Original Data Analysis

October 30, 2020

1 Selected Org Data Alalysis

[81]:
```
import numpy as np


import matplotlib.pyplot as plt
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
import seaborn as sns
from scipy import stats
```

2 Load Data

2.0.1 Just Testing Data, x,y,meta,coeffs

[25]:
```
X = np.load('./DATA/ORIGINAL/X_test.npy')
Y = np.load('./DATA/ORIGINAL/y_test.npy')
meta = np.load('./DATA/ORIGINAL/meta_test.npy',allow_pickle=True)
coeff = np.load('./DATA/ORIGINAL/coeff_features.npy')
```

3 See shapes and sample

[36]:
```
X.shape
```

[36]:
```
(15621, 2948)
```

[40]:
```
X[0]
```

[40]:
```
array([0., 0., 0., …, 0., 0., 0.])
```

[37]:
```
Y.shape
```

[37]:
```
(15621, 1)
```

[38]:
```
np.unique(Y,return_counts=True)
```

[38]:
```
(array([0, 1]), array([10414,  5207], dtype=int64))
```

[41]:
```
meta.shape
```

[41]:
```
(15621, 1)
```

[42]:
```
meta[0][0]
```

[42]:
```
{'sha256': '98E7842115234FD469B7B0DDE13BFEF8FC6B6C708192AA286DFFC8DD2E7D1217',
 'sha1': '5766861B1F32BB3F93E82AF4E81956B02481187C',
 'md5': 'EAB6F9C6374476045F0D46DA7B21A50C',
 'dex_date': '2017-05-04 01:54:00',
 'apk_size': 3886517,
 'pkg_name': 'com.selfie.beautyinsta.cat.kitty.zipper.screen.lock.free',
 'vercode': 1.0,
 'vt_detection': 0.0,
 'vt_scan_date': '2018-09-22 12:20:47',
 'dex_size': 5548608,
 'markets': 'play.google.com',
 'year': '2017',
 'month': '05',
 'day': '04',
 'tags': ['androzoo', '2017-h1'],
 'analysis_engines': [],
 'sample_path': '/media/nas/datasets/android/samples/Androzoo/9/8/E/98E784211523
4FD469B7B0DDE13BFEF8FC6B6C708192AA286DFFC8DD2E7D1217.apk',
 'submission_date': 1551584091}
```

[43]:
```
coeff[0]
```

[43]:
```
1.1951410224625851
```

4 EDA

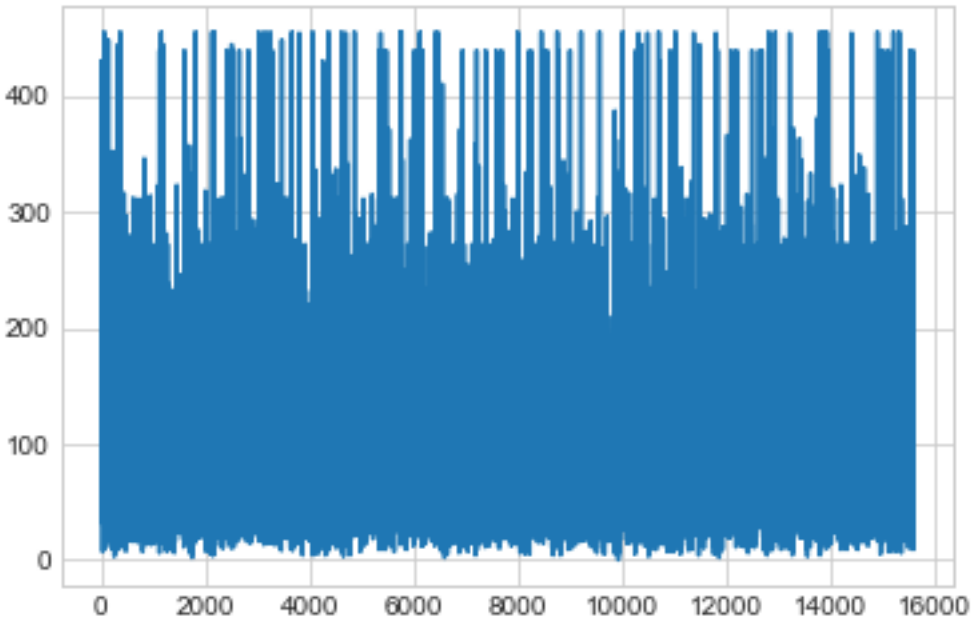
Visualizations and Analysis. Markdowns will have conclusions

5 Analysing by Row

[94]:
```
SumByRow = X.sum(axis=1)
SumByRow.shape
```

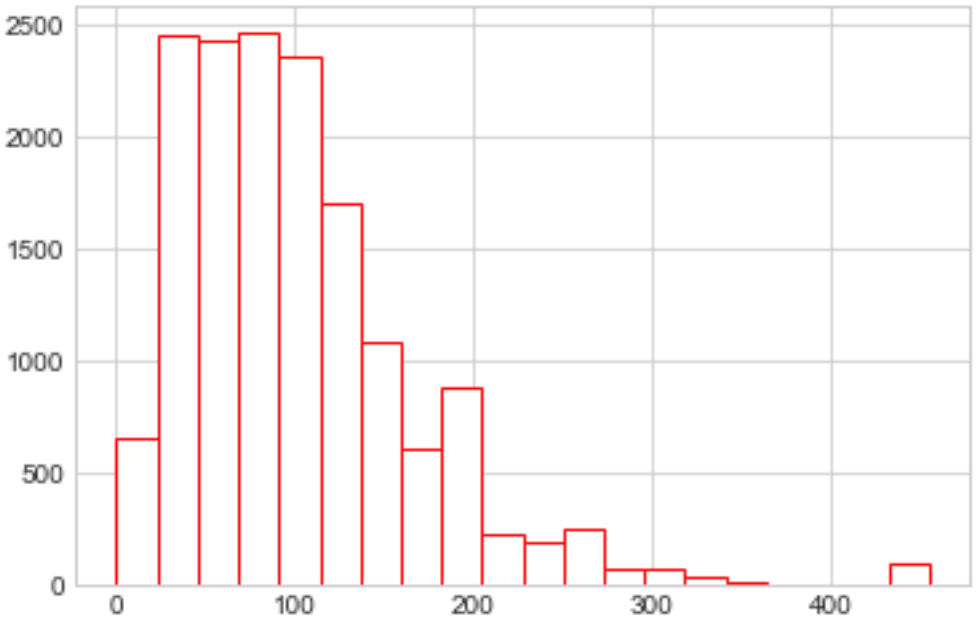
[94]:
```
(15621,)
```

[61]:
```
plt.plot(SumByRow)
plt.show()
```



As you can see, there is a little variation among the number of features in each apk. Lets now see it as a histogram for better understanding

[77]:
```
plt.hist(SumByRow,bins=20,color='white', edgecolor='red')
plt.show()
```



As you can see, a lot of rows middle around having a 100 features. Few apks having less that 30 features do exist and there are a rare few apps with over 400 features

[86]:
```
print(stats.describe(SumByRow),np.median(SumByRow))
```

```
DescribeResult(nobs=15621, minmax=(0.0, 455.0), mean=101.481979386723,
variance=4446.832533751432, skewness=1.6319371595416,
kurtosis=4.4600959165980045) 88.0
```

An app on average has 101 features and a median of 88 features

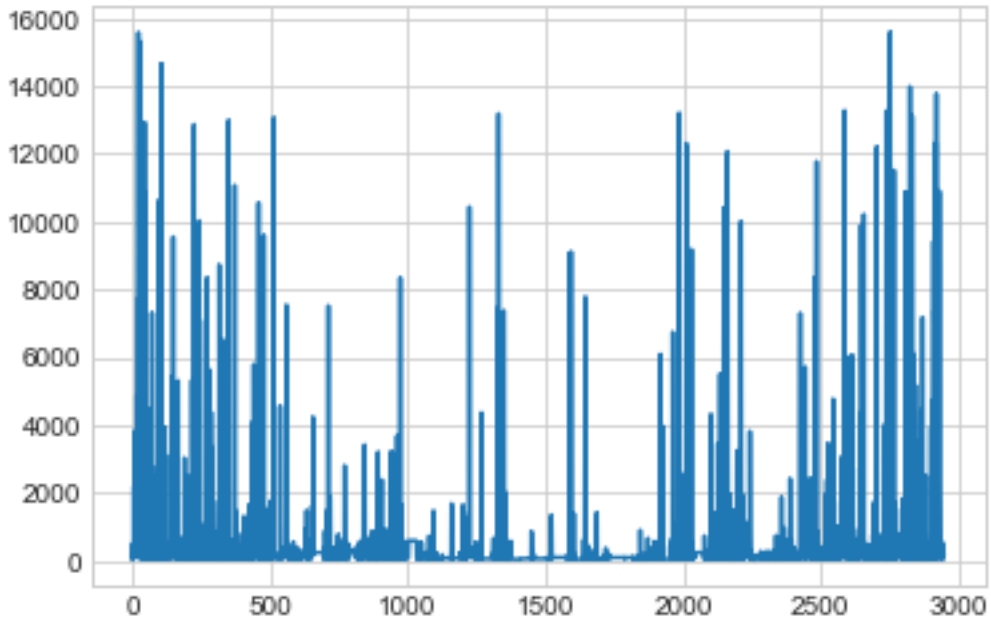
6 Analysing For Features

[88]:
```
SumByCol = X.sum(axis=0)
SumByCol.shape
```

[88]:
```
(2948,)
```

This counts the occurences of each feature across the chosen datset

[89]:
```
plt.plot(SumByCol)
plt.show()
```



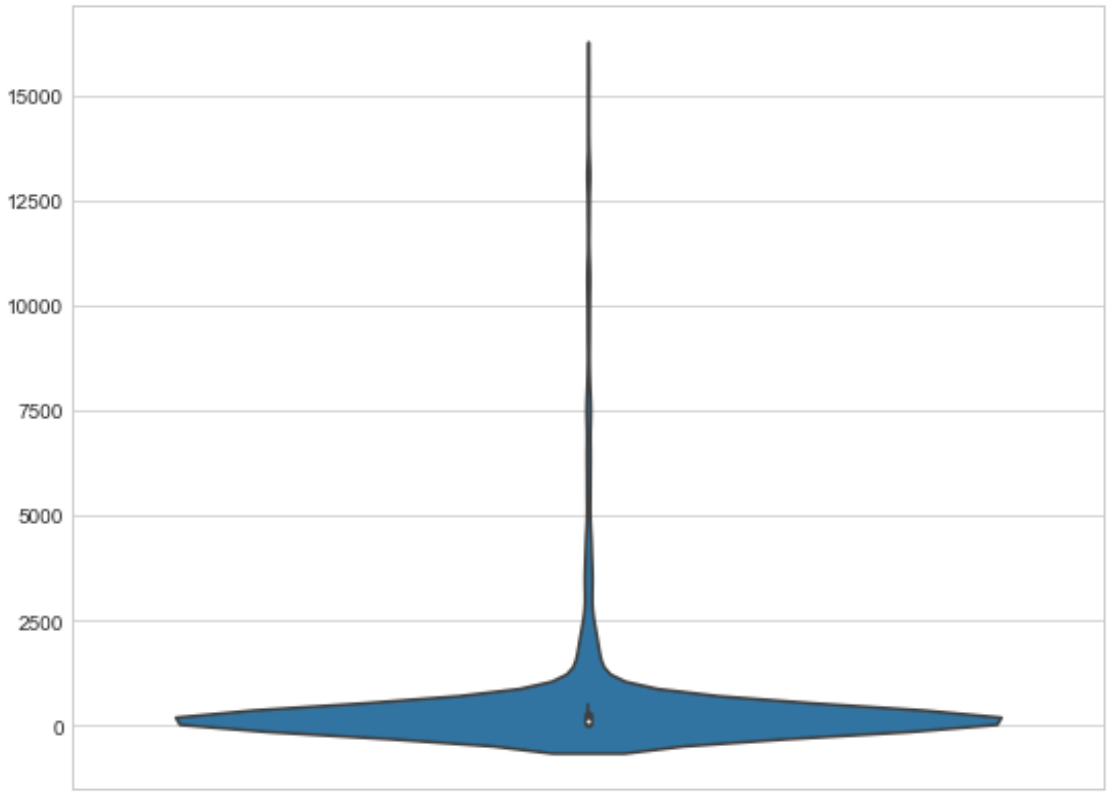
As you can see, there is a lot of variations

[93]:
```
print(stats.describe(SumByCol),np.median(SumByCol))
```

```
DescribeResult(nobs=2948, minmax=(23.0, 15603.0), mean=537.7374491180461,
variance=2761691.464128136, skewness=5.557891662948492,
kurtosis=34.13416814986031) 96.0
```

Median occurences is just 96 for a feature. But mean is 538 mostly beacsue of the outrageous value of few outliers like a feature appearing in 15603 rows with total dataset itself being only 15616

[92]:
```
fig, ax = plt.subplots(figsize =(9, 7))
sns.violinplot( ax = ax, y = SumByCol )
plt.show()
```



Above Violin Plot learly shows how a majority of features occur a small number of times

7 We have seen rows and cols and features in general. Lets now see it w.r.t malware and benign features

8 Find all ben and mal rows, bena nd mal features. Using Y and coeffs

[126]:
```
mal_col_index = []
ben_col_index = []
for i in range(len(coeff)):
    if coeff[i] > 0:
        mal_col_index.append(i)
    elif coeff[i] < 0:
        ben_col_index.append(i)
    else:
        print("DANGER")
print(len(mal_col_index),len(ben_col_index))
```

```
1513 1435
```

We have 1513 malware features and 1435 benign features

[128]:
```
ben_index = []
mal_index = []
for i in range(len(Y)):
    if Y[i] == 1:
        mal_index.append(i)
    elif Y[i] == 0:
        ben_index.append(i)
    else:
        print("DANGER")
print(len(mal_index),len(ben_index))
```

```
5207 10414
```

9 Separate Rows as Ben and Mal

[140]:
```
X_BEN = X[ben_index].copy()
X_MAL = X[mal_index].copy()
```

10 Further Analysis

11 Analyze Benign APKs with All features

[147]:
```
stats.describe(np.sum(X_BEN,axis=1))
```

[147]:
```
DescribeResult(nobs=10414, minmax=(0.0, 455.0), mean=108.31121567121183,
variance=4131.957778739038, skewness=1.9097510329919412,
kurtosis=6.628593811279975)
```

[148]:
```
np.median(np.sum(X_BEN,axis=1))
```

[148]:
```
99.0
```

A benign sample usually has around 99 features

12 Analyze Mal APKs with All features

[149]:
```
stats.describe(np.sum(X_MAL,axis=1))
```

[149]:
```
DescribeResult(nobs=5207, minmax=(4.0, 448.0), mean=87.82350681774534,
variance=4797.6121403698635, skewness=1.4041684781219137,
kurtosis=1.5131124153895437)
```

[150]:
```
np.median(np.sum(X_MAL,axis=1))
```

[150]:
```
57.0
```

A malware susally has around 57 features

13 Analyze Ben rows with ben and mal features

[187]:
```
X_BEN_MalFets = X_BEN[:,[mal_col_index]].reshape(len(X_BEN),-1)
X_BEN_MalFets.shape
```

[187]:
```
(10414, 1513)
```

[188]:
```
X_BEN_BenFets = X_BEN[:,[ben_col_index]].reshape(len(X_BEN),-1)
X_BEN_BenFets.shape

X_BEN_MalFets_SUM_BY_ROWS = np.sum(X_BEN_MalFets,axis=0)
X_BEN_BenFets_SUM_BY_ROWS = np.sum(X_BEN_BenFets,axis=0)
X_BEN_MalFets_SUM_BY_COLS = np.sum(X_BEN_MalFets,axis=1)
X_BEN_BenFets_SUM_BY_COLS = np.sum(X_BEN_BenFets,axis=1)
print(X_BEN_MalFets_SUM_BY_ROWS.shape,X_BEN_BenFets_SUM_BY_ROWS.
 ↪shape,X_BEN_MalFets_SUM_BY_COLS.shape,X_BEN_BenFets_SUM_BY_COLS.shape)
```

```
(1513,) (1435,) (10414,) (10414,)
```

[189]:
```
np.mean(X_BEN_MalFets_SUM_BY_COLS)
```

[189]:
```
51.416746687151914
```

[190]:
```
np.mean(X_BEN_BenFets_SUM_BY_COLS)
```

[190]:
```
56.89446898405992
```

Benign APKs usually have 51 malware contributing features and 57 benign contributing features

14 Analyze Mal rows with ben and mal features

[194]:
```
X_MAL_MalFets = X_MAL[:,[mal_col_index]].reshape(len(X_MAL),-1)
X_MAL_MalFets.shape
```

[194]:
```
(5207, 1513)
```

[195]:
```
X_MAL_BenFets = X_MAL[:,[ben_col_index]].reshape(len(X_MAL),-1)
X_MAL_BenFets.shape
```

[195]:
```
(5207, 1435)
```

[196]:
```
X_MAL_MalFets_SUM_BY_ROWS = np.sum(X_MAL_MalFets,axis=0)
X_MAL_BenFets_SUM_BY_ROWS = np.sum(X_MAL_BenFets,axis=0)
X_MAL_MalFets_SUM_BY_COLS = np.sum(X_MAL_MalFets,axis=1)
X_MAL_BenFets_SUM_BY_COLS = np.sum(X_MAL_BenFets,axis=1)
print(X_MAL_MalFets_SUM_BY_ROWS.shape,X_MAL_BenFets_SUM_BY_ROWS.
 ↪shape,X_MAL_MalFets_SUM_BY_COLS.shape,X_MAL_BenFets_SUM_BY_COLS.shape)
```

```
(1513,) (1435,) (5207,) (5207,)
```

[198]:
```
np.mean(X_MAL_MalFets_SUM_BY_COLS)
```

[198]:
```
47.47167274822355
```

[199]:
```
np.mean(X_MAL_BenFets_SUM_BY_COLS)
```

[199]:
```
40.3518340695218
```

Malware APKs usually have 47 malware features and 40 benign features.

[201]:
```
np.mean(X_MAL_MalFets_SUM_BY_ROWS)
```

[201]:
```
163.3740912095175
```

[202]:
```
np.mean(X_MAL_BenFets_SUM_BY_ROWS)
```

[202]:
```
146.41951219512194
```

A malicious feature appears in 163 of our 5000 odd malware samples. A benign apears 147 time sin the 5000.

[ ]:

FGSM ANALYSIS

October 30, 2020

[1]:
```
import numpy as np
```

[2]:
```
X_FULL = np.load("./DATA/FGSM/X_TEST_ORG.npy")
X_N_FULL = np.load("./DATA/FGSM/X_TEST_NOISED.npy")
Y_REAL_2_Col = np.load("./DATA/FGSM/Y_TEST_ORG.npy")
X_MALWARE_INDEX = np.load("./DATA/FGSM/X_TEST_MALWARE_INDEX.npy")
ORG_CLASSES = np.load("./DATA/FGSM/ORIGINAL_CLASSES.npy")
NOISE_CLASSES = np.load("./DATA/FGSM/NOISY_CLASSES.npy")
```

[3]:
```
Y_REAL_2_Col = Y_REAL_2_Col.reshape(-1,2)
```

[4]:
```
Y_REAL = []
for i in range(Y_REAL_2_Col.shape[0]):
    if Y_REAL_2_Col[i][0] == 1:
        Y_REAL.append(0)
    else:
        Y_REAL.append(1)
#     if Y_REAL_2_Col[i][0]==0:
#         print(Y_REAL_2_Col)
Y_REAL[:10]
```

[4]:
```
[0, 0, 1, 1, 0, 1, 1, 0, 0, 0]
```

[5]:
```
ORG_CLASSES[:10]
```

[5]:
```
array([1, 1, 0, 1, 0, 1, 1, 1, 1, 1], dtype=int64)
```

[6]:
```
NOISE_CLASSES[:10]
```

[6]:
```
array([0, 0, 1, 0, 1, 0, 0, 0, 0, 1], dtype=int64)
```

[7]:
```
c=0
for i in range(len(ORG_CLASSES)):
    if ORG_CLASSES[i] != Y_REAL[i]:
        c+=1
print(c,len(ORG_CLASSES)-c)
print((len(ORG_CLASSES)-c)/len(ORG_CLASSES))
```

```
3416 1791
0.3439600537737661
```

[8]:
```
c=0
for i in range(len(NOISE_CLASSES)):
    if NOISE_CLASSES[i] != Y_REAL[i]:
        c+=1
print(c,len(NOISE_CLASSES)-c)
print((len(NOISE_CLASSES)-c)/len(NOISE_CLASSES))
```

```
2170 3037
0.5832533128480891
```

Drop from 65 to 41

1 Just Taking Malware for all x,xnoise,y,org class,noise class

[9]:
```
X = X_FULL[X_MALWARE_INDEX].copy()
X.shape
```

[9]:
```
(5207, 2948)
```

[10]:
```
X_N = X_N_FULL[X_MALWARE_INDEX].copy()
X_N.shape
```

[10]:
```
(5207, 2948)
```

[11]:
```
Y = np.array(Y_REAL)[X_MALWARE_INDEX]
len(Y)
# np.unique(Y,return_counts=True)
np.unique(np.array(Y),return_counts=True)
```

[11]:
```
(array([1]), array([5207], dtype=int64))
```

[12]:
```
np.unique(ORG_CLASSES,return_counts=True)
```

[12]:
```
(array([0, 1], dtype=int64), array([ 284, 4923], dtype=int64))
```

[13]:
```
np.unique(NOISE_CLASSES,return_counts=True)
```

[13]:
```
(array([0, 1], dtype=int64), array([3944, 1263], dtype=int64))
```

2 1 is MALWARE

3 0 is BENIGN

Lets see how many rows remained unchanged after FGSM attack

[14]:
```
matches = [i for i in range(len(X)) if X[i].tolist()==X_N[i].tolist()]
matches
```

[14]:
```
[1657, 2416, 2504, 4701, 4879, 5058]
```

Out of 5207 malware, the 5 APKs above did not get affected/attacked at all

4 Lets Now get a list of all attacks that were successfull

[15]:
```
np.unique(ORG_CLASSES,return_counts=True)
```

[15]:
```
(array([0, 1], dtype=int64), array([ 284, 4923], dtype=int64))
```

4.0.1 The original model classified all 5207 malware as 284 benign and 4923 malware

[16]:
```
np.unique(NOISE_CLASSES,return_counts=True)
```

[16]:
```
(array([0, 1], dtype=int64), array([3944, 1263], dtype=int64))
```

4.0.2 After FGSM, we had 3944 benign and 1263 malware according to the classifier

[17]:
```
nc = 0
p2n = n2p = 0
for i in range(len(ORG_CLASSES)):
    if ORG_CLASSES[i] == NOISE_CLASSES[i]:
        nc+=1
    else:
        if ORG_CLASSES[i] == 1:
            p2n+=1
        else:
            n2p+=1
print(nc,p2n,n2p)
```

```
1073 3897 237
```

5 MISSCLASSIFICATION

[18]:
```
print("raw missclassification: "+str((3987+237)/(1073+3987+237)))
print("Accepted missclassification: "+str((3987/(1073+3987+237))))
print("Bad Missclassification: "+str(237/(1073+3987+237)))
print("observed change: "+str((3987-237)/(1073+3987+237)))
```

```
raw missclassification: 0.79743250896734
Accepted missclassification: 0.7526902020011327
Bad Missclassification: 0.04474230696620729
observed change: 0.7079478950349254
```

5.0.1 The attack successfully misclassified 3897 previously classified Malware as Benign.

5.0.2 But it also classified 237 previously classified Benign as Malware

5.0.3 Out the originally seeming 284 benign, after attacks, 237 have become malware, which is bad but we successfully hid 3897 so final result is acceptable

6 FIND ALL BEN AND MAL FEATURES AND STORE SEPARATELY

[19]:
```
coeff = np.load('./DATA/ORIGINAL/coeff_features.npy')

mal_col_index = []
ben_col_index = []
for i in range(len(coeff)):
    if coeff[i] > 0:
        mal_col_index.append(i)
    elif coeff[i] < 0:
        ben_col_index.append(i)
    else:
        print("DANGER")
print(len(mal_col_index),len(ben_col_index))
```

```
1513 1435
```

A sanity re-check if any 1s became 0s in attack

[20]:
```
c=0
cc = 0
ccc = 0
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        if X[i][j]!=X_N[i][j]:
            cc+=1
            if X[i][j]==1:
                print("DANGER")
                ccc+=1
        c+=1
if c == X.shape[0]*X.shape[1]:
    print(c)
    print("NUM CHECK OK")
print(cc)
if ccc == 0:
    print("NO Illegal changes")
```

```
15350236
NUM CHECK OK
9335133
```

```
NO Illegal changes
```

7 Analyzing the Good Attacks (the 3897)

[21]:
```
good_attacks = [i for i in range(len(ORG_CLASSES)) if ORG_CLASSES[i] !=␣
↪NOISE_CLASSES[i] and
                ORG_CLASSES[i] == 1]
len(good_attacks)
```

[21]:
```
3897
```

take copy of all good attack x and x_noise

[22]:
```
X_GOOD_CHANGE = X[good_attacks].copy()
X_N_GOOD_CHANGE = X_N[good_attacks].copy()
print(X_GOOD_CHANGE.shape,X_N_GOOD_CHANGE.shape)
```

```
(3897, 2948) (3897, 2948)
```

Define a function to calculate change of bits

[23]:
```
def manhattanDist(a,b):
    c=0
    for i in range(len(a)):
        if a[i] != b[i]:
            c+=1
    return c
```

Check out changes in the 1st sample

[24]:
```
manhattanDist(X_GOOD_CHANGE[0],X_N_GOOD_CHANGE[0])
```

[24]:
```
1666
```

[25]:
```
print(np.unique(X_GOOD_CHANGE[0],return_counts=True))
print(np.unique(X_N_GOOD_CHANGE[0],return_counts=True))
```

```
(array([0., 1.], dtype=float32), array([2914,   34], dtype=int64))
(array([0., 1.], dtype=float32), array([1248, 1700], dtype=int64))
```

As we can see, 1666 features were added

7.1 list of lists containing all added features for each row

[26]:
```
a = [c for c in range(len(X_GOOD_CHANGE[0])) if X_GOOD_CHANGE[0][c] !=␣
↪X_N_GOOD_CHANGE[0][c]]
```

[27]:
```
good_attack_changes = []
for i in range(X_GOOD_CHANGE.shape[0]):
    good_attack_changes.append([c for c in range(len(X_GOOD_CHANGE[i])) if␣
 ↪X_GOOD_CHANGE[i][c] != X_N_GOOD_CHANGE[i][c]])
len(good_attack_changes)
```

[27]:
```
3897
```

[28]:
```
good_attack_MBA = []
for i in good_attack_changes:
    m = b = 0
    for j in range(len(i)):
        if i[j] in mal_col_index:
            m+=1
        else:
            b+=1
    good_attack_MBA.append([b,m])
good_attack_MBA = np.array(good_attack_MBA)
good_attack_MBA.shape
```

[28]:
```
(3897, 2)
```

[29]:
```
np.mean(good_attack_MBA,axis=0)
```

[29]:
```
array([997.24095458, 882.85758276])
```

For Classifying a Malware into a Benign, The FGSM approximately adds around 997 extra Benign Features and 883 extra malware features in the APKs

8 Analyzing Bad Attacks (the 237)

[30]:
```
bad_attacks = [i for i in range(len(ORG_CLASSES)) if ORG_CLASSES[i] !=␣
 ↪NOISE_CLASSES[i] and
               ORG_CLASSES[i] == 0]
len(bad_attacks)
```

[30]:
```
237
```

[31]:
```
X_BAD_CHANGE = X[bad_attacks].copy()
X_N_BAD_CHANGE = X_N[bad_attacks].copy()
print(X_BAD_CHANGE.shape,X_N_BAD_CHANGE.shape)
```

```
(237, 2948) (237, 2948)
```

[32]:
```
bad_attack_changes = []
for i in range(X_BAD_CHANGE.shape[0]):
```

```
    bad_attack_changes.append([c for c in range(len(X_BAD_CHANGE[i]))
                               if X_BAD_CHANGE[i][c] != X_N_BAD_CHANGE[i][c]])
len(bad_attack_changes)
```

[32]: 237

[33]:
```
bad_attack_MBA = []
for i in bad_attack_changes:
    m = b = 0
    for j in range(len(i)):
        if i[j] in mal_col_index:
            m+=1
        else:
            b+=1
    bad_attack_MBA.append([b,m])
bad_attack_MBA = np.array(bad_attack_MBA)
bad_attack_MBA.shape
```

[33]: (237, 2)

[34]:
```
np.mean(bad_attack_MBA,axis=0)
```

[34]: array([582.91983122, 764.18987342])

For a bad attack, that is changing a ben into a mal classification, the FGSM added approx 582 Ben cols and 764 mal cols

9 Analyzing the No Changes (the 1073)

[35]:
```
none_attacks = [i for i in range(len(ORG_CLASSES)) if ORG_CLASSES[i] ==␣
 ↪NOISE_CLASSES[i]]
len(none_attacks)
```

[35]: 1073

[36]:
```
X_NONE_CHANGE = X[none_attacks].copy()
X_N_NONE_CHANGE = X_N[none_attacks].copy()
print(X_NONE_CHANGE.shape,X_N_NONE_CHANGE.shape)
```

(1073, 2948) (1073, 2948)

[37]:
```
none_attack_changes = []
for i in range(X_NONE_CHANGE.shape[0]):
    none_attack_changes.append([c for c in range(len(X_NONE_CHANGE[i]))
                               if X_NONE_CHANGE[i][c] != X_N_NONE_CHANGE[i][c]])
len(none_attack_changes)
```

[37]: 1073

[38]:
```
none_attack_MBA = []
for i in none_attack_changes:
    m = b = 0
    for j in range(len(i)):
        if i[j] in mal_col_index:
            m+=1
        else:
            b+=1
    none_attack_MBA.append([b,m])
none_attack_MBA = np.array(none_attack_MBA)
none_attack_MBA.shape
```

[38]: (1073, 2)

[39]:
```
np.mean(none_attack_MBA,axis=0)
```

[39]: array([705.82106244, 868.38583411])

in the no change category, FGSM added approx 705 ben and 868 mal cols.

10 Analyze all

[40]:
```
attack_changes = []
for i in range(X.shape[0]):
    attack_changes.append([c for c in range(len(X[i]))
                               if X[i][c] != X_N[i][c]])
len(attack_changes)
```

[40]: 5207

[41]:
```
attack_MBA = []
for i in attack_changes:
    m = b = 0
    for j in range(len(i)):
        if i[j] in mal_col_index:
            m+=1
        else:
            b+=1
    attack_MBA.append([b,m])
attack_MBA = np.array(attack_MBA)
attack_MBA.shape
```

[41]: (5207, 2)

[42]:
```
np.mean(attack_MBA,axis=0)
```

[42]:
```
array([918.33032456, 874.47416939])
```

11 Visualization < TO DO >

12 Conclusions

- FGSM does have a good missclassification
- It adds an approx 918 ben cols and 874 mal cols to a row while attacking
- When it adds more ben that mal features, then it missclassifies. It does come under this case a lot but there are a significant number of cases where it adds more malicious features to sometimes even makes a benign classification as malicious as demonstrated above

13 Missclassifications

- raw missclassification: 0.79743250896734
- Accepted missclassification: 0.7526902020011327
- Bad Missclassification: 0.04474230696620729
- observed change: 0.7079478950349254

[ ]:
```

```

JSMA ANALYSIS

October 30, 2020

[4]:
```
import numpy as np
```

[11]:
```
X_FULL = np.load('./DATA/ORIGINAL/X_test.npy')
Y_FULL = np.load('./DATA/ORIGINAL/y_test.npy')
coeff = np.load('./DATA/ORIGINAL/coeff_features.npy')
X_N_FULL = np.load('./DATA/JSMA/x_test_noised_benign_JSMA.npy')
```

1 for JSMA, we only replace the successfully attacked rows. So analyzing the changed rows b/w x and x_n should suffice as no other rows were changed. We wouldn't need Y also because we know that only malware->Benign changes were done.

2 Sanity Check

[24]:
```
c=0
cc = 0
ccc = 0
for i in range(X_FULL.shape[0]):
    for j in range(X_FULL.shape[1]):
        if X_FULL[i][j]!=X_N_FULL[i][j]:
            cc+=1
            if X_FULL[i][j]==1:
                print("DANGER")
                ccc+=1
        c+=1
if c == X_FULL.shape[0]*X_FULL.shape[1]:
    print(c)
    print("NUM CHECK OK")
print(cc)
if ccc == 0:
    print("NO Illegal changes")
```

```
46050708
NUM CHECK OK
46144
NO Illegal changes
```

3 Split features into ones that contribute maliciousness or benignness

[6]:
```
mal_col_index = []
ben_col_index = []
for i in range(len(coeff)):
    if coeff[i] > 0:
        mal_col_index.append(i)
    elif coeff[i] < 0:
        ben_col_index.append(i)
    else:
        print("DANGER")
print(len(mal_col_index),len(ben_col_index))
```

```
1513 1435
```

4 Find attacked Rows

[9]:
```
attacked_indexes = [i for i in range(len(X_FULL)) if X_FULL[i].tolist()!
 ↪=X_N_FULL[i].tolist()]
len(attacked_indexes)
```

[9]:
```
5018
```

[16]:
```
attacked_indexes[:10]
```

[16]:
```
[2, 3, 6, 15, 17, 18, 22, 29, 30, 33]
```

5 Missclassification

[14]:
```
np.load('./DATA/JSMA/x_test_mal_noisy_idx_JSMA_AZ_2948.npy').shape
```

[14]:
```
(5018,)
```

[12]:
```
len([i for i in range(len(Y_FULL)) if Y_FULL[i]==1])
```

[12]:
```
5207
```

[15]:
```
5018/5207
```

[15]:
```
0.9637027078932207
```

5.0.1 JSMA missclassifies 5018 of the 5207 malware APKs as Benign, or a 96% misscalssification

- Missclassification(good) = 96%

6 Further Analysis

[18]:
```
X = X_FULL[attacked_indexes].copy()
X_N = X_N_FULL[attacked_indexes].copy()
```

[19]:
```
attack_changes = []
for i in range(X.shape[0]):
    attack_changes.append([c for c in range(len(X[i]))
                                if X[i][c] != X_N[i][c]])
len(attack_changes)
```

[19]: 5018

[20]:
```
attack_MBA = []
for i in attack_changes:
    m = b = 0
    for j in range(len(i)):
        if i[j] in mal_col_index:
            m+=1
        else:
            b+=1
    attack_MBA.append([b,m])
attack_MBA = np.array(attack_MBA)
attack_MBA.shape
```

[20]: (5018, 2)

[22]:
```
np.mean(attack_MBA,axis=0)
```

[22]: array([5.76365086, 3.43204464])

7 JSMA on average adds 6 Benign and 3 Malware Features for each successfull attack

[52]:
```
totalDump = []
for i in range(len(attack_changes)):
    for j in range(len(attack_changes[i])):
        totalDump.append(attack_changes[i][j])
len(totalDump)
```

[52]: 46144

7.0.1 Across the 5018 samples, 46144 bits were changes

[55]:
```
A = np.unique(np.asarray(totalDump),return_counts=True)
```

[59]:
```
AX = np.array(A)
```

[60]:
```
AX.shape
```

[60]:
```
(2, 596)
```

8 JSMA changes 596 features only out of the almost 3000 throughout the attack

[77]:
```
colNum = []
timesChanged = []
for i in range(len(AX[0])):
    colNum.append(tuple(zip(*AX))[i][0])
    timesChanged.append(tuple(zip(*AX))[i][1])
```

[87]:
```
c = 0
a = []
colNumOver100 = []
timesChangedOver100 = []
for i in range(len(colNum)):
    if timesChanged[i] > 100:
        z = 'feature '+str(colNum[i])+" was changed in " + str(timesChanged[i])+" rows"
        a.append(z)
        c+=1
        colNumOver100.append(colNum[i])
        timesChangedOver100.append(timesChanged[i])
c
```

[87]:
```
80
```

[88]:
```
a
```

[88]:
```
['feature 0 was changed in 239 rows',
 'feature 32 was changed in 526 rows',
 'feature 60 was changed in 243 rows',
 'feature 61 was changed in 382 rows',
 'feature 97 was changed in 643 rows',
 'feature 101 was changed in 112 rows',
 'feature 112 was changed in 149 rows',
 'feature 133 was changed in 109 rows',
 'feature 244 was changed in 130 rows',
```

```
'feature 279 was changed in 189 rows',
'feature 321 was changed in 104 rows',
'feature 344 was changed in 638 rows',
'feature 359 was changed in 2318 rows',
'feature 360 was changed in 472 rows',
'feature 361 was changed in 134 rows',
'feature 364 was changed in 776 rows',
'feature 393 was changed in 119 rows',
'feature 427 was changed in 296 rows',
'feature 445 was changed in 281 rows',
'feature 449 was changed in 237 rows',
'feature 470 was changed in 604 rows',
'feature 749 was changed in 109 rows',
'feature 751 was changed in 289 rows',
'feature 861 was changed in 715 rows',
'feature 910 was changed in 252 rows',
'feature 911 was changed in 445 rows',
'feature 914 was changed in 374 rows',
'feature 922 was changed in 276 rows',
'feature 938 was changed in 318 rows',
'feature 1051 was changed in 958 rows',
'feature 1070 was changed in 465 rows',
'feature 1113 was changed in 225 rows',
'feature 1127 was changed in 194 rows',
'feature 1198 was changed in 270 rows',
'feature 1349 was changed in 1058 rows',
'feature 1662 was changed in 283 rows',
'feature 1666 was changed in 570 rows',
'feature 1985 was changed in 139 rows',
'feature 2147 was changed in 169 rows',
'feature 2185 was changed in 130 rows',
'feature 2187 was changed in 511 rows',
'feature 2279 was changed in 210 rows',
'feature 2315 was changed in 1146 rows',
'feature 2337 was changed in 153 rows',
'feature 2354 was changed in 101 rows',
'feature 2358 was changed in 681 rows',
'feature 2386 was changed in 337 rows',
'feature 2397 was changed in 120 rows',
'feature 2398 was changed in 163 rows',
'feature 2402 was changed in 329 rows',
'feature 2456 was changed in 367 rows',
'feature 2479 was changed in 286 rows',
'feature 2488 was changed in 189 rows',
'feature 2580 was changed in 627 rows',
'feature 2590 was changed in 1235 rows',
'feature 2592 was changed in 112 rows',
```

```
 'feature 2605 was changed in 1176 rows',
 'feature 2651 was changed in 221 rows',
 'feature 2657 was changed in 107 rows',
 'feature 2665 was changed in 128 rows',
 'feature 2681 was changed in 3272 rows',
 'feature 2711 was changed in 2054 rows',
 'feature 2720 was changed in 184 rows',
 'feature 2722 was changed in 199 rows',
 'feature 2725 was changed in 280 rows',
 'feature 2731 was changed in 783 rows',
 'feature 2737 was changed in 103 rows',
 'feature 2793 was changed in 371 rows',
 'feature 2819 was changed in 122 rows',
 'feature 2829 was changed in 101 rows',
 'feature 2836 was changed in 767 rows',
 'feature 2843 was changed in 551 rows',
 'feature 2847 was changed in 992 rows',
 'feature 2861 was changed in 316 rows',
 'feature 2898 was changed in 731 rows',
 'feature 2926 was changed in 581 rows',
 'feature 2935 was changed in 396 rows',
 'feature 2936 was changed in 378 rows',
 'feature 2946 was changed in 1989 rows',
 'feature 2947 was changed in 1474 rows']
```

[89]:
```
m = b = 0
for i in colNumOver100:
    if i in mal_col_index:
        m+=1
    else:
        b+=1
print(b,m)
```

```
45 35
```

9 80 features were changed over a 100 times in JSMA out of which 45 are benign features and 35 are malware features

[ ]:
```

```

FGSM JSMA Comparisions

October 30, 2020

1 JSMA

1.1 Obseravtions

- higly effective
- extremely time consuming
- does attack for each row so higher chance of attack success
- lot more stable in results, as in running again will give similar result
- 96% of the malware was missclassified
- total of 46144 bits were changed across all 500 odd rows and 3000 odd cols, which is pretty small but effective changes. Only around 0.0032% of the dataset was chaned for a 96% success
- For every successfull
  - JSMA adds 6 Ben and 3 Mal features (average)
- But in total, across all 5018 samples it attacked properly, it touched only 596 features
- in this 596, 80 features was changed atleast a 100 times
  - in this 80 features, 45 are Ben and 35 are Mal

2 FGSM

2.1 Observations

- kinda effective
- Super fast
- calculate across total loss and attacks. Much faster than focusing on each row
- highly unstable, every run can yeild anywhere from 90% to 10% missclassification
- There were many successfull attacks(changed malware into ben) and a few unwanted bad cases(changed ben into mal) -Seeing separately based on outcome of attack
  - Good Attacks (3987) => Malware into Ben
    - For Classifying a Malware into a Benign, The FGSM approximately adds around 997 extra Benign Features and 883 extra malware features in the APKs
  - Bad Attacks (237) => Ben into Mal
    - For a bad attack, that is changing a ben into a mal classification, the FGSM added approx 582 Ben cols and 764 mal cols
  - Attacks that didn't work (1073)
    - in the no change category, FGSM added approx 705 ben and 868 mal cols.
- Seeing the whole picture
  - FGSM adds an approx 918 ben cols and 874 mal cols to a row while attacking
  - When it adds more ben that mal features, then it missclassifies. It does come under this case a lot but there are a significant number of cases where it adds more malicious

features to sometimes even makes a benign classification as malicious as demonstrated above

- Missclassification
  - raw missclassification: 0.79743250896734
  - Accepted missclassification: 0.7526902020011327
  - Bad Missclassification: 0.04474230696620729
  - observed change: 0.7079478950349254

[ ]: