Run this notebook to check that your annotated data is in the proper format; replace the paths in the cell below and execute all cells to validate.

In [39]:
```
# Change these files to point to your data
adjudicated_path="adjudicated.txt"
individual_annotation_path="individual_annotations.txt"
```

In [40]:
```
from collections import Counter
import numpy as np
```

In [41]:
```
def check_file(filename, min_count):
    annotator_triples={}
    annos_by_data_id={}
    with open(filename, encoding="utf-8") as file:
        for idx, line in enumerate(file):
            cols=line.rstrip().split("\t")
            assert len(cols) == 4, "%s does not have 4 columns" % cols
            assert len(cols[3]) > 0, "text #%s# in row %s is empty" %
            assert len(cols[2]) > 0, "label #%s# in row %s is empty" %
            annotator_triples[cols[1], cols[0], cols[2]]=1
            annos_by_data_id[cols[0]]=1
        assert len(annos_by_data_id) >= min_count, "You must have at l

        print("This file looks to be in the correct format; %s data po
    return list(annotator_triples.keys())
```

In [42]:
```
adjudicated=check_file(adjudicated_path, 1000)
```

```
This file looks to be in the correct format; 1004 data points
```

In [43]:
```
def check_individual_file(filename):
    annotator_triples={}
    annos_by_data_id={}
    annos_by_annotator={}
    labels={}
    with open(filename, encoding="utf-8") as file:
        count=0
        for idx, line in enumerate(file):
            cols=line.rstrip().split("\t")
            data_id=cols[0]
            anno_id=cols[1]
            label=cols[2]

            assert len(cols) == 4, "%s does not have 4 columns" % cols
            assert len(cols[3]) > 0, "text #%s# in row %s is empty" %
            assert len(label) > 0, "label #%s# in row %s is empty" % (
            count+=1

            annotator_triples[anno_id, data_id, label]=1

            if data_id not in annos_by_data_id:
                annos_by_data_id[data_id]={}
            annos_by_data_id[data_id][anno_id]=1

            if anno_id not in annos_by_annotator:
                annos_by_annotator[anno_id]={}
            annos_by_annotator[anno_id][data_id]=1

            if label not in labels:
                labels[label]=0
            labels[label]+=1


    assert len(annos_by_data_id) >= 0, "You must have labels for at le

    for data_id in annos_by_data_id:
        assert len(annos_by_data_id[data_id]) == 2, "Each data point m

    print("Annotators:\n")
    for anno_id in annos_by_annotator:
        print("%s: %s" % (anno_id, len(annos_by_annotator[anno_id])))

    print("\nLabels:\n")
    for label in labels:
        print("%s: %s" % (label, labels[label]))

    print("\nThis file looks to be in the correct format; %s data poin
    return list(annotator_triples.keys())
```

In [44]:
```
annotation_triples=check_individual_file(individual_annotation_path)
```

```
Annotators:

stalebi: 504
zhossainzadeh: 504

Labels:

4: 76
3: 253
2: 336
1: 343

This file looks to be in the correct format; 504 data points; 1008 an
notations
```

Execute the following cell to calculate Fleiss' kappa on your individual annotations.

In [45]:
```
def fleiss(annotation_triples):
    cats={}
    items={}
    uid_counts=Counter()
    uid_id={}
    aid_counts=Counter()

    # get label categories and unique data points
    for aid, uid, label in annotation_triples:
         if label not in cats:
            cats[label]=len(cats)
            if uid not in uid_id:
                uid_id[uid]=len(uid_id)

            uid_counts[uid]+=1

    ncats=len(cats)
    ps=np.zeros(ncats)

    data = []

    for aid, uid, label in annotation_triples:

        if uid not in items:
            items[uid]=np.zeros(ncats)

        items[uid][cats[label]]+=1
        ps[cats[label]]+=1

    ps/=np.sum(ps)
```

```
    ps/ np.sum(ps)

    expected=0.
    for i in range(ncats):
        expected+=ps[i]*ps[i]

    agreements=[]
    for item in items:
        total=np.sum(items[item])
        assert total >= 2, "every data point must have at least two an
        summ=0

        for i in range(ncats):
            summ+=items[item][i]*(items[item][i]-1)
        summ/=(total*(total-1))

        agreements.append(summ)

    observed=np.mean(agreements)
    print ("Observed: %.3f" % (observed))
    print ("Expected: %.3f" % (expected))
    print ("Fleiss' kappa: %.3f" % ((observed-expected)/(1-expected)))
```

In [46]:
```
fleiss(annotation_triples)
```

```
Observed: 0.752
Expected: 0.296
Fleiss' kappa: 0.648
```

In [ ]:
```

```