Looking at reactions per-turn

We are analyzing the reactions linked to the debate transcript corpus on a turn-by-turn basis.

In [72]:
```
import pandas as pd
import reactions
import nltk
import random
```

Load the table.

In [2]:
```
%time t = reactions.link_reactions_to_transcript('data/reactions_oct3_4project.csv','corpora/oct3_code
t
```

```
CPU times: user 8.72 s, sys: 0.61 s, total: 9.33 s
Wall time: 9.46 s
```

Out[2]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 189015 entries, 0 to 191634
Data columns:
Frame             189015  non-null values
QuestionTopic     189015  non-null values
Reaction_what     189015  non-null values
Reaction_who      189015  non-null values
Speaker           189015  non-null values
Sync'd end        189015  non-null values
Sync'd start      189015  non-null values
Time              189015  non-null values
Tone              189015  non-null values
Topic             189015  non-null values
Transcript        189015  non-null values
UserID            189015  non-null values
start             189015  non-null values
statement         189015  non-null values
turn              189015  non-null values
dtypes: float64(6), int64(1), object(8)
```

In [3]:
```
t[['turn','Speaker','Transcript','start','Reaction_what','Reaction_who']].head(2)
```

Out[3]:

	turn	Speaker	Transcript	start	Reaction_what	Reaction_who
0	1	0	Good evening from the Magness Arena at the Uni...	01:02:01	Agree	Moderator
56861	1	0	Good evening from the Magness Arena at the Uni...	01:02:01.401000	Disagree	Moderator

In [4]:
```
t[['turn','Speaker','Transcript','start','Reaction_what','Reaction_who']].tail(2)
```

Out[4]:

	turn	Speaker	Transcript	start	Reaction_what	Reaction_who
68397	190	0	Thank you, and good night.	02:32:59.726000	Disagree	Romney
191634	190	0	Thank you, and good night.	02:32:59.840000	Agree	Romney

In [5]:
```
print t[:1]
```

```
   Frame  QuestionTopic Reaction_what Reaction_who  Speaker Sync'd end Sync'd start  \
0      9             99         Agree    Moderator        0    1:02:06      1:02:01

                      Time  Tone  Topic  \
0  2012-10-04 01:02:00.967000     0      9
```

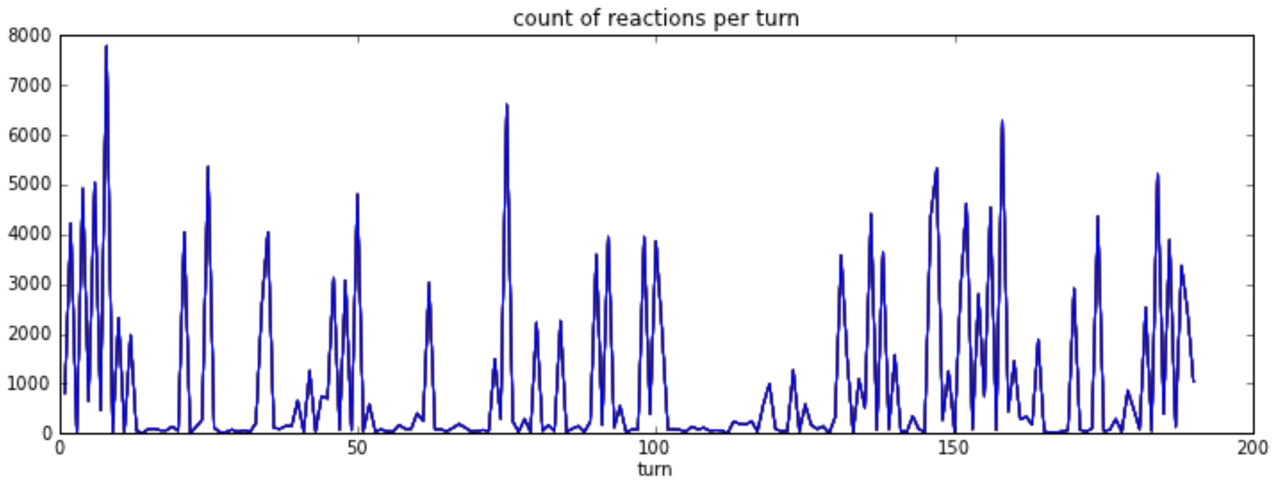
```
                                          Transcript  \
0  Good evening from the Magness Arena at the Uni...

                                              UserID     start  statement  turn
0  ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2YzNTQxZW...  01:02:01          0     1
```

Number of reactions for each turn

In [6]:
```
t.groupby('turn').count().plot(legend=False, figsize=(12, 4), title='count of reactions per turn')
```

Out[6]: <matplotlib.axes.AxesSubplot at 0x7b99d90>

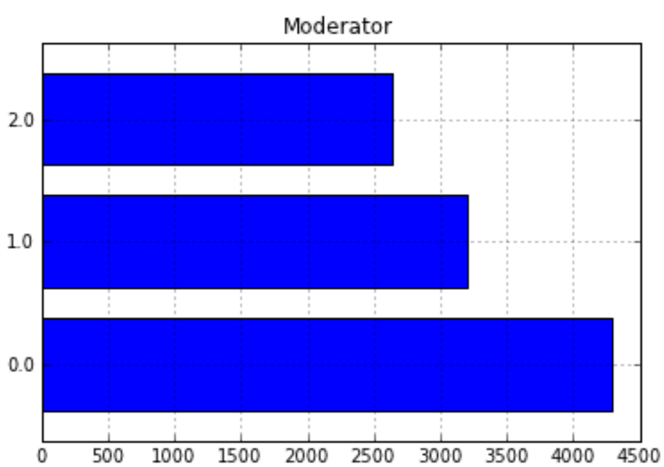


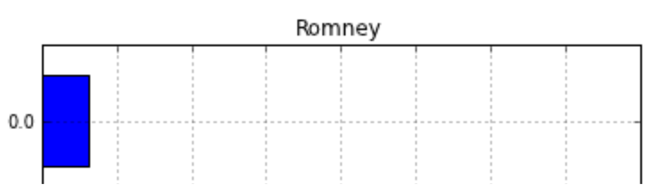
Looking at all reactions for each speaker

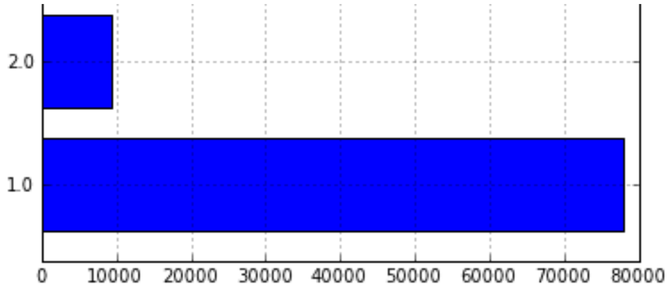
How does Speaker map to Reaction_who?

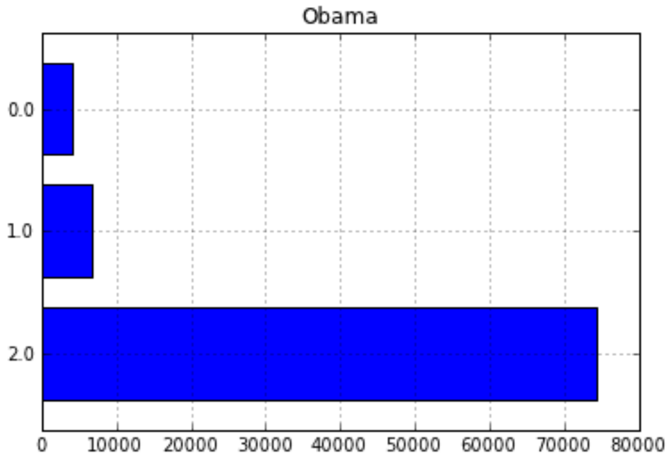
0 = Moderator **1** = Romney **2** = Obama

In [7]:
```
for s in ['Moderator','Romney','Obama']:
    t[t.Reaction_who == s].Speaker.value_counts().plot(title=s, kind='barh')
    show()
```





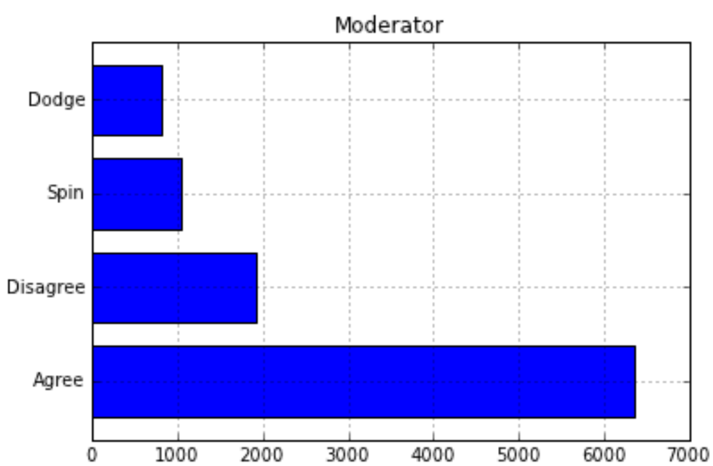


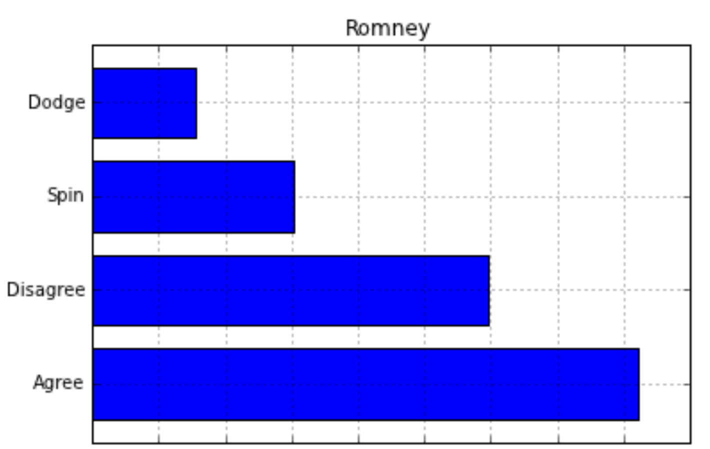


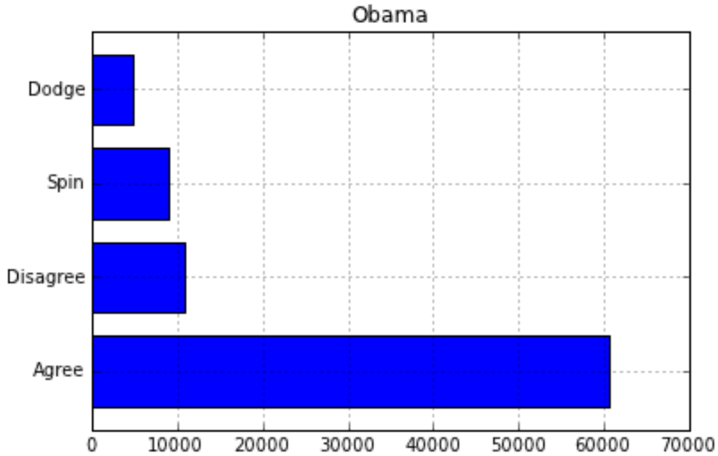
It is interesting that a lot of the time people are reacting to people who are not speaking. Why is this? This is especially true when the moderator is speaking, but perhaps that is not unexpected.

Now let's look at the the reaction data alone to see how people feel about each candidate.

In [8]:
```
for s in ['Moderator','Romney','Obama']:
    t[t.Reaction_who == s].Reaction_what.value_counts().plot(title=s, kind='barh')
    show()
```

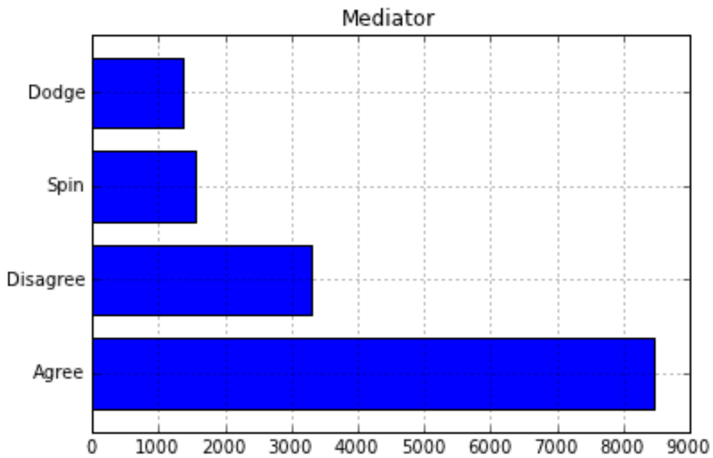


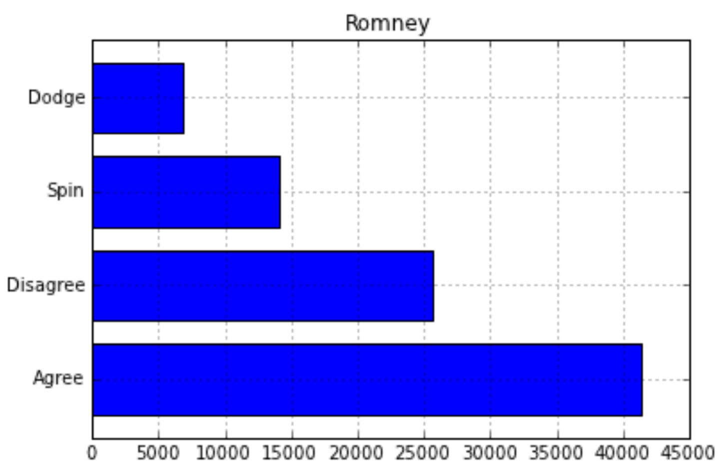


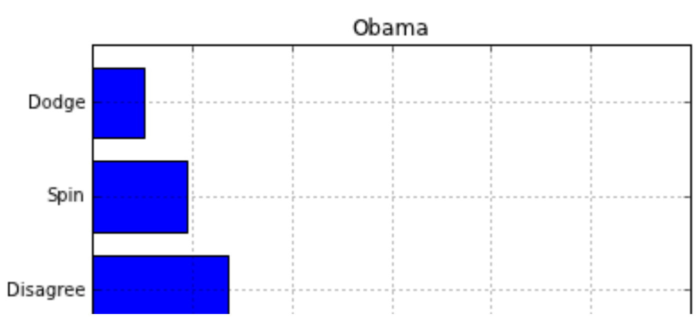


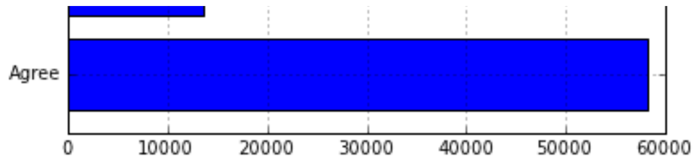
We can also look at the reactions for each candidate based on who the **transcript** says is speaking. The only difference that is obvious here is that the moderator is getting flack for some negative reactions for the candidates.

In [9]:
```
for s,n in [(0,'Mediator'),(1,'Romney'),(2,'Obama')]:
    t[t.Speaker == s].Reaction_what.value_counts().plot(title=n, kind='barh')
    show()
```









Text and reaction counts for each turn

Get all the transcript text for each turn as one string, and the number of reactions for that turn.

In [68]:
```
t2 = t.groupby(['statement']).first()[['Transcript','turn']]
t2.head(2)
```

Out[68]:

	Transcript	turn
statement		
0	Good evening from the Magness Arena at the Uni...	1
1	I'm Jim Lehrer of the PBS NewsHour,	1

In [75]:
```
t3 = pd.DataFrame({'reactions':t.groupby('turn').count().Time, 'text':t2.groupby('turn').apply(lambda
t3.head(2)
```

Out[75]:

	reactions	text
turn		
1	812	Good evening from the Magness Arena at the Uni...
2	4213	Well, thank you very much, Jim, for this oppor...

Text features

In [110]:
```
t3['words'] = t3.text.apply(lambda txt: [t.lower() for t in nltk.tokenize.word_tokenize(txt) if t.isal
```

In [111]:
```
t3['word_count'] = t3.words.apply(lambda words: len(words))
```

In [220]:
```
ranked_unigrams = nltk.FreqDist([w for word_list in t3.words for w in word_list]).keys()
```

In [218]:
```
MAX_FEATURES = 1000
```

In [219]:
```
t3['unigrams'] = t3.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:MAX_FEAT
```

In [169]:
```
t3['unigram_count'] = t3.unigrams.apply(lambda unigrams: len(unigrams))
```

In [177]:
```
t3.head()
```

Out[177]:

	reactions	text	words	word_count	unigrams	unigram_count	label
turn							
1	812	Good evening from the Magness Arena at the	[good, evening, from, the, magness, arena,	259	{'all': True, 'domestic': True,	112	False

		Uni...	at,...		'questions': T...		
2	4213	Well, thank you very much, Jim, for this oppor...	[well, thank, you, very, much, jim, for, this,...	282	{'sector': True, 'all': True, 'code': True, 'j...	138	True
3	27	Governor Romney, two minutes.	[governor, romney, two, minutes]	4	{'minutes': True, 'romney': True, 'two': True,...	4	False
4	4913	Thank you, Jim.It's an honor to be here with y...	[thank, you, an, honor, to, be, here, with, yo...	351	{'restore': True, 'particularly': True, 'help'...	148	True
5	651	Mr. President, please respond directly to what...	[president, please, respond, directly, to, wha...	22	{'respond': True, 'what': True, 'said': True, ...	16	False

What to predict?

In [194]:
```
t3['label'] = t3.reactions >= t3.reactions.quantile(.5)
```

Train and test

In [195]:
```
train_rows = random.sample(t3.index, len(t3)*9/10)
```

In [196]:
```
trn = t3.ix[train_rows]
tst = t3.drop(train_rows)
print len(trn)
print len(tst)
```

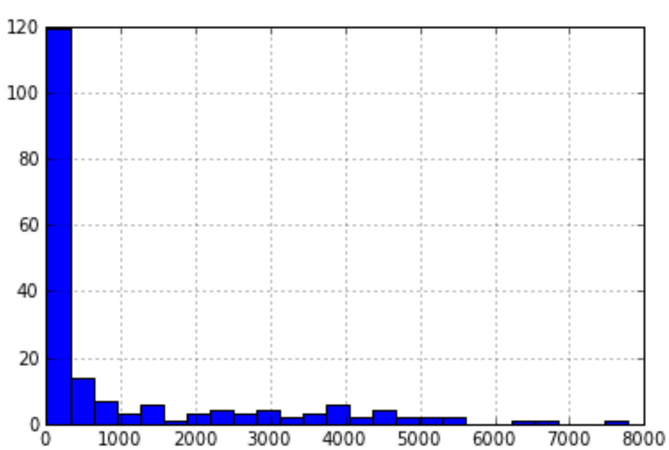
```
171
19
```

In [197]:
```
t3.reactions.describe()
```

Out[197]:
```
count      190.000000
mean       994.815789
std       1622.915791
min          1.000000
25%         53.500000
50%        141.500000
75%       1075.500000
max       7777.000000
```

In [198]:
```
t3.reactions.hist(bins=25)
```

Out[198]:
```
<matplotlib.axes.AxesSubplot at 0x4cd7130>
```



In [199]:
```
%time classifier = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
```

```
CPU times: user 0.07 s, sys: 0.02 s, total: 0.09 s
Wall time: 0.08 s
```

In [200]:
```
nltk.classify.accuracy(classifier, zip(tst.unigrams, tst.label))
```

Out[200]: 0.5263157894736842

Multiple train/test partitions

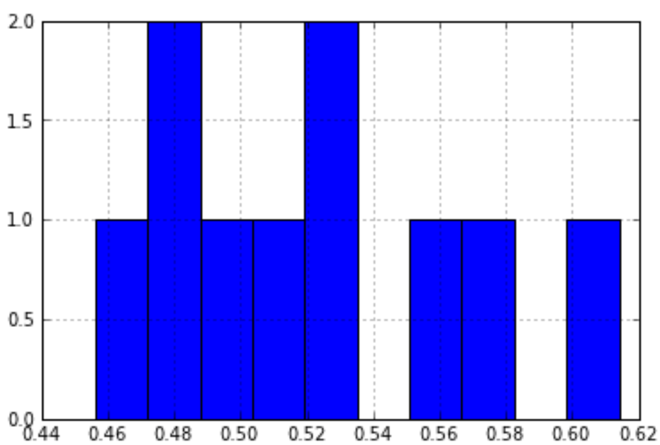
In [213]:
```
accs = []
for i in range(10):
    train_rows = random.sample(t3.index, len(t3)*9/10)
    trn = t3.ix[train_rows]
    tst = t3.drop(train_rows)
    classifier = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
    accs.append(nltk.classify.accuracy(classifier, zip(tst.unigrams, tst.label)))
a2 = pd.DataFrame({'accuracy':accs})
```

In [214]:
```
print a2.describe()
```

```
        accuracy
count  10.000000
mean    0.563158
std     0.126632
min     0.368421
25%     0.486842
50%     0.552632
75%     0.631579
max     0.789474
```

In [212]:
```
a2.accuracy.hist()
```

Out[212]: <matplotlib.axes.AxesSubplot at 0x4e18c50>



Feature hyperparams

Let's see if we can tune the max features hyper parameter (how many of the most frequent unigrams to use as features).

In [216]:
```
t4 = t3.copy()
```

In [232]:
```
len(ranked_unigrams)
```

Out[232]: 1798

In [239]:
```
p = []
trn_means = []
tst_means = []
for max_feats in range(1,len(ranked_unigrams),100):
    t4['unigrams'] = t4.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:max_
    trn_ac = []
    tst_ac = []
    print max_feats,
    for i in range(50):
        print i,
        train_rows = random.sample(t4.index, len(t4)*9/10)
        trn,tst = t4.ix[train_rows],t4.drop(train_rows)
        classifier = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
        trn_ac.append(nltk.classify.accuracy(classifier, zip(trn.unigrams, trn.label)))
        tst_ac.append(nltk.classify.accuracy(classifier, zip(tst.unigrams, tst.label)))
    p.append(max_feats)
    trn_means.append(mean(trn_ac))
    tst_means.append(mean(tst_ac))
    print ''
```

```
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
101 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
201 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
301 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
401 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
501 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
601 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
701 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
801 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
901 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
36 37 38 39 40 41 42 43 44 45 46 47 48 49
1001 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1101 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1201 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1301 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1401 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1501 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1601 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
1701 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34
35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
```

In [240]:
```
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
```

In [241]:
```
results
```

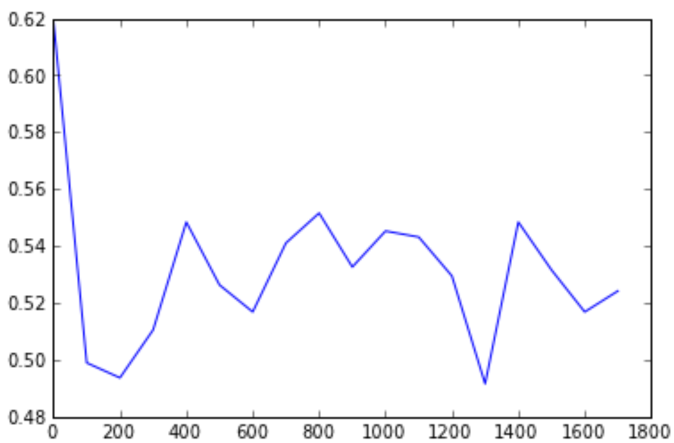
Out[241]:

	max_features	test	train
0	1	0.592632	0.651111
1	101	0.496842	0.527368
2	201	0.531579	0.523392
3	301	0.530526	0.529357

3	301	[illegible]	[illegible]
4	401	0.534737	0.541520
5	501	0.550526	0.543392
6	601	0.542105	0.547018
7	701	0.525263	0.544327
8	801	0.527368	0.543743
9	901	0.522105	0.544444
10	1001	0.507368	0.545263
11	1101	0.526316	0.542807
12	1201	0.534737	0.543626
13	1301	0.513684	0.549357
14	1401	0.553684	0.547135
15	1501	0.547368	0.546199
16	1601	0.521053	0.549123
17	1701	0.551579	0.547836

In [236]:
```
plot(p,means)
```

Out[236]: [<matplotlib.lines.Line2D at 0x5479eb0>]



In [ ]: