Reactions per turn specific to politics of app user

Continuing to other task than predicting total reactions in general.

In [1]:
```
import pandas as pd
import reactions
import nltk
import random
import matplotlib.pyplot as plt
from pandas.tools.plotting import scatter_matrix
from nltk.corpus import stopwords
```

In [2]:
```
%time r = reactions.link_reactions_to_transcript('data/reactions_oct3_4project.csv','corpora/oct3_coded_transcript_sync.csv')
```

```
CPU times: user 8.96 s, sys: 0.56 s, total: 9.52 s
Wall time: 9.53 s
```

In [3]:
```
r2 = r.copy()
#del r2["Sync'd start"]
#del r2["Sync'd end"]
del r2["Time"]
del r2["Speaker"]
r2.head(2)
```

Out[3]:

	Frame	QuestionTopic	Reaction_what	Reaction_who	Sync'd end	Sync'd start	Tone	Topic	Transcript	UserID	start
0	9	99	Agree	Moderator	1:02:06	1:02:01	0	9	Good evening from the Magness Arena at the Uni...	ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2YzNTQxZW...	01:02:01
56861	9	99	Disagree	Moderator	1:02:06	1:02:01	0	9	Good evening from the Magness Arena at the Uni...	ag1zfnJlYWN0bGFicy01ciwLEgRVc2VyIiJhX2U3YmFkZT...	01:02:01

In [4]:
```
r2
```

Out[4]:
```
<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
Sync'd end       189015  non-null values
Sync'd start     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
Speaker_name     189015  non-null values
dtypes: float64(5), int64(1), object(8)
```

Political questionnaire data

In [5]:
```
%time p = reactions.split_reactions_file('data/reactions_oct3_4project.csv')['quest_political']
```

```
CPU times: user 5.14 s, sys: 0.36 s, total: 5.50 s
Wall time: 5.50 s
```

In [6]:
```
p2 = p[['UserID','party_1','political_views_2','candidate_choice_3','confidence_in_choice_4','likely_to_vote_5','candidate_preferred_29']]
p2.head(2)
```

Out[6]:

	UserID	party_1	political_views_2	candidate_choice_3	confidence_in_choice_4	likely_to_vote_5	candidate_pref
0	ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2E0Mjc1MD...	closest to republican party	73	romney	100	100	NaN
62	ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2E0Mzk5OD...	closest to democratic party	20	obama	100	100	NaN

In [7]:
```
p2
```

Out[7]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3767 entries, 0 to 193268
Data columns:
UserID                     3767  non-null values
party_1                    3733  non-null values
political_views_2          3733  non-null values
candidate_choice_3         3733  non-null values
confidence_in_choice_4     3733  non-null values
likely_to_vote_5           3733  non-null values
candidate_preferred_29     2118  non-null values
dtypes: float64(4), object(3)
```

There are ~30 users for whom we don't have political preference info, and the the candidate_preferred_29 col was often left blank.

Simplify party membership into R/D/oth

Let's group the users into D/R/other.

In [8]:
```
p2.groupby('party_1').agg('count').UserID
```

Out[8]:
```
party_1
closest to democratic party    1267
closest to republican party     479
independent                     598
lean democrat                   781
lean republican                 527
no answer                        81
Name: UserID
```

In [9]:
```
p2['party'] = p2.party_1.apply(lambda a: {'closest to democratic party':'democrat',
                                          'lean democrat':'democrat',
                                          'lean republican':'republican',
                                          'closest to republican party':'republican'}.get(a,'other'))
p2.groupby('party').agg('count').UserID
```

Out[9]:
```
party
democrat      2048
other          713
republican    1006
Name: UserID
```

Merge political questionnaire with reactions

In [10]:
```
%time r3 = r2.merge(p2[['UserID','party']])
print 'pre-merge:',len(r2),'post-merge:',len(r3)
r3.head(2)
```

```
CPU times: user 0.30 s, sys: 0.02 s, total: 0.32 s
Wall time: 0.32 s
pre-merge: 189015 post-merge: 189015
```

Out[10]:

	Frame	QuestionTopic	Reaction_what	Reaction_who	Sync'd end	Sync'd start	Tone	Topic	Transcript	UserID	start
0	9	99	Agree	Moderator	1:02:06	1:02:01	0	9	Good evening from the Magness Arena at the Uni...	ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2YzNTQxZW...	01:02:01
1	3	5	Dodge	Obama	1:05:36	1:05:32	1	5	Over the last 30 months, we've seen 5 million ...	ag1zfnJlYWN0bGFicy00ciwLEgRVc2VyIiJhX2YzNTQxZW...	01:05:34.890(

Limit to reactions to the speaker of the ***current turn***.

In [11]:
```
r4 = r3[r3.Reaction_who == r3.Speaker_name]
print 'before:',len(r3),'current-speaker-only:',len(r4), 'difference:',len(r4)-len(r3), 1.0*(len(r4)-len(r3))/len(r4),'percent'
```

```
before: 189015 current-speaker-only: 156622 difference: -32393 -0.206822796287 percent
```

Group by turn

Statements

In [12]:

```
st = r4.groupby(['statement']).first()[['Speaker_name','Transcript','turn',"Sync'd start","Sync'd end"]]
st.head(2)
```

Out[12]:

	Speaker_name	Transcript	turn	Sync'd start	Sync'd end
statement					
0	Moderator	Good evening from the Magness Arena at the Uni...	1	1:02:01	1:02:06
1	Moderator	I'm Jim Lehrer of the PBS NewsHour,	1	1:02:06	1:02:09

Turns

In [13]:

```
t = pd.DataFrame({'speaker':st.groupby('turn').first().Speaker_name,
                  'start':st.groupby('turn').first()["Sync'd start"],
                  'end':st.groupby('turn').last()["Sync'd end"],
                  'reactions':r4.groupby('turn').count().Speaker_name,
                  'statements':st.groupby('turn').count().turn,
                  'text':st.groupby('turn').apply(lambda x: ''.join(x.Transcript)),
                  'agree':r4[r4.Reaction_what=='Agree'].groupby('turn').count().turn,
                  'agree_dem':r4[(r4.party=='democrat') & (r4.Reaction_what=='Agree')].groupby('turn').count().turn,
                  'agree_rep':r4[(r4.party=='republican') & (r4.Reaction_what=='Agree')].groupby('turn').count().turn,
                  'disagree':r4[r4.Reaction_what=='Disagree'].groupby('turn').count().turn,
                  'disagree_dem':r4[(r4.party=='democrat') & (r4.Reaction_what=='Disagree')].groupby('turn').count().turn,
                  'disagree_rep':r4[(r4.party=='republican') & (r4.Reaction_what=='Disagree')].groupby('turn').count().turn,
                  'dodge':r4[r4.Reaction_what=='Dodge'].groupby('turn').count().turn,
                  'spin':r4[r4.Reaction_what=='Spin'].groupby('turn').count().turn,
                  })
tmpstart = pd.to_datetime(t.start)
tmpend = pd.to_datetime(t.end)
t['dur'] = (tmpend - tmpstart)
t.duration = 1.0 * t.dur / 1000000000.0
t['words'] = t.text.apply(lambda txt: [tok.lower() for tok in nltk.tokenize.word_tokenize(txt) if tok.isalpha()])
t['word_count'] = t.words.apply(lambda words: len(words))
t['r_per_st'] = 1.0 * t.reactions / t.statements
t['r_per_w'] = 1.0 * t.reactions / t.word_count
t['r_per_sec'] = 1.0 * t.reactions / t.dur
t['sd_per_sec'] = 1.0 * (t.spin + t.dodge) / t.dur
t['a_to_d_dems'] = t.agree_dem / t.disagree_dem
t['a_to_d_reps'] = t.agree_rep / t.disagree_rep

del t['agree']
del t['agree_dem']
del t['agree_rep']
del t['disagree']
del t['disagree_dem']
del t['disagree_rep']
del t['dodge']
del t['spin']
del t['r_per_st']
del t['r_per_w']
del t['start']
del t['end']
```

In [14]:

```
ranked_unigrams = nltk.FreqDist([w for word_list in t.words for w in word_list]).keys()
MAX_FEATURES = 700 # avoid overfitting
t['unigrams'] = t.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:MAX_FEATURES] and not w in stopwords.words('engl
t['unigram_count'] = t.unigrams.apply(lambda unigrams: len(unigrams))
```

In [15]:

```
#t[['start','end','duration','text','speaker']].head()
t.head()
```

Out[15]:

	reactions	speaker	statements	text	dur	words	word_count	r_per_sec	sd_per_sec	a_to_d_dems	a_to_d_reps	unigrams	unigram_c
turn													
1	416	Moderator	20	Good evening from the Magness Arena at the Uni...	162000000000	[good, evening, from, the, magness, arena, at,...	257	2.567901e-09	5.000000e-10	5.555556	2.454545	{'governing': True, 'among': True, 'major': Tr...	57
2	3976	Obama	22	Well, thank you very much, Jim, for this oppor...	106000000000	[well, thank, you, very, much, jim, for, this,...	278	3.750943e-08	9.311321e-09	39.346939	0.802048	{'sector': True, 'two': True, 'reduce': True, ...	82
4	4392	Romney	33	It's an honor to be here with youand I appreci...	127000000000	[it, an, honor, to, be, here, with, youand, i,...	350	3.458268e-08	6.866142e-09	0.671460	51.296296	{'two': True, 'right': True, 'particularly': T...	79
				Mr.		[president.						{'respond':	

5	54	Moderator	2	President, please respond directly to what...	11000000000	please, respond, directly, to, wha...	22	4.909091e-09	1.181818e-09	6.666667	1.000000	True, 'said': True, 'please': True...	6
6	4635	Obama	23	Well, let me talk specifically about what I th...	131000000000	[well, let, me, talk, specifically, about, wha...	366	3.538168e-08	6.816794e-09	23.742574	1.564885	{'colleges': True, 'code': True, 'particularly...	103

Filter

For now, we get rid of the really short turns, which would seem to likely have noise from adjacent turns and the small numbers of words make the math more sketchy.

In [17]:
```
MIN_WORDS = 30
t2 = t[t.word_count >= MIN_WORDS]
print len(t),'->',len(t2)
```

```
181 -> 70
```

What to predict?

Agree - to - disagree ratio

At least one person of each party agrees and disagrees for each turn.

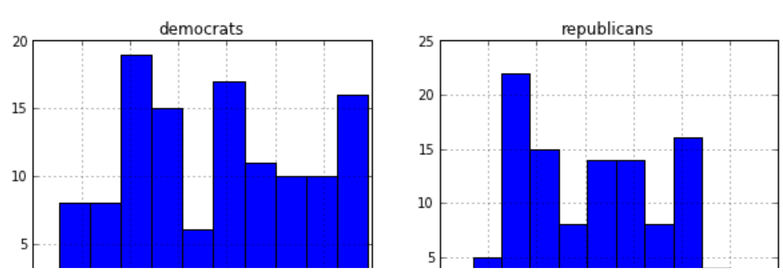
In [17]:
```
t.describe()
```

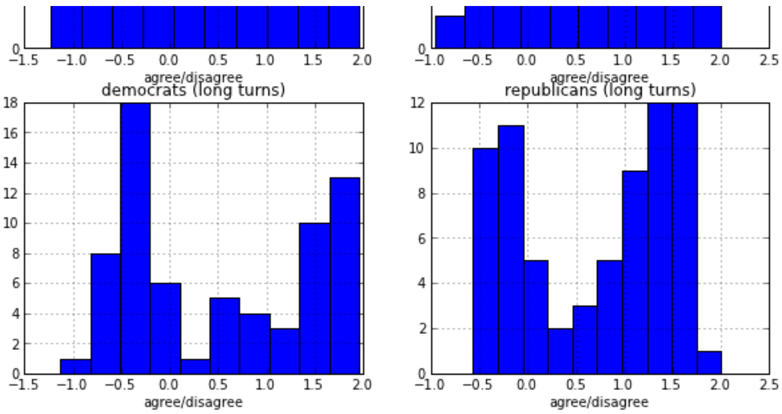
Out[17]:

	reactions	statements	dur	word_count	r_per_sec	sd_per_sec	a_to_d_dems	a_to_d_reps	unigram_count
count	181.000000	181.000000	1.810000e+02	181.000000	1.810000e+02	1.230000e+02	120.000000	109.000000	181.000000
mean	865.314917	6.381215	2.957459e+10	81.044199	inf	5.401644e-09	13.267172	11.007680	21.486188
std	1530.045332	9.050689	1.585391e+08	123.345037	NaN	4.012107e-09	21.156246	16.133355	29.820315
min	1.000000	1.000000	0.000000e+00	1.000000	5.000000e-10	1.304348e-10	0.058824	0.111111	0.000000
25%	11.000000	1.000000	1.000000e+09	5.000000	8.000000e-09	2.000000e-09	0.453767	0.800000	2.000000
50%	50.000000	2.000000	4.000000e+09	15.000000	2.533333e-08	5.000000e-09	2.958333	3.680000	5.000000
75%	1071.000000	8.000000	3.800000e+10	95.000000	3.450000e-08	7.447183e-09	14.361979	15.500000	27.000000
max	7357.000000	54.000000	1.930000e+11	497.000000	inf	1.950000e-08	90.333333	100.500000	112.000000

Most of the time more people seem to be agreeing than disagreeing. Republicans especially so..

In [18]:
```
figsize(10,8)

subplot(221)
log10(t.a_to_d_dems).hist()
xlabel('agree/disagree')
title('democrats')

subplot(222)
log10(t.a_to_d_reps).hist()
xlabel('agree/disagree')
title('republicans')

subplot(223)
log10(t2.a_to_d_dems).hist()
xlabel('agree/disagree')
title('democrats (long turns)')

subplot(224)
log10(t2.a_to_d_reps).hist()
xlabel('agree/disagree')
title('republicans (long turns)')

show()
```





So it seems that taking out reactions from short turns reveals more polarization. Perhaps this is because on short turns there is just more noise from adjacent turns? Or that people become more and more energized in their responses as the speakers continue to talk?

In [19]:
```
PERC = .03
print '{:_^80}'.format('dems agree'.upper())
for v in t2[t2.a_to_d_dems > t2.a_to_d_dems.quantile(1-PERC)].text.values: print v+'\n'
print '{:_^80}'.format('dems disagree'.upper())
for v in t2[t2.a_to_d_dems < t2.a_to_d_dems.quantile(PERC)].text.values: print v+'\n'
```

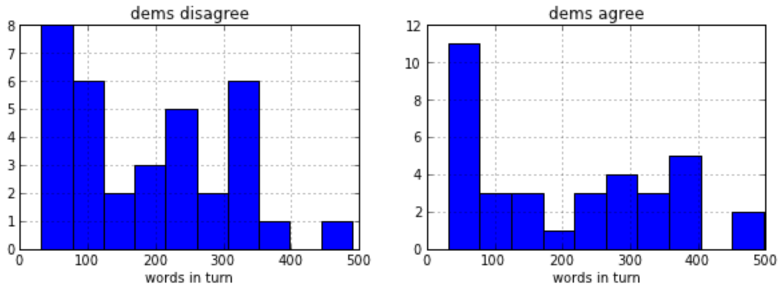
```
___________________________________DEMS AGREE___________________________________
It means that -- Governor Romney talked about Medicaid and how we could send it back to the statesbut effectively this means a 30 percent cut in the primary program we help for seniors who are in nursing homes, for kids who are with disabilities --

Well, four years ago when I was running for office I was traveling around and having those same conversations that Governor Romney talks about.And it wasn't just that small businesses were seeing costs skyrocket and they couldn't get affordable coverage even if they wanted to provide it to their employees; it wasn't just that this was the biggest driver of our federal deficit, our overall health care costs.But it was families who were worried about going bankrupt if they got sick -- millions of families, all across the country.If they had a pre-existing condition they might not be able to get coverage at all.If they did have coverage, insurance companies might impose an arbitrary limit.And so as a consequence, they're paying their premiums, somebody gets really sick, lo and behold they don't have enough money to pay the bills because the insurance companies say that they've hit the limit.So we did work on this alongside working on jobs, because this is part of making sure that middle-class families are secure in this country.And let me tell you exactly what �Obamacare� did.Number one, if you've got health insurance it doesn't mean a government take over.You keep your own insurance, you keep your own doctor.But it does say insurance companies can't jerk you around.They can't impose arbitrary lifetime limits.They have to let you keep your kid on their insurance -- your insurance plan till you're 26 years old.And it also says that they're -- you're going to have to get rebates if insurance companies are spending more on administrative costs and profits than they are on actual care.Number two, if you don't have health insurance, we're essentially setting up a group plan that allows you to benefit from group rates that are typically 18 percent lower than if you're out there trying to get insurance on the individual market.Now, the last point I'd make before --

Let me just point out, first of all, this board that we're talking about can't make decisions about what treatments are given. That's explicitly prohibited in the law.But let's go back to what Governor Romney indicated, that under his plan he would be able to cover people with pre-existing conditions.Well, actually, Governor, that isn't what your plan does. What your plan does is to duplicate what's already the law, which says if you are out of health insurance for three months then you can end up getting continuous coverage and an insurance company can't deny you if it's been under 90 days.But that's already the law. And that doesn't help the millions of people out there with pre-existing conditions. There's a reason why Governor Romney set up the plan that he did in Massachusetts. It wasn't a government takeover of health care. It was the largest expansion of private insurance. But what it does say is that insurers, you've got to take everybody. Now, that also means that you've got more customers.But when Governor Romney says that he'll replace it with something but can't detail how it will be in fact replacedand the reason he set up the system he did in Massachusetts is because there isn't a better way of dealing with the pre-existing conditions problem, it just reminds me of -- you know, he says that he's going to close deductions and loopholes for his tax plan.That's how it's going to be paid for. But we don't know the details. He says that he's going to replace Dodd-Frank, Wall Street reform. But we don't know exactly which ones. He won't tell us. He now says he's going to replace �Obamacare� and assure that all the good things that are in it are going to be in there and you don't have to worry.And at some point, I think the American people have to ask themselves, is the reason that Governor Romney is keeping all these plans to replace secret because they're too good? Is it because that somehow middle-class families are going to benefit too much from them? No, the reason is because when we reform Wall Street, when we tackle the problem of pre-existing conditions, then, you know, these are tough problems, and we've got to make choices. And the choices we've made have been ones that ultimately are benefiting middle-class families all across the country.

_________________________________DEMS DISAGREE__________________________________
Let me -- let me repeat -- let me repeat what I said -- (inaudible).I'm not in favor of a $5 trillion tax cut.That's not my plan.My plan is not to put in place any tax cut that will add to the deficit.That's point one.So you may keep referring to it as a $5 trillion tax cut, but that's not my plan.

First of all, the Department of Energy has said the tax break for oil companies is $2.8 billion a year.And it's actually an accounting treatment, as you know, that's been in place for a hundred years. Now --

And -- and in one year, you provided $90 billion in breaks to the green energy world.Now, I like green energy as well, but that's about 50 years' worth of what oil and gas receives, and you say Exxon and Mobil -- actually, this $2.8 billion goes largely to small companies, to drilling operators and so forth.But you know, if we get that tax rate from 35 percent down to 25 percent, why, that $2.8 billion is on the table, of course it's on the table.That's probably not going to survive, you get that rate down to 25 percent.But -- but don't forget, you put $90 billion -- like 50 years worth of breaks -- into solar and wind, to -- to Solyndra and Fisker and Tesla and Ener1.I mean, I -- I had a friend who said, you don't just pick the winners and losers; you pick the losers.All right? So -- so this is not -- this is not the kind of policy you want to have if you want to get America energy-secure.The second topic, which is you said you get a deduction for getting a plant overseas.Look, I've been in business for 25 years.I have no idea what you're talking about.I maybe need to get a new accountant.
```

How large are the turns where dems either agree or disagree?

In [20]:
```
figsize(10,3)
subplot(121)
t2[t2.a_to_d_dems < t2.a_to_d_dems.quantile(.5)].word_count.hist()
title('dems disagree')
xlabel('words in turn')
subplot(122)
t2[t2.a_to_d_dems >= t2.a_to_d_dems.quantile(.5)].word_count.hist()
```

```
title('dems agree')
xlabel('words in turn')
```

Out[20]: <matplotlib.text.Text at 0x940b590>



The threshold for converting the ratio to a true/false label is ~1.7 dems agreeing to 1 dem disagreeing.

In [21]:
```
t2['label'] = t2.a_to_d_dems >= t2.a_to_d_dems.quantile(.5)
print t2.a_to_d_dems.quantile(.5)
t2.label.describe()
```

```
2.73529411765
```

Out[21]:
```
count           70
mean           0.5
std      0.5036102
min          False
25%              0
50%            0.5
75%              1
max           True
```

Train and test experiment on dems

In [22]:
```
ex = t2
```

In [23]:
```
train_rows = random.sample(ex.index, len(ex)*9/10)
trn = ex.ix[train_rows]
tst = ex.drop(train_rows)
print len(trn)
print len(tst)
```

```
63
7
```

In [24]:
```
%time cl = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
```

```
CPU times: user 0.05 s, sys: 0.01 s, total: 0.06 s
Wall time: 0.06 s
```

In [25]:
```
nltk.classify.accuracy(cl, zip(tst.unigrams, tst.label))
```

Out[25]: 1.0

In [26]:
```
cl.show_most_informative_features(10)
```

```
Most Informative Features
                governor = True             True : False  =      9.9 : 1.0
                   comes = True             True : False  =      5.5 : 1.0
                  system = True             True : False  =      5.5 : 1.0
                   means = True             True : False  =      4.8 : 1.0
                    cuts = True             True : False  =      4.8 : 1.0
                approach = True             True : False  =      4.8 : 1.0
                 problem = True             True : False  =      4.8 : 1.0
                     top = True             True : False  =      4.2 : 1.0
                    made = True             True : False  =      4.2 : 1.0
                governor = None            False : True   =      4.1 : 1.0
```

Whoa! Dems really hate america.. haha

Train and test experiment on reps

The threshold we will use for republicans is higher than the threshold for democrats. This appears to be because more republicans were agreeing with what was said during the debate over all compared to democrats.

In [27]:
```
t2['label2'] = t2.a_to_d_reps >= t2.a_to_d_reps.quantile(.5)
print t2.a_to_d_reps.quantile(.5)
t2.label2.describe()
```

```
8.75
```

Out[27]:
```
count           70
mean           0.5
std      0.5036102
```

```
min          False
25%              0
50%            0.5
75%              1
max           True
```

In [28]:
```
ex = t2
```

In [29]:
```
train_rows2 = random.sample(ex.index, len(ex)*9/10)
trn2 = ex.ix[train_rows2]
tst2 = ex.drop(train_rows2)
print len(trn2)
print len(tst2)
```

```
63
7
```

In [30]:
```
%time cl2 = nltk.NaiveBayesClassifier.train(zip(trn2.unigrams, trn2.label2))
```

```
CPU times: user 0.05 s, sys: 0.01 s, total: 0.06 s
Wall time: 0.05 s
```

In [31]:
```
nltk.classify.accuracy(cl2, zip(tst2.unigrams, tst2.label2))
```

Out[31]:
```
0.8571428571428571
```

In [32]:
```
cl2.show_most_informative_features(10)
```

```
Most Informative Features
                governor = True             False : True   =     17.9 : 1.0
                  romney = True             False : True   =     16.5 : 1.0
                  system = True             False : True   =      6.2 : 1.0
                     top = True             False : True   =      4.8 : 1.0
                    cuts = True             False : True   =      4.8 : 1.0
                   comes = True             False : True   =      4.8 : 1.0
                governor = None              True : False  =      4.6 : 1.0
                   bring = True              True : False  =      4.6 : 1.0
                  making = True             False : True   =      4.2 : 1.0
                    even = True             False : True   =      4.0 : 1.0
```

Really, these train/test sets are **so** small, that (1) we can't draw much information from them without cross validation and (2) we are very prone to overfitting.

Hyperparams grid search on dems

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

In [18]:
```
gr = t2.copy()
```

In [19]:
```
len(ranked_unigrams)
```

Out[19]:
```
1752
```

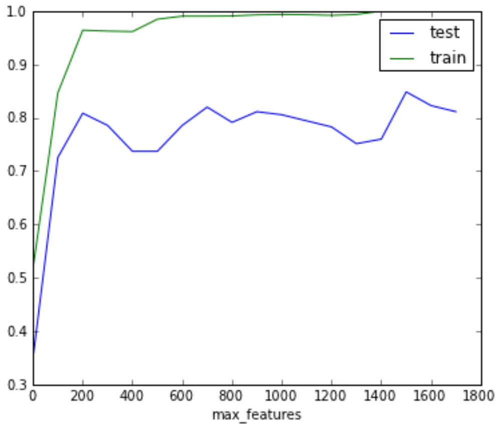
In [24]:
```
t2['label'] = t2.a_to_d_dems >= t2.a_to_d_dems.quantile(.5)
p = []
trn_means = []
tst_means = []
#for max_feats in range(1,len(ranked_unigrams),100):
for max_feats in range(1,700,100):
    gr['unigrams'] = gr.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:max_feats] and not w in stopwords.words('e
    trn_ac = []
    tst_ac = []
    print max_feats,
    for i in range(50):
        print i,
        train_rows = random.sample(gr.index, len(gr)*9/10)
        trn,tst = gr.ix[train_rows],gr.drop(train_rows)
        cl = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
        trn_ac.append(nltk.classify.accuracy(cl, zip(trn.unigrams, trn.label)))
        tst_ac.append(nltk.classify.accuracy(cl, 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

In [36]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
```

Out[36]: <matplotlib.axes.AxesSubplot at 0x93d50b0>



In [37]:
```
results
```

Out[37]:

	max_features	test	train
0	1	0.348571	0.516825
1	101	0.725714	0.846667
2	201	0.808571	0.964127
3	301	0.785714	0.962540
4	401	0.737143	0.961587
5	501	0.737143	0.984762
6	601	0.785714	0.990476
7	701	0.820000	0.990476
8	801	0.791429	0.990794
9	901	0.811429	0.993016
10	1001	0.805714	0.993651
11	1101	0.794286	0.993333
12	1201	0.782857	0.992063
13	1301	0.751429	0.993651
14	1401	0.760000	1.000000
15	1501	0.848571	1.000000
16	1601	0.822857	1.000000
17	1701	0.811429	1.000000

It looks like going past ~200 unigram features is not helpful, and by then we are overfitting on train.

How about narrowing down on a smaller number of max features, and seeing what is happening close to max_features == 0.

In [50]:
```
p = []
trn_means = []
tst_means = []
MAX = 500
for max_feats in range(1,500,10):
    gr['unigrams'] = gr.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:max_feats] and not w in stopwords.words('e
    trn_ac = []
    tst_ac = []
    print max_feats,
    for i in range(50):
        print i,
        train_rows = random.sample(gr.index, len(gr)*9/10)
        trn,tst = gr.ix[train_rows],gr.drop(train_rows)
        cl = nltk.NaiveBayesClassifier.train(zip(trn.unigrams, trn.label))
        trn_ac.append(nltk.classify.accuracy(cl, zip(trn.unigrams, trn.label)))
        tst_ac.append(nltk.classify.accuracy(cl, zip(tst.unigrams, tst.label)))
    p.append(max_feats)
    trn_means.append(mean(trn_ac))
    tst_means.append(mean(tst_ac))
    print ''
```

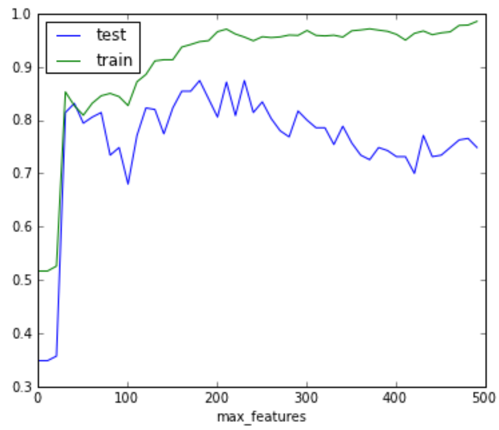
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49
11 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48

49
21 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
31 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
41 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
51 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
61 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
71 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
81 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
91 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
111 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
121 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
131 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
141 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
151 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
161 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
171 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
181 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
191 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
211 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
221 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
231 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
241 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
251 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
261 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
271 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
281 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
291 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
311 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
321 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
331 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
341 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
351 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
361 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
371 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
381 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
391 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
411 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
421 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
431 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
441 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
451 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
461 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
471 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
481 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

```
491 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 [51]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
```

Out[51]: <matplotlib.axes.AxesSubplot at 0x9542ef0>



In [52]:
```
results
```

Out[52]:

	max_features	test	train
0	1	0.348571	0.516825
1	11	0.348571	0.516825
2	21	0.357143	0.526032
3	31	0.814286	0.853016
4	41	0.831429	0.826984
5	51	0.794286	0.809206
6	61	0.805714	0.831746
7	71	0.814286	0.845714
8	81	0.734286	0.850159
9	91	0.748571	0.844127
10	101	0.680000	0.827302
11	111	0.771429	0.872063
12	121	0.822857	0.885714
13	131	0.820000	0.911111
14	141	0.774286	0.913333
15	151	0.822857	0.913333
16	161	0.854286	0.937143
17	171	0.854286	0.941905
18	181	0.874286	0.947302
19	191	0.840000	0.949206
20	201	0.805714	0.966032
21	211	0.871429	0.970794
22	221	0.808571	0.961587
23	231	0.874286	0.955873
24	241	0.814286	0.948889
25	251	0.834286	0.956508
26	261	0.802857	0.954921
27	271	0.780000	0.956190
28	281	0.768571	0.959683
29	291	0.817143	0.959048
30	301	0.800000	0.968254
31	311	0.785714	0.959048
32	321	0.785714	0.958095
33	331	0.754286	0.959365
34	341	0.788571	0.955873
35	351	0.757143	0.967619
36	361	0.734286	0.969206
37	371	0.725714	0.971429
38	381	0.748571	0.968889

In [ ]: