Reactions per turn

Focusing on generating evaluation results for our specific tasks.

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
```

```
CPU times: user 9.87 s, sys: 0.56 s, total: 10.43 s
Wall time: 10.52 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)
```

Political questionnaire data

In [4]:
```
%time p = reactions.split_reactions_file('data/reactions_oct3_4project.csv')['quest_politica
```

```
CPU times: user 4.73 s, sys: 0.34 s, total: 5.07 s
Wall time: 5.07 s
```

In [5]:
```
p2 = p[['UserID','party_1','political_views_2','candidate_choice_3','confidence_in_choice_4'
```

Simplify party membership into R/D/oth

In [6]:
```
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,
p2['candidate'] = p2.candidate_choice_3
#p2['candidate'] = p2.candidate_choice_3.apply(lambda a: {'obama':'democrat',
#                                          'lean democrat':'democrat',
#                                          'lean republican':'republican',
#                                          'closest to republican party':'republican'}.get(a
```

Merge political questionnaire with reactions

In [7]:
```
%time r3 = r2.merge(p2[['UserID','party','candidate']])
```

```
print 'pre-merge:',len(r2),'post-merge:',len(r3)
#r3.head(2)
```

```
CPU times: user 0.63 s, sys: 0.05 s, total: 0.67 s
Wall time: 0.67 s
pre-merge: 189015 post-merge: 189015
```

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

In [86]:
```
r4 = r3[r3.Reaction_who == r3.Speaker_name]
#r4 = r3
print 'before:',len(r3),'current-speaker-only:',len(r4), 'difference:',len(r4)-len(r3), 1.0*
```

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

Group by turn

Statements

In [87]:
```
st = r4.groupby(['statement']).first()[['Speaker_name','Transcript','turn',"Sync'd start","S
```

Turns

In [17]:
```
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,
                  'reactions_oba':r4[(r4.candidate=='obama')].groupby('turn').count().turn,
                  'reactions_rom':r4[(r4.candidate=='romney')].groupby('turn').count().turn,
                  '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')].group
                  'agree_rep':r4[(r4.party=='republican') & (r4.Reaction_what=='Agree')].gro
                  'agree_oba':r4[(r4.candidate=='obama') & (r4.Reaction_what=='Agree')].grou
                  'agree_rom':r4[(r4.candidate=='romney') & (r4.Reaction_what=='Agree')].gro
                  'disagree':r4[r4.Reaction_what=='Disagree'].groupby('turn').count().turn,
                  'disagree_dem':r4[(r4.party=='democrat') & (r4.Reaction_what=='Disagree')]
                  'disagree_rep':r4[(r4.party=='republican') & (r4.Reaction_what=='Disagree'
                  'disagree_oba':r4[(r4.candidate=='obama') & (r4.Reaction_what=='Disagree')
                  'disagree_rom':r4[(r4.candidate=='romney') & (r4.Reaction_what=='Disagree'
                  'dodge_oba':r4[(r4.candidate=='obama') & (r4.Reaction_what=='Dodge')].grou
                  'dodge_rom':r4[(r4.candidate=='romney') & (r4.Reaction_what=='Dodge')].gro
                  'dodge':r4[r4.Reaction_what=='Dodge'].groupby('turn').count().turn,
                  'spin_oba':r4[(r4.candidate=='obama') & (r4.Reaction_what=='Spin')].groupb
                  'spin_rom':r4[(r4.candidate=='romney') & (r4.Reaction_what=='Spin')].group
                  '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(tx
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['rps_oba'] = 1.0 * t.reactions_oba / t.dur
t['rps_rom'] = 1.0 * t.reactions_rom / t.dur
```

```
#t['sd_per_sec'] = 1.0 * (t.spin + t.dodge) / t.dur
t['sdps_oba'] = 1.0 * (t.spin_oba + t.dodge_oba) / t.dur
t['sdps_rom'] = 1.0 * (t.spin_rom + t.dodge_rom) / t.dur
t['a_to_d_dems'] = t.agree_dem / t.disagree_dem
t['a_to_d_reps'] = t.agree_rep / t.disagree_rep
t['a_to_d_oba'] = t.agree_oba / t.disagree_oba
t['a_to_d_rom'] = t.agree_rom / t.disagree_rom

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']
del t['dur']
```

In [33]:
```
all_words = [w for word_list in t.words for w in word_list]
all_bigrams = nltk.bigrams(all_words)
ranked_unigrams = nltk.FreqDist(all_words).keys()
ranked_bigrams = nltk.FreqDist(all_bigrams).keys()
#MAX_FEATURES = 200 # avoid overfitting
MAX_FEATURES = 700 # avoid overfitting
t['unigrams'] = t.words.apply(lambda words: {w:True for w in words if w in ranked_unigrams[:
t['bigrams'] = t.words.apply(lambda words: {"%s_%s"%(w1,w2):True for w1,w2 in nltk.bigrams(w
                                            (w1,w2 in ranked_bigrams[:MAX_FEATURES]) and
                                            (not w1 in stopwords.words('english')) and
                                            (not w2 in stopwords.words('english'))
                                            })
```

Filter

For now, we get rid of the really short turns.

In [88]:
```
MIN_WORDS = 30 # good results
#MIN_WORDS = 20 #
#MIN_WORDS = 0 # this really affects the republican results strongly
t2 = t[t.word_count >= MIN_WORDS]
print len(t),'->',len(t2)
```

```
181 -> 70
```

Crossvalidation code

In [35]:
```
def cv(df, num_folds = 10, classifier=nltk.NaiveBayesClassifier, maxent_params=None, print_f
    df = df.copy()
    df = df.reindex(np.random.permutation(df.index))
    fold_size = len(df)/num_folds
```

```
    fold_starts = range(0, len(df)+fold_size, fold_size)
    folds = zip(fold_starts,fold_starts[1:])
    accs = []
    for (first,last) in folds:
        test_rows = df.index[first:last]
        tst = df.ix[test_rows]
        trn = df.drop(test_rows)
        if maxent_params == None:
            cl = classifier.train(zip(trn.features, trn.label))
        else:
            cl = classifier.train(zip(trn.features, trn.label),
                                  algorithm=maxent_params['algorithm'],
                                  max_iter=maxent_params['max_iter'],
                                  trace=maxent_params['trace'])
        accs.append(nltk.classify.accuracy(cl, zip(tst.features, tst.label)))
        #if print_features:
        #    cl
    #print accs
    return {'mean':mean(accs),'stdev':std(accs)}
```

N-Gram Type

In [66]:
```
t2['features'] = t2.bigrams
```

Task 1: Reactions per second

In [67]:
```
e1 = t2.copy()
e1['label'] = e1.rps_oba >= e1.rps_oba.quantile(.5)
print 'DT', cv(e1, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e1, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e1, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.12453996981544782, 'mean': 0.39999999999999991}
ME {'stdev': 0.13997084244475305, 'mean': 0.59999999999999987}
NB {'stdev': 0.10690449676496977, 'mean': 0.59999999999999998}
```

In [68]:
```
e1 = t2.copy()
e1['label'] = e1.rps_rom >= e1.rps_rom.quantile(.5)
print 'DT', cv(e1, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e1, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e1, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.16225452416572211, 'mean': 0.58571428571428563}
ME {'stdev': 0.18126539343499315, 'mean': 0.75714285714285712}
NB {'stdev': 0.15907898179514349, 'mean': 0.77142857142857135}
```

In [93]:
```
e1['label'] = e1.rps_rom >= e1.rps_rom.quantile(.5)
train_rows = random.sample(e1.index, len(e1)*9/10)
```

```
trn = e1.ix[train_rows]
tst = e1.drop(train_rows)
cl = nltk.NaiveBayesClassifier.train(zip(trn.features, trn.label))
nltk.classify.accuracy(cl, zip(tst.features, tst.label))
cl.show_most_informative_features(25)
```

```
Most Informative Features
                cut_taxes = True             True : False  =      3.6 : 1.0
                  get_rid = True             True : False  =      2.9 : 1.0
          trillion_dollars = True             True : False  =      2.9 : 1.0
            million_people = True             True : False  =      2.5 : 1.0
               health_care = True            False : True   =      2.5 : 1.0
          health_insurance = True            False : True   =      2.4 : 1.0
              running_mate = True            False : True   =      2.4 : 1.0
               get_america = True             True : False  =      2.3 : 1.0
          small_businesses = True             True : False  =      2.1 : 1.0
           governor_romney = True            False : True   =      2.1 : 1.0
               making_sure = True            False : True   =      1.9 : 1.0
                four_years = True             True : False  =      1.7 : 1.0
    traditional_medicare = True            False : True   =      1.7 : 1.0
               wall_street = True            False : True   =      1.7 : 1.0
             balanced_way = True            False : True   =      1.7 : 1.0
                  go_back = True            False : True   =      1.7 : 1.0
                come_back = True            False : True   =      1.7 : 1.0
          unelected_board = True            False : True   =      1.7 : 1.0
          social_security = True            False : True   =      1.7 : 1.0
              romney_says = True            False : True   =      1.7 : 1.0
         training_programs = True            False : True   =      1.7 : 1.0
               whole_bunch = True            False : True   =      1.7 : 1.0
        federal_government = True            False : True   =      1.6 : 1.0
           american_energy = True             True : False  =      1.6 : 1.0
          closing_loopholes = True             True : False  =      1.6 : 1.0
```

Task 2a: Majority agrees with speaker THIS IS TASK 2

Democrats

In [39]:
```
e2ad = t2.copy()
#e2ad['label'] = e2ad.agree_dem >= e2ad.disagree_dem
e2ad['label'] = e2ad.agree_oba >= e2ad.disagree_oba
print 'DT', cv(e2ad, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e2ad, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e2ad, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.15386185163241442, 'mean': 0.74285714285714277}
ME {'stdev': 0.13997084244475305, 'mean': 0.82857142857142851}
NB {'stdev': 0.1743793659390529, 'mean': 0.84285714285714286}
```

In [95]:
```
e2ad['label'] = e2ad.agree_oba >= e2ad.disagree_oba
train_rows = random.sample(e2ad.index, len(e2ad)*9/10)
trn = e2ad.ix[train_rows]
tst = e2ad.drop(train_rows)
cl = nltk.NaiveBayesClassifier.train(zip(trn.features, trn.label))
nltk.classify.accuracy(cl, zip(tst.features, tst.label))
cl.show_most_informative_features(25)
```

```
Most Informative Features
                  romney = True             True : False  =     11.9 : 1.0
```

```
governor = True              True : False  =      7.7 : 1.0
  course = True             False : True   =      5.7 : 1.0
  making = True              True : False  =      5.3 : 1.0
 schools = True             False : True   =      4.8 : 1.0
     rid = True             False : True   =      4.8 : 1.0
   came = True              False : True   =      4.8 : 1.0
  system = True              True : False  =      4.8 : 1.0
    idea = True             False : True   =      4.5 : 1.0
   place = True             False : True   =      4.5 : 1.0
   comes = True              True : False  =      4.3 : 1.0
question = True              True : False  =      4.3 : 1.0
  answer = True             False : True   =      4.0 : 1.0
    hire = True             False : True   =      4.0 : 1.0
    rate = True             False : True   =      4.0 : 1.0
important = True             True : False  =      3.8 : 1.0
 problem = True              True : False  =      3.8 : 1.0
approach = True              True : False  =      3.8 : 1.0
republican = True            True : False  =      3.3 : 1.0
    cuts = True              True : False  =      3.3 : 1.0
families = True              True : False  =      3.3 : 1.0
     get = True             False : True   =      3.2 : 1.0
   first = True              True : False  =      3.2 : 1.0
    ever = True             False : True   =      3.1 : 1.0
 instead = True             False : True   =      3.1 : 1.0
```

Republicans

In [43]:
```
e2ar = t2.copy()
#e2ar['label'] = e2ar.agree_rep >= e2ar.disagree_rep
e2ar['label'] = e2ar.agree_rom >= e2ar.disagree_rom
print 'DT', cv(e2ar, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e2ar, classifier=nltk.classify.MaxentClassifier,
               maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                              'max_iter':25,
                              'trace':0})
print 'NB', cv(e2ar, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.18126539343499315, 'mean': 0.75714285714285723}
ME {'stdev': 0.15971914124998499, 'mean': 0.7857142857142857}
NB {'stdev': 0.1743793659390529, 'mean': 0.44285714285714278}
```

In [94]:
```
e2ar['label'] = e2ar.agree_rom >= e2ar.disagree_rom
train_rows = random.sample(e2ar.index, len(e2ar)*9/10)
trn = e2ar.ix[train_rows]
tst = e2ar.drop(train_rows)
cl = nltk.NaiveBayesClassifier.train(zip(trn.features, trn.label))
nltk.classify.accuracy(cl, zip(tst.features, tst.label))
cl.show_most_informative_features(25)
```

```
Most Informative Features
    idea = True              True : False  =      6.1 : 1.0
   folks = True             False : True   =      5.8 : 1.0
  course = True              True : False  =      5.5 : 1.0
    four = True              True : False  =      5.5 : 1.0
  number = True              True : False  =      5.2 : 1.0
    year = True              True : False  =      4.8 : 1.0
difference = True            False : True   =      4.5 : 1.0
   small = True              True : False  =      4.5 : 1.0
  saying = True              True : False  =      4.2 : 1.0
democrats = True             True : False  =      4.2 : 1.0
  energy = True              True : False  =      4.2 : 1.0
     job = True              True : False  =      4.2 : 1.0
    best = True              True : False  =      4.2 : 1.0
```

```
          worked = True           False : True   =      3.8 : 1.0
        american = True            True : False  =      3.7 : 1.0
       different = True            True : False  =      3.6 : 1.0
            came = True            True : False  =      3.6 : 1.0
            hire = True            True : False  =      3.6 : 1.0
            seen = True            True : False  =      3.6 : 1.0
         million = True            True : False  =      3.5 : 1.0
          better = True            True : False  =      3.3 : 1.0
        spending = True            True : False  =      3.3 : 1.0
         percent = True            True : False  =      3.3 : 1.0
            look = True            True : False  =      3.2 : 1.0
            work = True            True : False  =      3.1 : 1.0
```

Task 2b: Ratio agree-to-disagree above median

Democrats

In [77]:
```
e2bd = t2.copy()
#e2bd['label'] = e2bd.a_to_d_dems >= e2bd.a_to_d_dems.quantile(.5)
e2bd['label'] = e2bd.a_to_d_oba >= e2bd.a_to_d_oba.quantile(.5)
print 'DT', cv(e2bd, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e2bd, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e2bd, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.14568627181693672, 'mean': 0.77142857142857146}
ME {'stdev': 0.18571428571428572, 'mean': 0.87142857142857155}
NB {'stdev': 0.11157499537009505, 'mean': 0.81428571428571428}
```

Republicans

In [22]:
```
e2br = t2.copy()
#e2br['label'] = e2br.a_to_d_reps >= e2br.a_to_d_reps.quantile(.5)
e2br['label'] = e2br.a_to_d_rom >= e2br.a_to_d_rom.quantile(.5)
print 'DT', cv(e2br, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e2br, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e2br, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.24824658035241429, 'mean': 0.73232323232323226}
ME {'stdev': 0.19998979669922559, 'mean': 0.64646464646464641}
NB {'stdev': 0.15907086614165275, 'mean': 0.59595959595959602}
```

Task 3: Spins+dodges per second above median

In [55]:
```
e3 = t2.copy()
e3['label'] = e3.sdps_oba >= e3.sdps_oba.quantile(.5)
print 'DT', cv(e3, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e3, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
```

```
print 'NB', cv(e3, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.15714285714285714, 'mean': 0.75714285714285712}
ME {'stdev': 0.15971914124998496, 'mean': 0.78571428571428559}
NB {'stdev': 0.13997084244475305, 'mean': 0.82857142857142851}
```

In [56]:
```
e3 = t2.copy()
e3['label'] = e3.sdps_rom >= e3.sdps_rom.quantile(.5)
print 'DT', cv(e3, classifier=nltk.classify.DecisionTreeClassifier)
print 'ME', cv(e3, classifier=nltk.classify.MaxentClassifier,
                maxent_params={'algorithm':nltk.classify.MaxentClassifier.ALGORITHMS[0],
                               'max_iter':25,
                               'trace':0})
print 'NB', cv(e3, classifier=nltk.NaiveBayesClassifier)
```

```
DT {'stdev': 0.12453996981544781, 'mean': 0.68571428571428572}
ME {'stdev': 0.12453996981544782, 'mean': 0.59999999999999987}
NB {'stdev': 0.1142857142857143, 'mean': 0.48571428571428565}
```

Hyperparameters

In [89]:
```
gr = t2.copy()
p = []
trn_means = []
tst_means = []
#MAX = 500
#gr['label'] = gr.rps_oba >= gr.rps_oba.quantile(.5) # Task 1 ~300
#gr['label'] = gr.agree_oba >= gr.disagree_oba # Task 2 ~700
gr['label'] = gr.agree_rom >= gr.disagree_rom # Task 2
#gr['label'] = gr.sdps_oba >= gr.sdps_oba.quantile(.5) # Task 3 ~500
#gr['label'] = gr.sdps_rom >= gr.sdps_rom.quantile(.5) # Task 3

#gr['label'] = gr.a_to_d_dems >= gr.a_to_d_dems.quantile(.5)

#for max_feats in range(1,700,100):
for max_feats in range(1,len(ranked_unigrams),100):
    gr['features'] = gr.words.apply(lambda words: {w:True for w in words if w in ranked_unig
    trn_ac = []
    tst_ac = []
    print max_feats,
    for i in range(0,50):
        print i,
        train_rows = random.sample(gr.index, len(gr)*9/10)
        #print train_rows
        trn,tst = gr.ix[train_rows],gr.drop(train_rows)
        cl = nltk.NaiveBayesClassifier.train(zip(trn.features, trn.label))
        trn_ac.append(nltk.classify.accuracy(cl, zip(trn.features, trn.label)))
        tst_ac.append(nltk.classify.accuracy(cl, zip(tst.features, tst.label)))
    p.append(max_feats)
    trn_means.append(mean(trn_ac))
    tst_means.append(mean(tst_ac))
    print ''
```

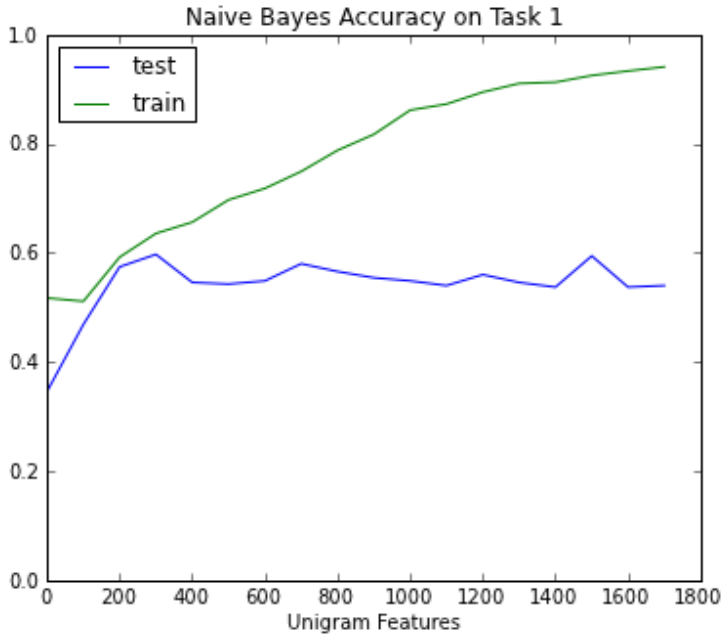
```
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
101 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
201 0 1 2 3 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
```

Task 1

In [48]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
xlabel('Unigram Features')
title('Naive Bayes Accuracy on Task 1')
ylim(0,1)
```

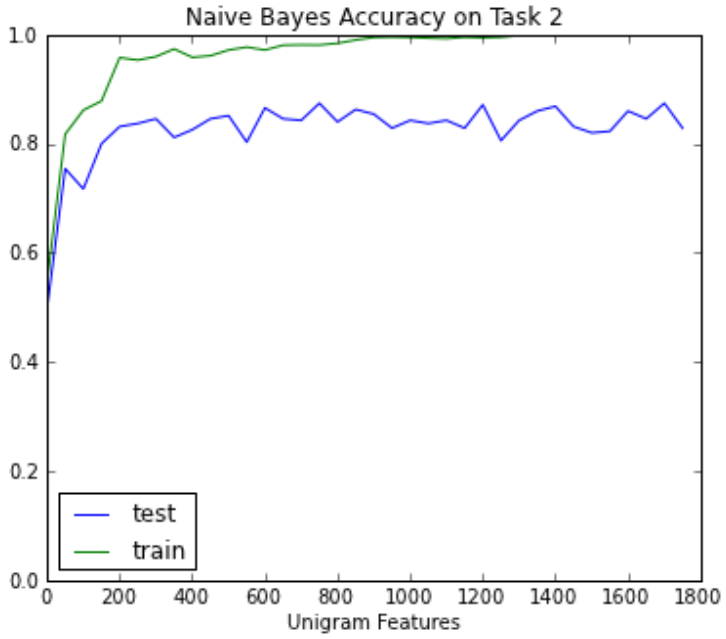
Out[48]: (0, 1)



Task 2 obama

In [31]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
xlabel('Unigram Features')
title('Naive Bayes Accuracy on Task 2')
ylim(0,1)
```

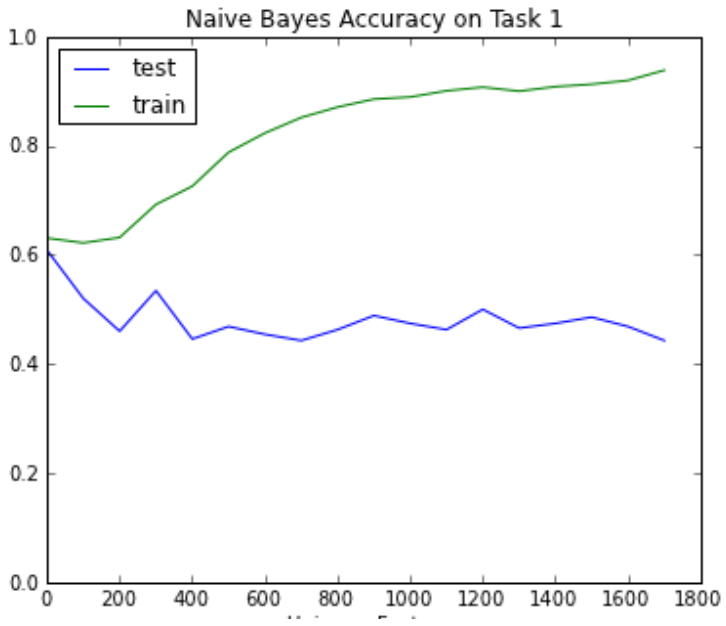
Out[31]: (0, 1)



Task 2 Romney

In [90]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
xlabel('Unigram Features')
title('Naive Bayes Accuracy on Task 1')
ylim(0,1)
```

Out[90]: (0, 1)

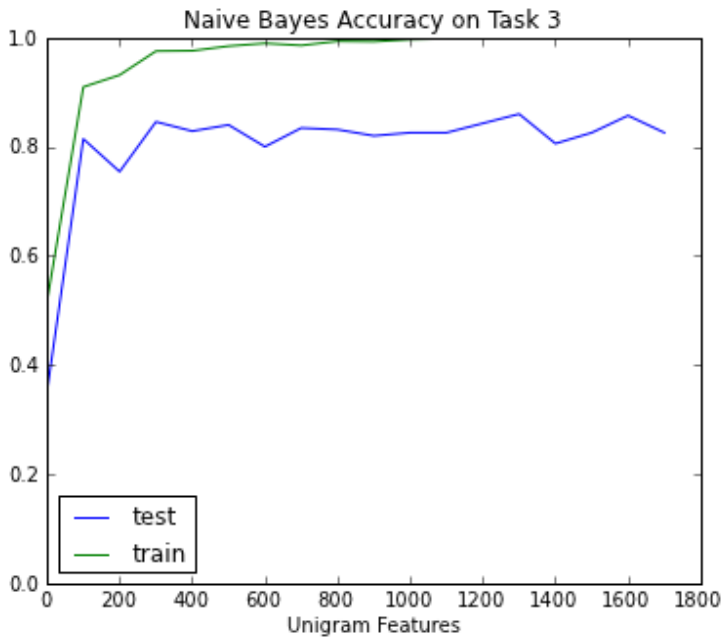


Unigram Features

Task 3

In [53]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
xlabel('Unigram Features')
title('Naive Bayes Accuracy on Task 3')
ylim(0,1)
```

Out[53]: (0, 1)



In [54]:
```
results
```

Out[54]:

	max_features	test	train
0	1	0.351429	0.516508
1	101	0.814286	0.909524
2	201	0.754286	0.931429
3	301	0.845714	0.975238
4	401	0.828571	0.975873
5	501	0.840000	0.984444
6	601	0.800000	0.989524
7	701	0.834286	0.985714
8	801	0.831429	0.993333
9	901	0.820000	0.992698
10	1001	0.825714	0.996508
11	1101	0.825714	0.998095
12	1201	0.842857	0.998730
13	1301	0.860000	0.998095
14	1401	0.805714	0.997778
15	1501	0.825714	0.998095

16	1601	0.857143	0.998095
17	1701	0.825714	0.998095

Smaller range

In [18]:
```
gr = t2.copy()
p = []
trn_means = []
tst_means = []
MAX = 500
gr['label'] = gr.rps_oba >= gr.rps_oba.quantile(.5)
for max_feats in range(1,200,10):
    gr['features'] = gr.words.apply(lambda words: {w:True for w in words if w in ranked_unig
    trn_ac = []
    tst_ac = []
    print max_feats,
    for i in range(0,50):
        print i,
        train_rows = random.sample(gr.index, len(gr)*9/10)
        #print train_rows
        trn,tst = gr.ix[train_rows],gr.drop(train_rows)
        cl = nltk.NaiveBayesClassifier.train(zip(trn.features, trn.label))
        trn_ac.append(nltk.classify.accuracy(cl, zip(trn.features, trn.label)))
        tst_ac.append(nltk.classify.accuracy(cl, zip(tst.features, tst.label)))
    p.append(max_feats)
    trn_means.append(mean(trn_ac))
    tst_means.append(mean(tst_ac))
    print ''
```

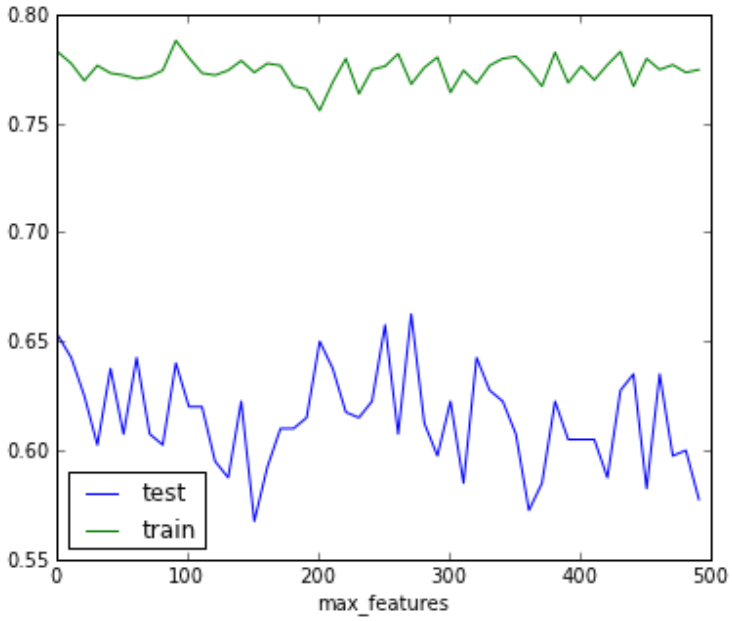
```
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
11 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
21 0 1 2 3 4 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 [19]:
```
figsize(6,5)
results = pd.DataFrame({'max_features':p, 'train':trn_means, 'test':tst_means})
results.plot(x='max_features')
```

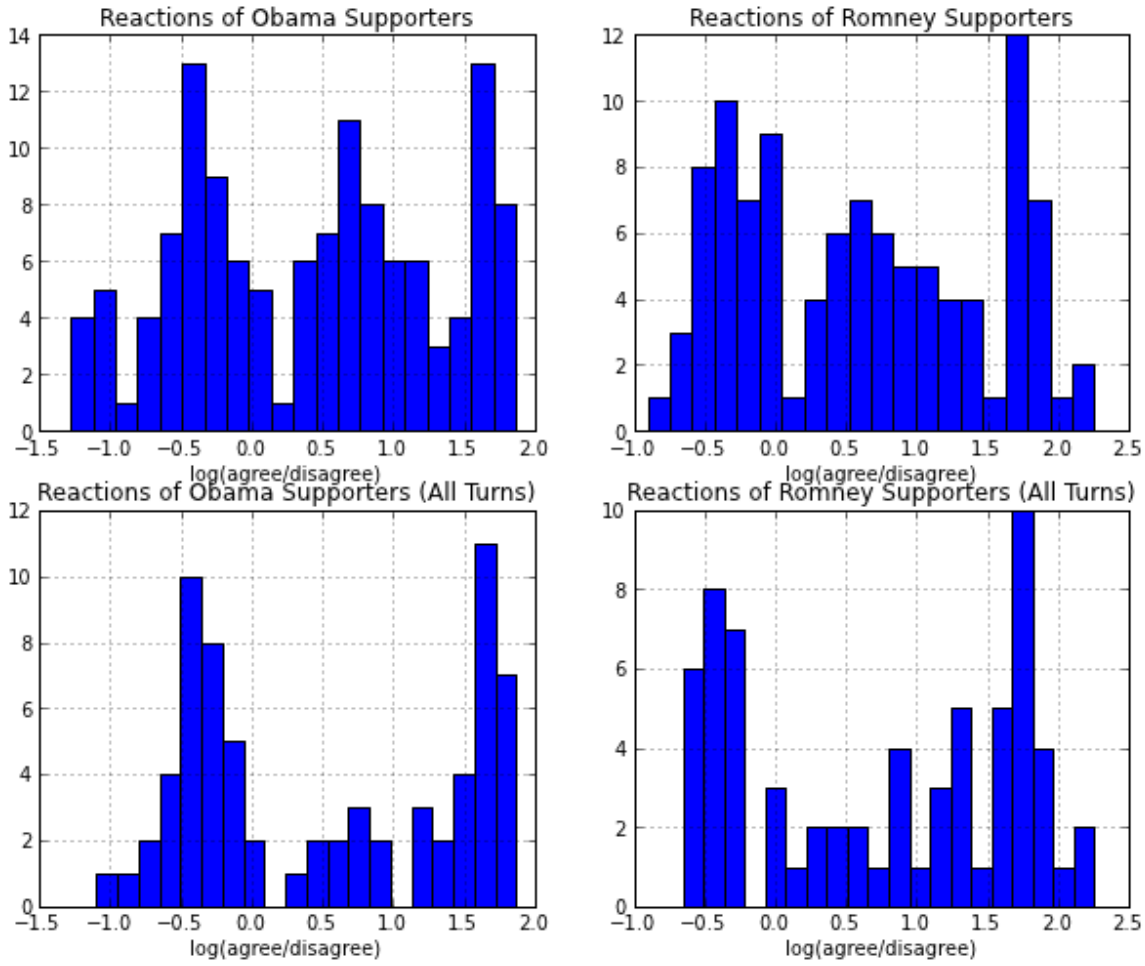
Out[19]: <matplotlib.axes.AxesSubplot at 0x8b171f0>



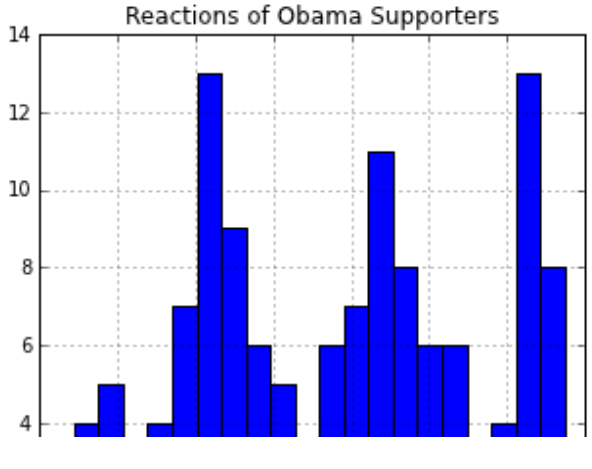
More interpretation

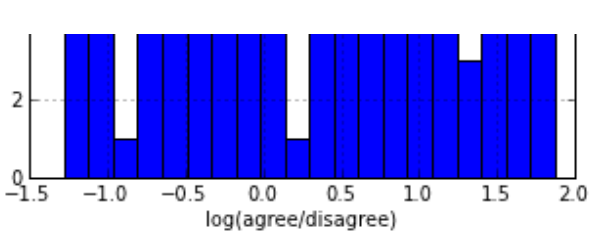
In [80]:
```
figsize(10,8)

BINS=20

subplot(221)
#log10(t.a_to_d_dems).hist()
log10(t.a_to_d_oba).hist(bins=BINS)
xlabel('log(agree/disagree)')
title('Reactions of Obama Supporters')

subplot(222)
#log10(t.a_to_d_reps).hist(bins=30)
log10(t.a_to_d_rom).hist(bins=BINS)
xlabel('log(agree/disagree)')
title('Reactions of Romney Supporters')

subplot(223)
#log10(t2.a_to_d_dems).hist()
log10(t2.a_to_d_oba).hist(bins=BINS)
xlabel('log(agree/disagree)')
title('Reactions of Obama Supporters (All Turns)')

subplot(224)
#log10(t2.a_to_d_reps).hist()
log10(t2.a_to_d_rom).hist(bins=BINS)
xlabel('log(agree/disagree)')
```

```
title('Reactions of Romney Supporters (All Turns)')

show()
```

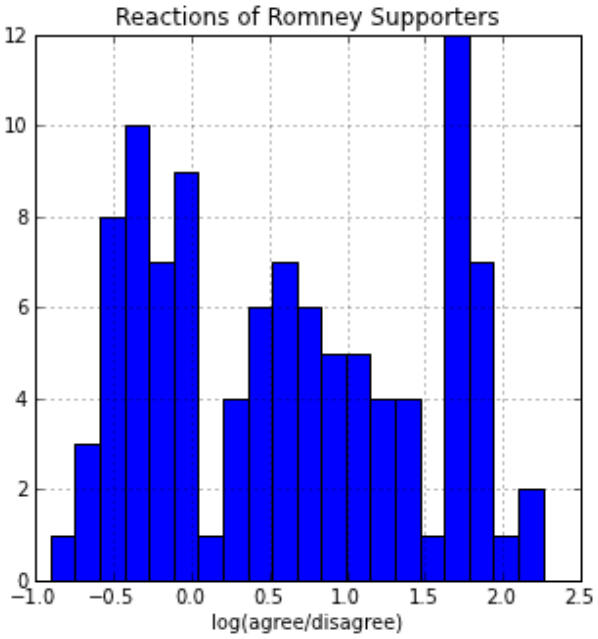


In [84]:
```
figsize(5,5)

BINS=20

#log10(t.a_to_d_dems).hist()
log10(t.a_to_d_oba).hist(bins=BINS)
xlabel('log(agree/disagree)')
title('Reactions of Obama Supporters')


show()
```





In [85]:
```
figsize(5,5)

BINS=20

#log10(t.a_to_d_reps).hist(bins=30)
log10(t.a_to_d_rom).hist(bins=BINS)
xlabel('log(agree/disagree)')
title('Reactions of Romney Supporters')


show()
```



In [ ]: