry_mel
24              Industry_Industry_lev_dist
25               Industry_Industry_lev_sim
26                   Industry_Industry_nmw
27                    Industry_Industry_sw
Name: feature_name, dtype: object
```

3.3 Extracting feature vectors

In [7]:
```
# Convert the I into a set of feature vectors using F
H = em.extract_feature_vecs(I,
                            feature_table=F,
                            attrs_after='is_match',
                            show_progress=False)
# Display first few rows
H.head()
```

Out[7]:

	_id	ltable_id	rtable_id	id_id_exm	id_id_anm	id_id_lev_dist	id_id_lev_sim	Na
29	4438	1907	564	0	0.295752	4	0.00	0.1
146	23473	1881	2550	0	0.737647	4	0.00	0.1
16	2770	870	302	0	0.347126	3	0.00	0.3
56	11077	1298	1014	0	0.781202	3	0.25	0.2
75	13987	1366	1467	0	0.931152	2	0.50	0.5

5 rows × 32 columns

In [8]:
```
# Check if the feature vectors contain missing values
# A return value of True means that there are missing values
any(pd.notnull(H))
```

Out[8]: True

We observe that the extracted feature vectors contain missing values. We have to impute the missing values for the learning-based matchers to fit the model correctly.

In [9]:
```
# Impute feature vectors with the mean of the column values.
H = em.impute_table(H,
                exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_ma
tch'],
                strategy='mean')
```

3.4 Selecting the best matcher using cross-validation

First, we need to create a set of learning-based matchers. The following matchers are supported in Magellan: (1) decision tree, (2) random forest, (3) naive bayes, (4) svm, (5) logistic regression, and (6) linear regression.

In [10]:
```
# Create a set of ML-matchers
dt = em.DTMatcher(name='DecisionTree', random_state=seed)
svm = em.SVMMatcher(name='SVM', random_state=seed)
rf = em.RFMatcher(name='RF', random_state=seed)
nb = em.NBMatcher(name='NB')
lg = em.LogRegMatcher(name='LogReg', random_state=seed)
ln = em.LinRegMatcher(name='LinReg')
matchers = [dt, rf, svm, nb, ln, lg]
```

Now, we select the best matcher using k-fold cross-validation. Here we use 5-fold cross validation and use precision, recall, and F-1 metric to select the best matcher.

In [11]:
```
# Performing CV using precision
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='precision', random_state=seed)
result['cv_stats']
```

Out[11]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.85000
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.94736
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.89473
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [12]:
```
# Performing CV using recall
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='recall', random_state=seed)
result['cv_stats']
```

Out[12]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.77272
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.81818
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.77272
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [13]:
```
# Performing CV using F-1
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='f1', random_state=seed)
result['cv_stats']
```

Out[13]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.80952
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.87804
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.82926
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

We select random forest matcher for its highest precision.

3.5 Debug X (Random Forest)

Split the feature vectors H of the development set I into a training set P and a testing set Q.

In [14]:
```
# Split H into P and Q
PQ = em.split_train_test(H, train_proportion=0.5, random_state=seed)
P = PQ['train']
Q = PQ['test']
```

We use the visual debugger for random forest provided by Magellen.

In [15]:
```
# Debug Random Forest matcher using GUI
em.vis_debug_rf(rf, P, Q,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        target_attr='is_match')
```

First we figured out there are some wrong labels. See example below.

In [19]:
```
S[77:78]
```

Out[19]:

	Unnamed: 0	_id	ltable_id	rtable_id	ltable_Country	ltable_Industry	ltable_Marke
77	14222	14222	2513	1497	United States	Pipelines	3616

After fixing the wrong labels manully we reload all the revised data.

In [20]:
```
# Load the revised labeled data
S = em.read_csv_metadata(data_dir + labeled_filename,
                         key='_id',
                         ltable=A, rtable=B,
                         fk_ltable='ltable_id', fk_rtable='rtable_id',
encoding = "ISO-8859-1")
```

```
Metadata file is not present in the given path; proceeding to read t
he csv file.
```

In [21]:
```
# Split S into I an J
IJ = em.split_train_test(S, train_proportion=0.6, random_state=seed)
I = IJ['train']
J = IJ['test']
print(Counter(I.is_match))
print(Counter(J.is_match))
```

```
Counter({1: 100, 0: 80})
Counter({0: 60, 1: 60})
```

Save revised split I, J into files.

In [22]:
```
I.to_csv(data_dir +"I_revised.csv")
J.to_csv(data_dir +"J_revised.csv")
```

And recalculate feature vectors.

In [23]:
```
# Convert the I into a set of feature vectors using F
H = em.extract_feature_vecs(I,
                            feature_table=F,
                            attrs_after='is_match',
                            show_progress=False)
# Display first few rows
H.head()
```

Out[23]:

	_id	ltable_id	rtable_id	id_id_exm	id_id_anm	id_id_lev_dist	id_id_lev_sim	Nar
29	4438	1907	564	0	0.295752	4	0.00	0.1
146	23473	1881	2550	0	0.737647	4	0.00	0.1
16	2770	870	302	0	0.347126	3	0.00	0.3
56	11077	1298	1014	0	0.781202	3	0.25	0.2
75	13987	1366	1467	0	0.931152	2	0.50	0.5

5 rows × 32 columns

In [24]:
```
# Impute feature vectors with the mean of the column values.
H = em.impute_table(H,
                exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_ma
tch'],
                strategy='mean')
```

3.6 Selecting the best matcher using cross-validation - Round 2

In [25]:
```
# Performing CV using precision
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='precision', random_state=seed)
result['cv_stats']
```

Out[25]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.85000
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.94736
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.89473
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [26]:
```
# Performing CV using recall
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='recall', random_state=seed)
result['cv_stats']
```

Out[26]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.77272
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.81818
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.77272
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [27]:
```
# Performing CV using F-1
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='f1', random_state=seed)
result['cv_stats']
```

Out[27]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.80952
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.87804
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.77272
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.82926
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

3.7 Debug X (Random Forest) - Round 2

Since the precision of X is already above 90%. we focus on false negatives in order to improve recall.

In [28]:
```
# Split H into P and Q
PQ = em.split_train_test(H, train_proportion=0.5, random_state=seed)
P = PQ['train']
Q = PQ['test']
```

We use the visual debugger for random forest provided by Magellen.

In [29]:
```
# Debug Random Forest matcher using GUI
em.vis_debug_rf(rf, P, Q,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        target_attr='is_match')
```

- Add new feature

The "MarketValue" features are similar for same companies in the two tables. Thus, We add the feature to calculate the ratio of "MarketValue" for the instances from two tables. Same company should have similar market value in the two tables, aka. the ratio should be close to 1. We use 0 if either value from the two table is missing.

In [30]:
```
# Add some new feature to F
def MarketValue_ratio(ltuple, rtuple) :
    try :
        return float(ltuple.MarketValue) / float(rtuple.MarketValue)
    except ValueError :
        return 0
em.add_blackbox_feature(F, 'MarketValue_ratio', MarketValue_ratio)
```

Out[30]: True

Recalculate feature vectors.

In [31]:
```
# Convert I into feature vectors using updated F
H = em.extract_feature_vecs(I,
                            feature_table=F,
                            attrs_after='is_match',
                            show_progress=False)
# Impute feature vectors with the mean of the column values.
H = em.impute_table(H,
                exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_ma
tch'],
                strategy='mean')
```

Evaluate X again with the new feature.

In [32]:
```
# Check whether the updated precison improves X
result = em.select_matcher([rf], table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='precision', random_state=0)
result['cv_stats']
```

Out[32]:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fold 3
0	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.869565	1.0	0.8947

In [33]:
```
# Check whether the updated recall improves X
result = em.select_matcher([rf], table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='recall', random_state=0)
result['cv_stats']
```

Out[33]:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fo
0	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.909091	0.954545	0.9

In [34]:
```
# Check whether the updated f1 improves X
result = em.select_matcher([rf], table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='f1', random_state=0)
result['cv_stats']
```

Out[34]:

	Name	Matcher	Num folds	Fold 1	Fold 2	Fo
0	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.888889	0.976744	0.9

We observe our recall improved to 91.35% and the precision remains to stay high at 92.05%. The final F1 score is 0.9163. Thus, we decided to move on to the next iteration.

3.7 Selecting the best matcher using cross-validation - Round 3

In [35]:
```
# Performing CV using precision
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='precision', random_state=seed)
result['cv_stats']
```

Out[35]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.86956
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.86956
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.74074
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.85000
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [36]:
```
# Performing CV using recall
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='recall', random_state=seed)
result['cv_stats']
```

Out[36]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.90909
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.90909
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.90909
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.77272
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

In [37]:
```
# Performing CV using F-1
result = em.select_matcher(matchers, table=H,
        exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match'],
        k=5,
        target_attr='is_match', metric='f1', random_state=seed)
result['cv_stats']
```

Out[37]:

	Name	Matcher	Num folds	Fold 1
0	DecisionTree	<py_entitymatching.matcher.dtmatcher.DTMatcher object at 0x11322fac8>	5	0.88888
1	RF	<py_entitymatching.matcher.rfmatcher.RFMatcher object at 0x11322fb00>	5	0.88888
2	SVM	<py_entitymatching.matcher.svmmatcher.SVMMatcher object at 0x11322fba8>	5	0.81632
3	NB	<py_entitymatching.matcher.nbmatcher.NBMatcher object at 0x11322fa58>	5	0.80952
4	LinReg	<py_entitymatching.matcher.linregmatcher.LinRegMatcher object at 0x11322fc50>	5	0.86363
5	LogReg	<py_entitymatching.matcher.logregmatcher.LogRegMatcher object at 0x11322fbe0>	5	0.90909

Since we already have the best recall > 90% (91.35%, Random Forest) for those matchers whose precision > 90%, we decide to stop here and evaluate over the testing set J.

Testing over J

In [38]:
```
# Instantiate matchers to evaluate.
matchers = [
    em.DTMatcher(name='DecisionTree', random_state=seed),
    em.SVMMatcher(name='SVM', random_state=seed),
    em.RFMatcher(name='RF', random_state=seed),
    em.NBMatcher(name='NB'),
    em.LogRegMatcher(name='LogReg', random_state=seed),
    em.LinRegMatcher(name='LinReg'),
]
```

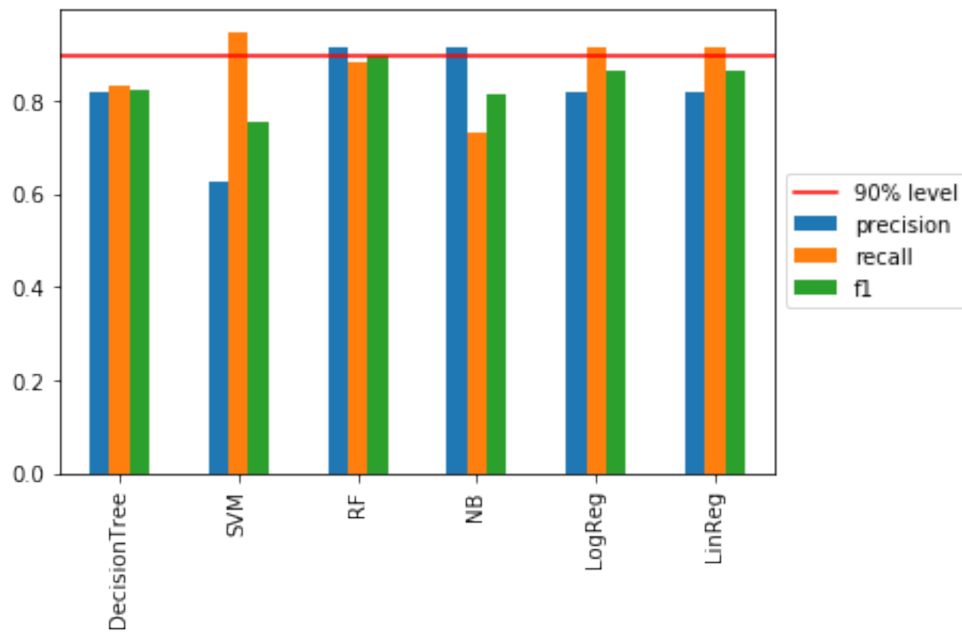
In [39]:
```
eval_results = []
for m in matchers :
    print('#', m.name)

    # Train using feature vectors from I
    m.fit(table=H,
           exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'is_match']
,
           target_attr='is_match')

    # Convert J into a set of feature vectors using F
    L = em.extract_feature_vecs(J, feature_table=F,
                                attrs_after='is_match', show_progress=
False)

    # Impute feature vectors with the mean of the column values.
    L = em.impute_table(L,
                    exclude_attrs=['_id', 'ltable_id', 'rtable_id', 'i
s_match'],
                    strategy='mean')

    # Predict on L
    predictions = m.predict(table=L, exclude_attrs=['_id', 'ltable_id'
, 'rtable_id', 'is_match'],
                  append=True, target_attr='predicted', inplace=False)
    # Evaluate the predictions
    eval_result = em.eval_matches(predictions, 'is_match', 'predicted'
)
    eval_results.append(eval_result)
    em.print_eval_summary(eval_result)
```

```
# DecisionTree
Precision : 81.97% (50/61)
Recall : 83.33% (50/60)
F1 : 82.64%
False positives : 11 (out of 61 positive predictions)
False negatives : 10 (out of 59 negative predictions)
# SVM
Precision : 62.64% (57/91)
Recall : 95.0% (57/60)
F1 : 75.5%
False positives : 34 (out of 91 positive predictions)
False negatives : 3 (out of 29 negative predictions)
# RF
Precision : 91.38% (53/58)
Recall : 88.33% (53/60)
F1 : 89.83%
False positives : 5 (out of 58 positive predictions)
False negatives : 7 (out of 62 negative predictions)
# NB
Precision : 91.67% (44/48)
Recall : 73.33% (44/60)
F1 : 81.48%
False positives : 4 (out of 48 positive predictions)
False negatives : 16 (out of 72 negative predictions)
# LogReg
Precision : 82.09% (55/67)
Recall : 91.67% (55/60)
F1 : 86.61%
False positives : 12 (out of 67 positive predictions)
False negatives : 5 (out of 53 negative predictions)
# LinReg
Precision : 82.09% (55/67)
Recall : 91.67% (55/60)
F1 : 86.61%
False positives : 12 (out of 67 positive predictions)
False negatives : 5 (out of 53 negative predictions)
```

In [40]:

```
df = pd.DataFrame(eval_results, index=[m.name for m in matchers])
```

In [41]:
```
df[['precision', 'recall', 'f1']].plot(kind='bar');
plt.plot(list(range(-1,7)),[0.9]*8, color = "r",label = "90% level");
plt.legend(loc="center left", bbox_to_anchor=(1, 0.5), ncol=1, fancybo
x=1);
```



As shown in above figure, the best matcher is random forest matcher with precision 91.38%, recall 88.33% and F1 89.93%.

4. Summary and discussion

4.1 Time estimates

This only shows the time spent on the following steps explicitly. This does not count the time used for learning Magellan, consolidating ideas, writing up reports and summary, which also takes considerably long time.

4.1.1 Blocking step

In the blokcing step, we have used three different blokcing techniques (overlap, EA, hashing). At first we used overlap to test out how many pairs can be removed. As we kept testing the detailed setup and the additional rules (common stop words and the common starting word dictionary), the blocking step used about 6 hours to achieve the final acceptable blocking power.

4.1.2 Labeling step

In the labeling step we have different iterations to label small sample sets to help examine the blocking results. Along the labeling step we also added rules to the blocking functions.

1. We used about 20 minutes to label the 50-sample dataset
2. We used 1 hour to add additional rules to the blocker and refine the blocking results.
3. We used 3 hours to label the 300-sample G dataset, including checking different company names and websites online to make sure they are the same or different.

The total time used is about 6 hours.

4.1.3 Matching step

In the macthing step we have initially run the 6 matchers with cross-validation. Then after two iterations of debugging we finaly selected random forest as the best matcher.

The time used in this step is about 6 hours, which includes coding, debugging, trails and final analysis.

4.2 Matching results discussion

Here we discuss on why you didn't reach higher recall, and what you can do in the future to obtain higher recall.

4.2.1 Under-used "Industry" field

Although "Industry" field appears in both Forbes and NASDAQ tables, the possible values they use are quite different. "Industry" field values in the two table may have no word in common even for the same company. For example, 3M is "Conglomerates" in Forbes but "Medical/Dental Instruments" in NASDAQ. This makes it very hard to directly utilize the field to help identify matching companies.

In the future, we may investigate into the categorization of the field in the two tables in order to find intrinsic correlations, maybe some sophisticated translation table, that can help distinguish companies from different or possibly same industries.

4.2.2 Human errors

During the matching and debugging process, we found multiple labels themselved are marked wrong, attributed to the teammate who labeled all those pairs of companies. We cannot blame him for the considerable amount tiredness and boredness raised from the exhaustive labor work as all of us could possible imagine, not to mention the tight time limit.

In the future, we shall be more carful about and double check labeled data, and shall also be ready to make corrections to new mistakes found.

4.3 Comments on Magellan

4.3.1 Bug reports

- A typo (missing `","`) at `In[6]` in the overlap blocker Jupyter notebook (https://nbviewer.jupyter.org/github/anhaidgroup/py_entitymatching/blob/rel_0.1.x/notebooks/guides/step In%20Blockers%20%28Overlap%20Blocker%29.ipynb) in the stepwise guide (http://anhaidgroup.github.io/py_entitymatching/v0.1.x/singlepage.html#stepwise-guides).

4.3.2 Features/capabilities that we would really like to see being added

- Cross validation (`select_matcher`) to return multiple metrics at the same time. Now we need to call `select_matcher` three times in order to access precision, recall and F1, which can actually be down within the same run.

4.3.3 What else would we like to see in Magellan

- A detailed explanation on how to use existing facilities to build now feature functions (`get_feature_fn`). Although there are some examples in guilds and API documentations, the demostrated usage are still limited. I realized later by myself that we might be able to use functions/tokenizers provided in `get_sim_funs_for_matching()` and `get_tokenizers_for_matching()`. It may be very helpful at first place some documentation on how to write a feature declaratively as

  ```
  'jaccard(qgm_3(ltuple.address + ltuple.zipcode), qgm_3(rtuple.addre
  ss + rtuple.zipcode)'
  ```

  in API Reference (http://anhaidgroup.github.io/py_entitymatching/v0.1.x/singlepage.html#ways-to-edit-the-manual-feature-generation-process).