In [1]:
```
NAME = "Dustin Seltz"
```

Purpose:

This aims to answer Question 4: Taking into account the student's progress and goals, what is the best set of kanji / vocab to teach to them next?

This file aims to use the frequency information regarding Twitter, News, Wikipedia, and Aozora (from [https://scriptin.github.io/kanji-frequency/ (https://scriptin.github.io/kanji-frequency/)](https://scriptin.github.io/kanji-frequency/)) and compare it to the difficulty levels and frequency from WaniKani, JLPT, Grade, and Genki in order to tell a user (who wants to learn how to read one Twitter or News or Wikipedia or Aozora) the optimal sequence to follow in learning Kanji.

Input:

cleaned_link.csv This file contains information for the 2136 Jōyō kanji. This program uses the difficulty levels and frequency information.

Output:

Tells the user which learning sequence should be used to quickly learn each of the fours datasources. Currently only inline output, no csv.

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
from numpy import isnan
```

In [3]:
```
##filename = "combined_genki_lessons.csv"
#filename = "new_combined_genki.csv"
filename = "../Question1/cleaned_link.csv"
df = pd.read_csv(filename)
print(len(df))
df.head()
#Index and Unnamed: 0 are the Joyo ranking.
#When using the Joyo ranking I use the Unnamed: 0 as the column, so the index shoul
d be irrelevant to this program.
#I would clean that up, but I actually just decided to remove Joyo analysis since i
t wasn't really relevant
```

```
2136
```

Out[3]:

	Unnamed: 0	Unnamed: 0.1	kanji	strokes	frequency	grade	jlpt	parts	radicals	on_readings	...	Number c Appearance on Wikipedi
0	0	0	亜	7.0	1509.0	junior high	N1	['一', '｜', '口']	{'二': 'two'}	['ア']	...	172858.
1	1	1	哀	9.0	1715.0	junior high	N1	['亠', '口', '衣']	{'口': 'mouth, opening'}	['アイ']	...	19390.
2	2	2	挨	10.0	2258.0	junior high	NaN	['厶', '扎', '矢', '乞']	{'手 (扌龵)': 'hand'}	['アイ']	...	12111.
3	3	3	愛	13.0	640.0	grade 4	N3	['冖', '夂', '心', '爪']	{'心 (忄, ⺗)': 'heart'}	['アイ']	...	754387.
4	4	4	曖	17.0	NaN	junior high	NaN	['冖', '夂', '心', '日', '爪']	{'日': 'sun, day'}	['アイ']	...	116055.

5 rows × 28 columns

In [4]:
```
#The source is actually not clear on where exactly the "News" is sourced
datasources = ["Twitter", "Aozora", "Wikipedia"]#, "News"]
kanjis = df["kanji"]
```

In [5]:
```
def createBins(numberOfBins, col):
    #Create a sorted version of the column. #No, that'll ruin the ordering
    #arr = []
    #for value in col:
    #    arr.append(value)
    #arr.sort()

    #qcut to get bin numbers for the column.
    #    Each column's kanji now has a numeric level equivalent based on frequency.
    #    This will allow us to compare bins, like N5 through N1 vs bins 1 through
5.
    bins = pd.qcut(col, numberOfBins, labels=False)
    #Lets not start at 0, levels start at 1 for everything.
    bins += 1
    print("Created bin column:", bins)
    print("Using ranges ", pd.qcut(col, numberOfBins))
    return bins
```

In [6]:
```
#Col name should be the col of the dataframe with the levels,
#    translator translates those level strings to integer levels from 1..max_level,
inclusive,
#        Ex: "N5" translator should say is 1, while "N1" should be 5. (lowest to hi
ghest difficulty)
#    max_level is how many levels, like 60 for WaniKani's 1..60 system, or 5 for JL
PT.
def getAvgLevelDiff(level_col_name, translator, max_level):
    results = []
    for sourceName in datasources:
        bins = createBins(max_level, df["Rank of Appearances on "+sourceName])
        rankLevel_col = pd.DataFrame(bins)
        rankLevel_col.columns = ["Rank Converted to Level"]
        new_df = df.join(rankLevel_col)
        numberOfComparisons = 0
        numberOfLevelDifference = 0
        for kanji in kanjis:
            row = new_df[kanjis == kanji].iloc[0]
            colValue = row[level_col_name]
            #We need a numeric representation of the level, which our caller will d
efine
            colLevel = translator(colValue)
            #We then separate into bins for comparison.
            #Then take the average of ("level" (bin) number compared with the actua
l level)
            rankLevel = row["Rank Converted to Level"]
            diff = abs(colLevel - rankLevel)
            if(not isnan(diff)):
                numberOfComparisons += 1
                numberOfLevelDifference += diff
                print(kanji+": level "+str(colValue)+" translated to "+str(colLeve
l)+" and corresponds to rank "
                      +str(rankLevel)+" with a diff of abs(level - rank)="+str(dif
f))
        averageLevelDifference = numberOfLevelDifference / numberOfComparisons
        results.append(averageLevelDifference)
        print(sourceName+" vs "+level_col_name+" average level difference="+str(ave
rageLevelDifference))
    return results
```

In [7]:
```
#How strongly does WaniKani level corrolate with each source?
#WaniKani levels range from 1 to 60, with higher being harder (or rather, learned l
ater. Harder or more obscure).
intIsJustItself = lambda x: x
wani_levels = 60
wani_results = getAvgLevelDiff("wanikani_level", intIsJustItself, wani_levels)
```

```
Created bin column: 0       38.0
1       36.0
2       36.0
3        2.0
4       52.0
5        4.0
6       30.0
7       26.0
8       35.0
9       54.0
10      35.0
11       5.0
12      19.0
13      28.0
14       7.0
15      25.0
16       7.0
17      24.0
18      24.0
19      38.0
20      36.0
21      43.0
22      21.0
23      59.0
24      31.0
25      58.0
26      25.0
27      22.0
28      24.0
29      36.0
        ...
2106    53.0
2107    10.0
2108    53.0
2109    14.0
2110    53.0
2111    57.0
2112    13.0
2113    35.0
2114    24.0
2115    25.0
2116    51.0
2117    15.0
2118    43.0
2119    30.0
2120    47.0
2121    52.0
2122    38.0
2123    49.0
2124    22.0
2125    16.0
2126    57.0
2127    19.0
2128     8.0
2129     2.0
2130    51.0
2131    38.0
2132    21.0
2133    37.0
2134    36.0
2135    27.0
Name: Rank of Appearances on Twitter, Length: 2136, dtype: float64
Using ranges  0       (1364.883, 1401.367]
1         (1285.917, 1326.4]
```

In [8]:
```
#How strongly does JLPT level corrolate with each source?
#Low N# means higher level. Scale of N5 to N1.
JLPT_levels = 5
levelValues = {"N"+str(i): (JLPT_levels+1)-i for i in range(1, JLPT_levels+1)}
levelValues["none"] = 0 #'none' is support for the queries at the end of this file
print(levelValues)
def translateJLPT(levelStr):
    try:
        return levelValues[levelStr]
    except:
        return float('nan')
print(translateJLPT("N1"))
JLPT_results = getAvgLevelDiff("jlpt", translateJLPT, JLPT_levels)
```

```
{'N1': 5, 'N2': 4, 'N3': 3, 'N4': 2, 'N5': 1, 'none': 0}
5
Created bin column: 0       4.0
1       3.0
2       3.0
3       1.0
4       5.0
5       1.0
6       3.0
7       3.0
8       3.0
9       5.0
10      3.0
11      1.0
12      2.0
13      3.0
14      1.0
15      3.0
16      1.0
17      2.0
18      2.0
19      4.0
20      3.0
21      4.0
22      2.0
23      5.0
24      3.0
25      5.0
26      3.0
27      2.0
28      2.0
29      3.0
       ... 
2106    5.0
2107    1.0
2108    5.0
2109    2.0
2110    5.0
2111    5.0
2112    2.0
2113    3.0
2114    2.0
2115    3.0
2116    5.0
2117    2.0
2118    4.0
2119    3.0
2120    4.0
2121    5.0
2122    4.0
2123    5.0
2124    2.0
2125    2.0
2126    5.0
2127    2.0
2128    1.0
2129    1.0
2130    5.0
2131    4.0
2132    2.0
2133    4.0
2134    3.0
2135    3.0
Name: Rank of Appearances on Twitter, Length: 2136, dtype: float64
```

In [9]:
```
#How strongly does grade level corrolate with each source?
print(df["grade"].unique())
gradeLevels = {
              'none': 0, #'none' is support for the queries at the end of this file
              'grade 1': 1,
              'grade 2': 2,
              'grade 3': 3,
              'grade 4': 4,
              'grade 5': 5,
              'grade 6': 6,
              'junior high': 7,
              }
def translateGradeLevel(levelStr):
    try:
        return gradeLevels[levelStr]
    except:
        return float('nan')

grade_levels = 7
grade_results = getAvgLevelDiff("grade", translateGradeLevel, grade_levels)
```

```
['junior high' 'grade 4' 'grade 3' 'grade 5' 'grade 6' 'grade 1' 'grade 2'
 nan]
Created bin column: 0       5.0
1       5.0
2       5.0
3       1.0
4       6.0
5       1.0
6       4.0
7       3.0
8       4.0
9       7.0
10      5.0
11      1.0
12      3.0
13      4.0
14      1.0
15      3.0
16      1.0
17      3.0
18      3.0
19      5.0
20      5.0
21      5.0
22      3.0
23      7.0
24      4.0
25      7.0
26      3.0
27      3.0
28      3.0
29      5.0
       ...
2106    7.0
2107    2.0
2108    7.0
2109    2.0
2110    7.0
2111    7.0
2112    2.0
2113    5.0
2114    3.0
2115    3.0
2116    6.0
2117    2.0
2118    6.0
2119    4.0
2120    6.0
2121    7.0
2122    5.0
2123    6.0
2124    3.0
2125    2.0
2126    7.0
2127    3.0
2128    1.0
2129    1.0
2130    6.0
2131    5.0
2132    3.0
2133    5.0
2134    5.0
2135    4.0
Name: Rank of Appearances on Twitter, Length: 2136, dtype: float64
```

In [10]:
```
#How strongly does Jisho frequency level corrolate with each source?
print(max(df["frequency"].unique())) #Ranges from 1 to 2495. So it's including more
entries than joyo.
jisho_levels = 2495
#See comment on Joyo below. Bugged, and not really meaningful anyway.
#jisho_results = getAvgLevelDiff("frequency", intIsJustItself, jisho_levels)
```

```
2495.0
```

In [11]:
```
#How strongly does Joyo rank corrolate with each source?
#TODO should really just rename that column.
print(max(df["Unnamed: 0"].unique())) #Ranges from 0 to 2135.
joyo_levels = 2136
#We need level numbers to be 1..joyo_levels, inclusive
def translateJoyo(x):
    try:
        return x+1
    except:
        return float('nan')
#Those bins look a little weird? Why's it using e^0 to e^3?
#    Each bin should correspond to a level number. We have more levels than bins th
ough so something's wrong here.
#    The bin function the way I'm doing it must not support having more bins than e
lements to qcut.
#But really, these results aren't useful anyway since you don't learn by Joyo ranki
ng. Same with Jisho.
#joyo_results = getAvgLevelDiff("Unnamed: 0", translateJoyo, joyo_levels)
```

```
2135
```

In [12]:
```
#How strongly does Genki frequency level corrolate with each source?

#max isn't working on this? NaN throwing it off ?
possibleGenkiValues = df["Genki_Lesson"].unique()
possibleGenkiValues.sort()
print(possibleGenkiValues)

genki_levels = 0
for value in df["Genki_Lesson"].unique():
    if(not isnan(value)):
        genki_levels += 1
print(genki_levels, "possible valid values.")

#It looks like there is no lesson before lesson 3. A consequence of our source?
#http://genki.japantimes.co.jp/self/genki-kanji-list-linked-to-wwkanji
#We need level numbers to be 1..genki_levels, inclusive
def translateGenki(x):
    try:
        return x-2
    except:
        return float('nan')
#Test that we get 1..genki_levels
print("1 =?=", translateGenki(3))
print(genki_levels, "=?=", translateGenki(23))
genki_results = getAvgLevelDiff("Genki_Lesson", translateGenki, genki_levels)
```

```
[ 3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18. 19. 20.
 21. 22. 23. nan]
21 possible valid values.
1 =?= 1
21 =?= 21
Created bin column: 0       14.0
1       13.0
2       13.0
3        1.0
4       18.0
5        2.0
6       11.0
7        9.0
8       12.0
9       19.0
10      13.0
11       2.0
12       7.0
13      10.0
14       3.0
15       9.0
16       3.0
17       9.0
18       9.0
19      14.0
20      13.0
21      15.0
22       8.0
23      21.0
24      11.0
25      21.0
26       9.0
27       8.0
28       9.0
29      13.0
        ...
2106    19.0
2107     4.0
2108    19.0
2109     5.0
2110    19.0
2111    20.0
2112     5.0
2113    13.0
2114     9.0
2115     9.0
2116    18.0
2117     6.0
2118    16.0
2119    11.0
2120    17.0
2121    19.0
2122    14.0
2123    18.0
2124     8.0
2125     6.0
2126    20.0
2127     7.0
2128     3.0
2129     1.0
2130    18.0
2131    13.0
2132     8.0
2133    13.0
```

In [13]:
```
testq = [1,4,2,3]
pd.qcut(testq, 4, labels=False) + 1
```

Out[13]:
```
array([1, 4, 2, 3], dtype=int64)
```

In [14]:
```
#Test that the output of twitter freq vs twitter freq is 0
test_levels = 4 #Four is arbitrary. This is a made up learning sequence that corres
ponds to frequency perfectly.
testColName = "Twitter Test"
#I need to qcut the rank col, with lower ranks being in lower bin numbers
rankColName = "Rank of Appearances on Twitter"
rankCol = df[rankColName]
print(rankCol.head())
testCol = pd.qcut(rankCol, test_levels, labels=False)
print("Bins:", pd.qcut(rankCol, test_levels).head())
testCol = testCol+1
df[testColName] = testCol
#I think something's wrong with my test binning. First 3 are level 3? They should b
e common.
print(df[testColName].head())
#Just a number ?
def translateTest(x):
    try:
        return x
    except:
        return float('nan')
sampleTestRow = df.loc[df["kanji"] == "亜"]
print("Sample has value", sampleTestRow[testColName], "and rank", sampleTestRow[ran
kColName])
#ctrl f and you can find "Twitter vs Twitter Test average level difference=0.0"
test_results = getAvgLevelDiff(testColName, translateTest, test_levels)
```

```
0    1391.0
1    1307.0
2    1292.0
3      56.0
4    2068.0
Name: Rank of Appearances on Twitter, dtype: float64
Bins: 0    (1087.5, 1739.75]
1    (1087.5, 1739.75]
2    (1087.5, 1739.75]
3      (0.999, 536.25]
4    (1739.75, 4490.0]
Name: Rank of Appearances on Twitter, dtype: category
Categories (4, interval[float64]): [(0.999, 536.25] < (536.25, 1087.5] < (1087.
5, 1739.75] < (1739.75, 4490.0]]
0    3.0
1    3.0
2    3.0
3    1.0
4    4.0
Name: Twitter Test, dtype: float64
Sample has value 0    3.0
Name: Twitter Test, dtype: float64 and rank 0    1391.0
Name: Rank of Appearances on Twitter, dtype: float64
Created bin column: 0       3.0
1       3.0
2       3.0
3       1.0
4       4.0
5       1.0
6       2.0
7       2.0
8       3.0
9       4.0
10      3.0
11      1.0
12      2.0
13      2.0
14      1.0
15      2.0
16      1.0
17      2.0
18      2.0
19      3.0
20      3.0
21      3.0
22      2.0
23      4.0
24      3.0
25      4.0
26      2.0
27      2.0
28      2.0
29      3.0
       ...
2106    4.0
2107    1.0
2108    4.0
2109    1.0
2110    4.0
2111    4.0
2112    1.0
2113    3.0
2114    2.0
2115    2.0
```

In [15]:
```
#Now compare them. If I want to read Twitter, what's the best option to learn? Wiki
pedia? Etc.
#Note that there is some inherent rounding with 5 levels (JLPT) versus 60 (WaniKan
i).
#  I quantify the results as an average percentage.
#  20% inaccuracy would be 1 level off for JLPT, or 12 for WaniKani.
results = [(wani_results, wani_levels, "WaniKani"),
           (JLPT_results, JLPT_levels, "JLPT"),
           (grade_results, grade_levels, "Grade"),
           #(jisho_results, jisho_levels, "Jisho"), #This is frequency, not really
a sequence you'd learn.
           #(joyo_results, joyo_levels, "Jōyō"),  #This is a ranking system, not ve
ry relevant.
           (genki_results, genki_levels, "Genki")]
#String for datasource name, string for best sequence, float for % match.
#Data in this is modified in the loop below.
best_sequence_for_sources = {datasources[i]: ("Name of best sequence goes here", 1.
0)
                             for i in range(len(datasources))}

percent_results_of_each_source = [[] for _ in range(len(datasources))]
for (result, level, name) in results:
    i = 0
    for datasource in datasources:
        correlationWithThisSource = result[i]
        correlationWithThisSource = correlationWithThisSource / level
        percent_results_of_each_source[i].append(correlationWithThisSource)
        print(datasource, name, correlationWithThisSource)
        if(best_sequence_for_sources[datasource][1] > correlationWithThisSource):
            new_tup = (name, correlationWithThisSource)
            best_sequence_for_sources[datasource] = new_tup
        i = i + 1

#TODO refactor: I really should just use a dataframe from the start
#We could maybe just say each has some +- inaccuracy based on number of levels as w
ell.
#We also could maybe say something about coverage, particularly for Genki
result_df = pd.DataFrame(percent_results_of_each_source)
result_df.columns = [name for (_, _, name) in results]
result_df.index = [source for source in datasources]
print(result_df)

print("")
print("Results: ")
for result in best_sequence_for_sources:
    print("Best for learning to read", result, ": ",
          best_sequence_for_sources[result][0], "with"+" {:.2f}".format(best_sequen
ce_for_sources[result][1]*100),
          "% inaccuracy compared to actual usage on", result)
```

```
Twitter WaniKani 0.14522003034901365
Aozora WaniKani 0.15872534142640363
Wikipedia WaniKani 0.15977069634125782
Twitter JLPT 0.23180428134556577
Aozora JLPT 0.23574338085539717
Wikipedia JLPT 0.25132382892057026
Twitter Grade 0.23840837415285512
Aozora Grade 0.23525864379522915
Wikipedia Grade 0.2436738519212746
Twitter Genki 0.39556962025316456
Aozora Genki 0.38276069921639544
Wikipedia Genki 0.3693490054249548
           WaniKani      JLPT     Grade     Genki
Twitter    0.145220  0.231804  0.238408  0.395570
Aozora     0.158725  0.235743  0.235259  0.382761
Wikipedia  0.159771  0.251324  0.243674  0.369349

Results:
Best for learning to read Twitter :  WaniKani with 14.52 % inaccuracy compared t
o actual usage on Twitter
Best for learning to read Aozora :  WaniKani with 15.87 % inaccuracy compared to
actual usage on Aozora
Best for learning to read Wikipedia :  WaniKani with 15.98 % inaccuracy compared
to actual usage on Wikipedia
```

It looks like we have a reasonable amount of variation. These learning sequences weren't chosen randomly, but can't perfectly model use while teaching in a way that makes sense.

Notes about shortcomings of this comparison:

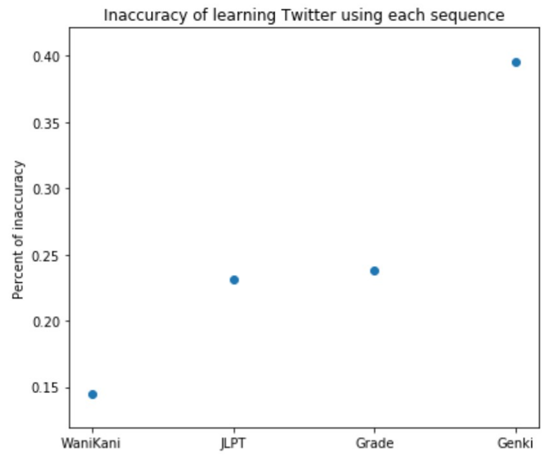
We only have the WaniKani/JLPT/Genki/Grade data for the Jōyō set, results might vary with more than 2136 kanji.

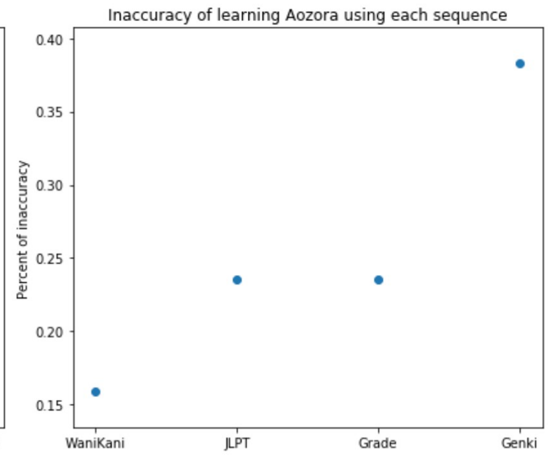
Coverage is taken into consideration in a somewhat strange way due to how I binned frequency ranks. By saying the levels and frequencies should correspond, I am assuming two things:

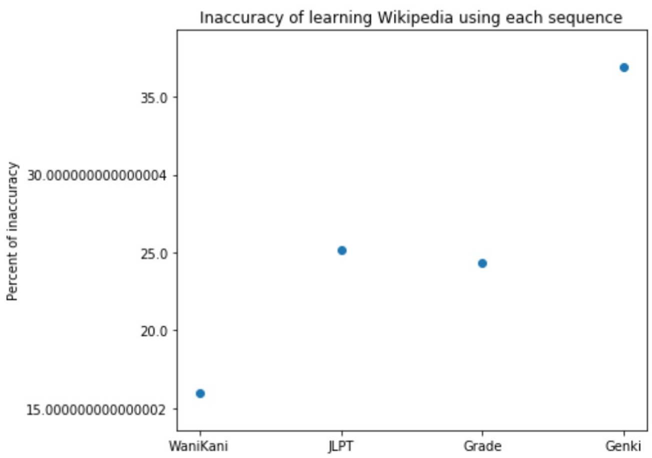
- The learning sequence is trying to teach the whole Joyo set. If the sequence's last levels correspond to earlier bin levels, the algorithm penalises it because it sees them as having poor correlation. This is part of why Genki scores so poorly.
- The learning sequence distributes kanjis roughly evenly between levels. This is part of why WaniKani scores so well

In any case, this comparison should at least tell you which source a sequence best matches (for example, WaniKani teaches Twitter better than Wikipedia, slightly).

In [16]:
```
#https://stackoverflow.com/a/43348337
import matplotlib.ticker as ticker
#Currently this xaxis function is unused, doesn't work for scatter plot or somethin
g.
#Turn the x axis into names of sources
def formatterX(x, pos):
    #The name of the data source
    return results[x][2]
#Translate 0.0 to 1.0 to 0.0 to 100.0
#Long floating points is an issue.
#Maybe force the y axis to use nice round numbers? 0, 10, 20...
#    Maybe we won't end up using this type of chart though.
def formatterY(y, pos):
    return y*100

fig = plt.figure(figsize=(13,10))

dataX = [i for i in range(0, len(results))]
dataY = []
for source in datasources:
    dataYPiece = [val for val in result_df.loc[source]]
    dataY.append(dataYPiece)
sequence_names = [val[2] for val in results]
#Only need colors if I show them all on the same vertical line.
#colors = ["black", "blue", "red", "green"]
#correspondingColors = [colors[0] for i in range(1, 4*4+1)]
for ax_index in range(0, len(datasources)):
    ax = fig.add_subplot(2, 2, ax_index+1)
    ax.scatter(dataX, dataY[ax_index])
    props = {
        #Inaccuracy is a strange word to use here. If there were 4 levels and it wa
s
        #    all off by 1 it'd be 25% "inaccuracy," right (should probably develop
better tests, hard to
        #    verify these large calculations)? So more, the average difference in l
evel.
        'title': 'Inaccuracy of learning '+datasources[ax_index]+' using each seque
nce',
        'ylabel': 'Percent of inaccuracy'
    }
    ax.set(**props)
    plt.xticks(range(len(results)), sequence_names, size='medium')
#plt.gca().xaxis.set_major_formatter(ticker.FuncFormatter(formatterX))#Doesn't work
for scatter plot?
plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter(formatterY))

#fig.subplots_adjust(wspace=0, hspace=0)
#Prevent overlap
fig.tight_layout()

None
```

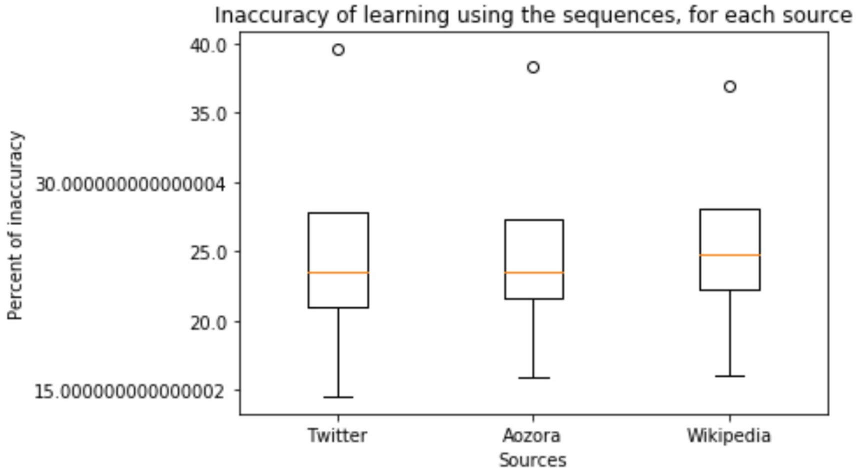

Inaccuracy of learning Twitter using each sequence
Percent of inaccuracy
0.40
0.35
0.30
0.25
0.20
0.15
WaniKani
JLPT
Grade
Genki


Inaccuracy of learning Aozora using each sequence
Percent of inaccuracy
0.40
0.35
0.30
0.25
0.20
0.15
WaniKani
JLPT
Grade
Genki


Inaccuracy of learning Wikipedia using each sequence
Percent of inaccuracy
35.0
30.000000000000004
25.0
20.0
15.000000000000002
WaniKani
JLPT
Grade
Genki

In [17]:
```
#https://stackoverflow.com/a/43348337
import matplotlib.ticker as ticker
#Turn the x axis into names of sources
def formatterX(x, pos):
    if(x >= 1 and x <= 4):
        return datasources[x-1]
    #This shouldn't occur. Maybe print a warning but it'd be pretty noticeable.
    return x
#Translate 0.0 to 1.0 to 0.0 to 100.0
#Long floating points is an issue.
#Maybe force the y axis to use nice round numbers? 0, 10, 20...
#    Maybe we won't end up using this type of chart though.
def formatterY(y, pos):
    return y*100

fig = plt.figure()
ax1 = fig.add_subplot(1, 1, 1)
ax1.boxplot(percent_results_of_each_source)
props = {
    #Inaccuracy is a strange word to use here. If there were 4 levels and it was
    #    all off by 1 it'd be 25% "inaccuracy," right (should probably develop bett
er tests, hard to
    #    verify these large calculations)? So more, the average difference in leve
l.
    'title': 'Inaccuracy of learning using the sequences, for each source',
    'xlabel': 'Sources',
    'ylabel': 'Percent of inaccuracy'
}
ax1.set(**props)
plt.gca().xaxis.set_major_formatter(ticker.FuncFormatter(formatterX))
plt.gca().yaxis.set_major_formatter(ticker.FuncFormatter(formatterY))
print("For each source, we have considered the learning sequences from: ")
for result in results:
    print(result[2])
print("An inaccuracy of 0 would mean that the order it's taught perfectly correspon
ds to the frequency of usage")
#TODO can I draw additional conclusions from this? Maybe certain sources have more
variety, etc.
#    But I can get that from the frequency numbers and I'm not sure how useful that
'd be to know.
```

```
For each source, we have considered the learning sequences from:
WaniKani
JLPT
Grade
Genki
An inaccuracy of 0 would mean that the order it's taught perfectly corresponds t
o the frequency of usage
```



In [18]:
```python
#TODO remove this, I have this in a dataframe now.
#Print the values for each source
i=0
for source in datasources:
    print(source, percent_results_of_each_source[i])
    i += 1
```

```
Twitter [0.14522003034901365, 0.23180428134556577, 0.23840837415285512, 0.395569
62025316456]
Aozora [0.15872534142640363, 0.23574338085539717, 0.23525864379522915, 0.3827606
9921639544]
Wikipedia [0.15977069634125782, 0.25132382892057026, 0.2436738519212746, 0.36934
90054249548]
```

In [19]:
```python
#We could also store this in a csv at this point, but nothing else in the project i
s using this data.
#result_df.to_csv("level_correlation_results.csv")
```