Andrew Gillock apg2255 Noah Shimizu ncs838 Tanushree Devi Balaji tb33857

In [1]:
```
import pandas as pd
%pylab inline
%matplotlib inline
```

```
%pylab is deprecated, use %matplotlib inline and import the required libraries.
Populating the interactive namespace from numpy and matplotlib
```

Q1. Read in data

In [2]:
```
### Reading in the two tab-separated files without headers and assigning column names
# Read in gold.txt, using tabs as the seperation
gold = pd.read_csv("gold.txt", sep = "\t", names = ["url","category"], index_col = "url"

# Read in labels.txt, using tabs as the seperation
labels = pd.read_csv("labels.txt", sep = "\t",names = ["turk", "url", "category"])
```

In [3]:
```
# Printing Sample Data
gold[:3]
```

Out[3]:

url	category
http://0800-horoscope.com	G
http://18games.net	X
http://1pixelout.net	G

In [4]:
```
# Printing Sample Data
labels[:3]
```

Out[4]:

	turk	url	category
0	A1OT3A29R9N1DG	http://000.cc	P
1	A1PXXEOGQ76RNJ	http://000.cc	G
2	A1PXXEOGQ76RNJ	http://000.cc	G

Q2.Split into two DataFrames

In [5]:
```
# Left joining labels with gold to create a new data frame
labels_join = labels.merge(gold, how = "left", on = "url", suffixes = ["","_true"])

# Filtering for rows present only in labels but not in gold and calling it 'labels_unkno
labels_unknown = labels_join[labels_join["category_true"].isna()].drop("category_true",

# Filtering for rows present in gold and calling it 'labels_on_gold'
labels_on_gold = labels_join.dropna()
```

In [6]:
```
# Sample output for gold labels
labels_on_gold[:5]
```

Out[6]:

	turk	url	category	category_true
245	A1253FXHCZ9CWM	http://0800-horoscope.com	G	G
246	A153PKAL7OAY36	http://0800-horoscope.com	G	G

247	A1FV9SAPL5C6KY	http://0800-horoscope.com	G	G
248	A1JTOT0DWM6QGL	http://0800-horoscope.com	G	G
249	A1PXXEOGQ76RNJ	http://0800-horoscope.com	G	G

In [7]:
```
# Sample output for unknown labels
labels_unknown[:5]
```

Out[7]:

	turk	url	category
0	A1OT3A29R9N1DG	http://000.cc	P
1	A1PXXEOGQ76RNJ	http://000.cc	G
2	A1PXXEOGQ76RNJ	http://000.cc	G
3	A21US576U8SCO4	http://000.cc	G
4	A2LGX47NN7C5D3	http://000.cc	G

Q3.Compute accuracies of turks

In [8]:
```
labels_on_gold["is_correct"] = labels_on_gold["category"] == labels_on_gold["category_tr
rater_goodness = labels_on_gold.groupby("turk")["is_correct"]
rater_goodness = rater_goodness.agg(["count","mean"])
rater_goodness.columns = ['number_of_ratings', 'average_correctness']
rater_goodness[:5]
```

```
/var/folders/fh/_y8ts9w54q5gnnnd7k56qkch0000gn/T/ipykernel_30952/2114287393.py:1: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  labels_on_gold["is_correct"] = labels_on_gold["category"] == labels_on_gold["category_
true"]
```

Out[8]:

turk	number_of_ratings	average_correctness
A112DVP1KG4QZU	1	1.000000
A1253FXHCZ9CWM	29	0.517241
A12CY1Q7XKJJDE	1	1.000000
A12RE8G66WTO8B	20	0.750000
A12Y1GTGIQDGRA	3	0.333333

Q4. Odds Ratio

In [9]:
```
# Create new column with odds = p/(1.001-p) with p=average_correctness
rater_goodness['odds'] = rater_goodness['average_correctness']/(1.001 - rater_goodness['
rater_goodness[:10]
```

Out[9]:

turk	number_of_ratings	average_correctness	odds
A112DVP1KG4QZU	1	1.000000	1000.000000

A1253FXHCZ9CWM	29	0.517241	1.069214
A12CY1Q7XKJJDE	1	1.000000	1000.000000
A12RE8G66WTO8B	20	0.750000	2.988048
A12Y1GTGIQDGRA	3	0.333333	0.499251
A13CEW9JGDWGX1	1	1.000000	1000.000000
A13OE9GBRJ0S2U	4	0.750000	2.988048
A14IQ4GLNWNPOJ	1	1.000000	1000.000000
A153PKAL7OAY36	148	0.722973	2.600369
A1554ZM0CLKSG5	1	1.000000	1000.000000

Q5. Most accurate turks

In [10]:
```
# Mask to filter for turks with over 20 ratings
mask_atleast20ratings = (rater_goodness['number_of_ratings']>=20)

# Displaying top 10 accurate turks based on average_correctness
accurate_turks = rater_goodness[mask_atleast20ratings]
accurate_turks.sort_values(by='average_correctness',ascending=False)[:10]
```

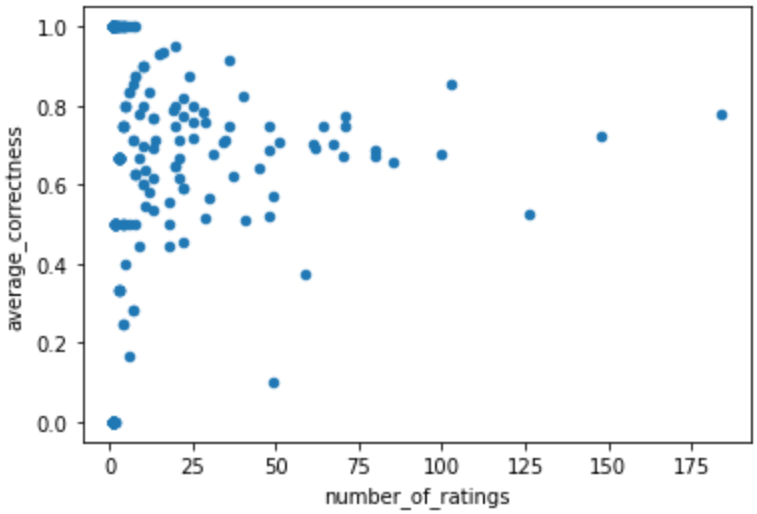
Out[10]:

turk	number_of_ratings	average_correctness	odds
A2U0R4X38GUKZE	20	0.950000	18.627451
A22C0PJUBFJTI0	36	0.916667	10.869565
A23YQUBXZPKILZ	24	0.875000	6.944444
ATVALOQVDCMZW	103	0.854369	5.826657
A1HIXWH4OXT8S4	40	0.825000	4.687500
A3220HG1O83HQ4	22	0.818182	4.475385
A32W20KGQXS0LL	25	0.800000	3.980100
A20PWAB7G3HDHU	20	0.800000	3.980100
AJSJVK40F5HM6	28	0.785714	3.649635
A31OCN4MNHUQ6W	184	0.777174	3.472222

Q6. Rating counts versus accuracy

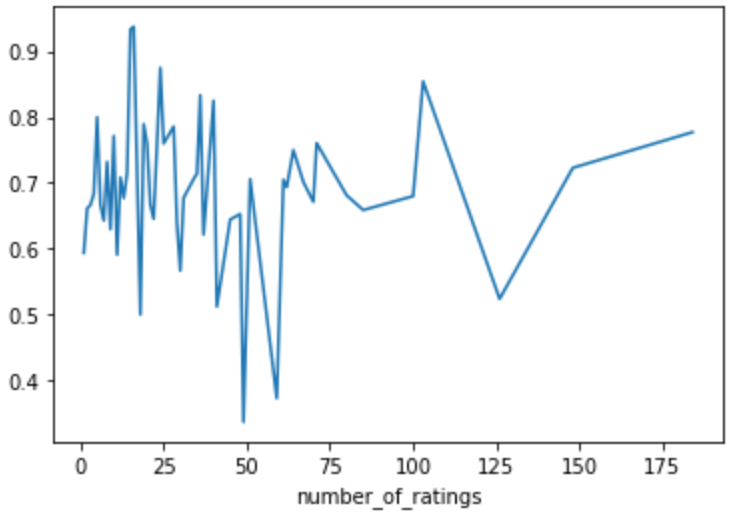
In [11]:
```
rater_goodness.plot(kind = "scatter", x = "number_of_ratings", y = "average_correctness"
```

Out[11]:
```
<AxesSubplot:xlabel='number_of_ratings', ylabel='average_correctness'>
```



In [12]:
```
rater_goodness.groupby("number_of_ratings")["average_correctness"].mean().plot()
```

Out[12]:
```
<AxesSubplot:xlabel='number_of_ratings'>
```



Our initial plot gave evidence that there might be a trend, with low count reviewers having lower mean. It is difficult to tell though, as the lower counts have a much higher variance. This is of course to be expected, as the mean of higher count values come from larger sample sizes. If we plot the mean of each proportion of correctness, we see that the mean doesn't show any clear increasing or decreasing. We thus conclude that the accuracy of the rater is not clearly a function of count

Q7. Overall predicted odds

In [13]:
```
## Finding all the labels & category combinations for non-gold urls
unknown_url_category = labels_unknown[['url','category']].drop_duplicates()

## Finding "reliable turks" who have more than 75th percentile ratings
reliable_turks_mask = (rater_goodness['number_of_ratings']>numpy.percentile(rater_goodne
reliable_turks = rater_goodness[reliable_turks_mask]

## Restricting the unknown labels to only those rated by reliable turks
reliable_turks_rating = reliable_turks.merge(labels_unknown,left_on='turk',right_on='tur

## Finding all rows of data where turk's rating of url matches (u,c)
reliable_turks_rating_match = reliable_turks_rating.merge(unknown_url_category,
                                                          left_on = ['url','category'],
                                                          right_on = ['url','category'],
                                                          suffixes=('_rel','_ng'))
```

```
reliable_turks_rating_match.set_index('turk',inplace=True)

## Finding overall odds for such combinations
odds_75 = reliable_turks_rating_match.groupby(['url','category'])[['odds']].prod()
odds_75[:5]
```

Out[13]:

		odds
url	**category**	
http://0-101.net	**G**	2.155963
http://000.cc	**G**	1.460583
http://0000.jp	**G**	14.488244
http://000relationships.com	**G**	5.681060
	P	1.851852

Q8.Predicted categories

In [14]:

```
results_75 = odds_75.unstack("category")

def top_category(x):
    return(x.idxmax()[1])

def top_odds(x):
    return(x.max())

results_75 = results_75.apply([top_odds,top_category], axis = 1)

results_75[:5]
```

Out[14]:

	top_odds	top_category
url		
http://0-101.net	2.155963	G
http://000.cc	1.460583	G
http://0000.jp	14.488244	G
http://000relationships.com	5.68106	G
http://000vitamins.com	3.784982	G

Q9. Predicted categories using more turks

In [15]:

```
## Finding all the labels & category combinations for non-gold urls
unknown_url_category25 = labels_unknown[['url','category']].drop_duplicates()

## Finding "reliable turks" who have more than 25th percentile ratings
reliable_turks_mask25 = (rater_goodness['number_of_ratings']>numpy.percentile(rater_good
reliable_turks25 = rater_goodness[reliable_turks_mask25]

## Restricting the unknown labels to only those rated by reliable turks
reliable_turks_rating25 = reliable_turks25.merge(labels_unknown,left_on='turk',right_on=

## Finding all rows of data where turk's rating of url matches (u,c)
reliable_turks_rating_match25 = reliable_turks_rating25.merge(unknown_url_category25,
                                                              left_on = ['url','category'],
                                                              right_on = ['url','category'],
                                                              suffixes=('_rel','_ng'))
```

```
reliable_turks_rating_match25.set_index('turk',inplace=True)

## Finding overall odds for such combinations
odds_25 = reliable_turks_rating_match25.groupby(['url','category'])[['odds']].prod()

results_25 = odds_25.unstack("category")
results_25 = results_25.apply([top_odds,top_category], axis = 1)

cross_results = results_25.merge(results_75, left_on = "url", right_on = "url", suffixes
cross_results
final_df = cross_results.groupby(['top_category_75','top_category_25'])[['url']].count()
final_df
```

Out[15]:

		url		
top_category_25	G	P	R	X
top_category_75				
G	8327	574	186	216
P	189	328	47	19
R	21	34	128	25
X	27	6	26	457

In [16]:

```
## Finding Errors

final_df_unpivot = final_df.stack().reset_index()
mask = (final_df_unpivot['top_category_75']!=final_df_unpivot['top_category_25'])

df_masked = final_df_unpivot[mask]
df_masked_max = pd.DataFrame(df_masked.groupby('top_category_75')['url'].max()).reset_in

error_df = df_masked.merge(df_masked_max, left_on = ['top_category_75','url'], right_on
error_df.set_index('top_category_75',inplace=True)

## Greatest misclassification for each category_75
error_df
```

Out[16]:

	top_category_25	url
top_category_75		
G	P	574
P	G	189
R	P	34
X	G	27

In [17]:

```
## Greatest misclassification overall
error_df[error_df['url']==error_df['url'].max()]
```

Out[17]:

	top_category_25	url
top_category_75		
G	P	574

The most disagreements are where the results_25 predict P but the results_75 predict G