Untitled18

June 26, 2019

3. Exploratory Data Analysis

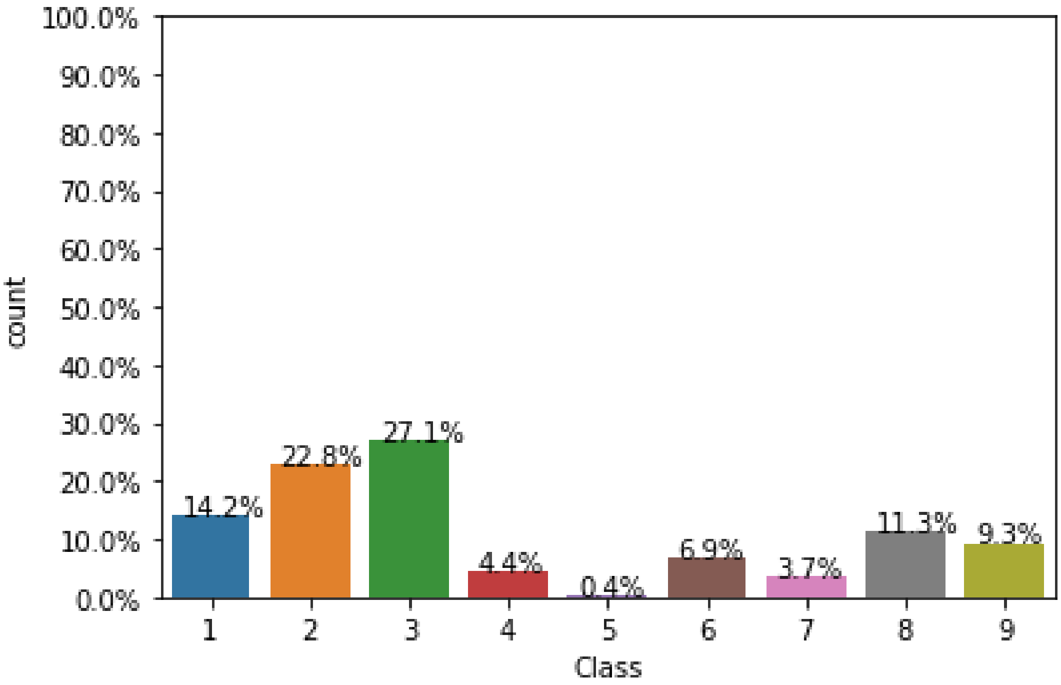
```
In [0]: import warnings
        warnings.filterwarnings("ignore")
        import shutil
        import scipy
        import os
        import pandas as pd
        import matplotlib
        matplotlib.use(u'nbAgg')
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import pickle
        import xgboost
        from sklearn.manifold import TSNE
        from sklearn import preprocessing
        import pandas as pd
        from multiprocessing import Process# this is used for multithreading
        import multiprocessing
        import codecs# this is used for file operations
        import random as r
        from xgboost import XGBClassifier
        from sklearn.model_selection import RandomizedSearchCV
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.calibration import CalibratedClassifierCV
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import log_loss
        from sklearn.metrics import confusion_matrix
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestClassifier
        from tqdm import tqdm
        from sklearn.feature_extraction.text import CountVectorizer
```

```
In [0]: #separating byte files and asm files

        source = 'train'
```

```
destination = 'byteFiles'

# we will check if the folder 'byteFiles' exists if it not there we will create a fold
if not os.path.isdir(destination):
    os.makedirs(destination)

# if we have folder called 'train' (train folder contains both .asm files and .bytes f
# for every file that we have in our 'asmFiles' directory we check if it is ending wit
# 'byteFiles' folder

# so by the end of this snippet we will separate all the .byte files and .asm files
if os.path.isdir(source):
    os.rename(source,'asmFiles')
    source='asmFiles'
    data_files = os.listdir(source)
    for file in data_files:
        if (file.endswith("bytes")):
            shutil.move('asmFiles/'+file,destination)
```

3.1. Distribution of malware classes in whole data set

```
In [0]: %matplotlib inline
        Y=pd.read_csv("trainLabels.csv")
        total = len(Y)*1.
        ax=sns.countplot(x="Class", data=Y)
        for p in ax.patches:
                ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()+0.1, p.get_

        #put 11 ticks (therefore 10 steps), from 0 to the total number of rows in the datafram
        ax.yaxis.set_ticks(np.linspace(0, total, 11))

        #adjust the ticklabel to the desired format, without changing the position of the tick
        ax.set_yticklabels(map('{:.1f}%'.format, 100*ax.yaxis.get_majorticklocs()/total))
        plt.show()
```



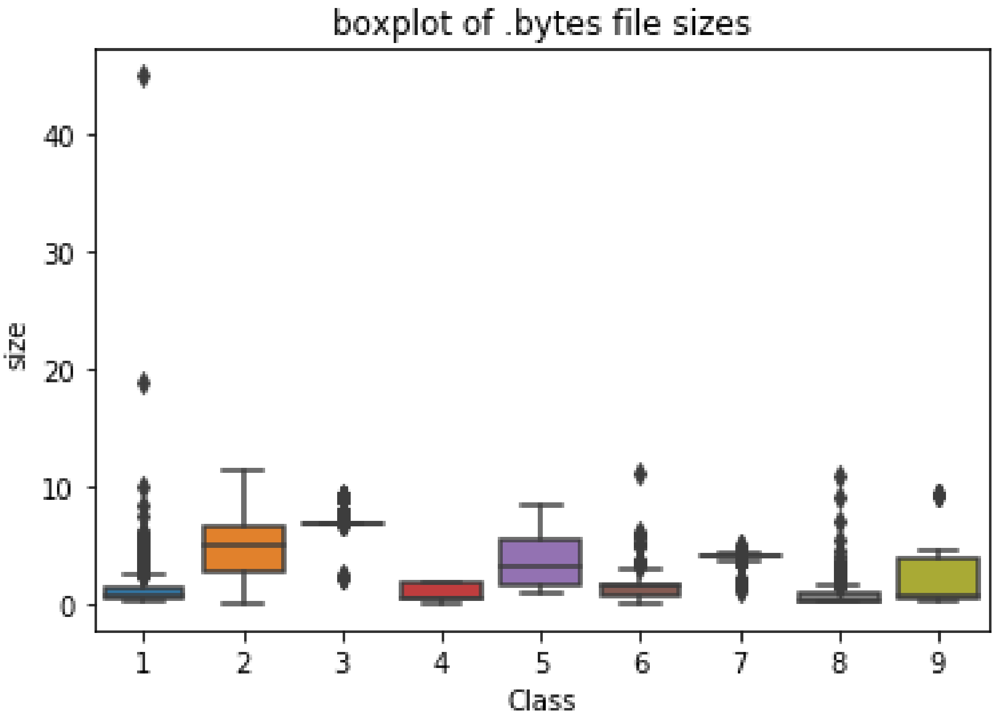
3.2. Feature extraction

```
In [0]: #file sizes of byte files

        files=os.listdir('byteFiles')
        filenames=Y['Id'].tolist()
        class_y=Y['Class'].tolist()
        class_bytes=[]
        sizebytes=[]
        fnames=[]
        for file in files:
            # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
            # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nli
            # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
            # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
            statinfo=os.stat('byteFiles/'+file)
            # split the file name at '.' and take the first part of it i.e the file name
            file=file.split('.')[0]
            if any(file == filename for filename in filenames):
                i=filenames.index(file)
                class_bytes.append(class_y[i])
                # converting into Mb's
                sizebytes.append(statinfo.st_size/(1024.0*1024.0))
                fnames.append(file)
        data_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
        print (data_size_byte.head())
```

```
                     ID      size  Class
0  HJQiyIRqr6FPeBcoaEsk  6.842773      3
1  G8hm6UqIKBQWlMpeTScb  0.801514      1
2  6mUHQtCBjzWA0fGIEnP7  7.596436      2
3  9gMZ6wVFX7KvHN3y8LoG  7.285400      2
4  hqzvHQ4UBkTPinujM1RC  2.308838      6
```

3.2.2 box plots of file size (.byte files) feature

```
In [0]: #boxplot of byte files
        ax = sns.boxplot(x="Class", y="size", data=data_size_byte)
        plt.title("boxplot of .bytes file sizes")
        plt.show()
```



3.2.3 feature extraction from byte files

```
In [0]: #removal of addres from byte files
        # contents of .byte files
        # ----------------
        #00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
        #-------------------
        #we remove the starting address 00401000
        #removal of addres from byte files
        # contents of .byte files
# ----------------
#00401000 56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
#-------------------
#we remove the starting address 00401000
import datetime
start = datetime.datetime.now()
files = os.listdir('byteFiles')
filenames=[]
array=[]
for f in files:
    if(f.endswith("bytes")):
        file=f.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        with open('byteFiles/'+file+".bytes" ,"r") as fp:
            lines=""
            for line in fp:
                a=line.rstrip().split(" ")[1:]
                b=' ' + ' '.join(a)
                b=b+"\n"
                text_file.write(b)
            fp.close()
            os.remove('byteFiles/'+file+".bytes")
        text_file.close()
print("Time required to run this cell:", datetime.datetime.now() - start)



print('done!')
```

In [0]:

```
#program to convert into bag of words of bytefiles
#this is custom-built bag of words this is unigram bag of words
byte_feature_file=open('result.csv','w+')
byte_feature_file.write("ID,0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16
for file in files:
    filenames2.append(f)
    byte_feature_file.write(file+",")
    if(file.endswith("txt")):
        with open('byteFiles/'+file,"r") as byte_flie:
            for lines in byte_flie:
                line=lines.rstrip().split(" ")
                for hex_code in line:
                    if hex_code=='??':
                        feature_matrix[k][256]+=1
                    else:
                        feature_matrix[k][int(hex_code,16)]+=1
        byte_flie.close()
    for i in feature_matrix[k]:
        byte_feature_file.write(str(i)+",")
```

```
            byte_feature_file.write("\n")

            k += 1

        byte_feature_file.close()
```

```
In [0]: byte_features=pd.read_csv("result_with_size.csv")
        print (byte_features.head())
```

```
   Unnamed: 0                    ID       0     1     2     3     4     5  \
0           0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242   
1           1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844   
2           2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330   
3           3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   
4           4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   

      6     7  ...    f9    fa    fb    fc    fd     fe     ff     ??  \
0  3650  3201  ...  3101  3211  3097  2758  3099   2759   5753   1824   
1  8420  7589  ...   439   281   302  7639   518  17001  54902   8588   
2  9007  2342  ...  2242  2885  2863  2471  2786   2680  49144    468   
3   550   523  ...   485   462   516  1133   471    761   7998  13940   
4   262   249  ...   350   209   239   653   221    242   2199   9008   

       size  Class  
0  4.234863      9  
1  5.538818      2  
2  3.887939      9  
3  0.574219      1  
4  0.370850      8  

[5 rows x 261 columns]
```

```
In [0]: data_size_byte.columns

Out[0]: Index(['ID', 'size', 'Class'], dtype='object')
```

```
In [0]: result = pd.merge(byte_features, data_size_byte, on='ID', how='left')
        result.head()
```

```
Out[0]:    Unnamed: 0                    ID       0     1     2     3     4     5  \
        0           0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242   
        1           1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844   
        2           2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330   
        3           3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   
        4           4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   

              6     7  ...    fb    fc    fd     fe     ff     ??    size_x  Class_x  \
        0  3650  3201  ...  3097  2758  3099   2759   5753   1824  4.234863        9   
```

```
1  8420  7589  ...   302  7639   518  17001  54902   8588  5.538818        2
2  9007  2342  ...  2863  2471  2786   2680  49144    468  3.887939        9
3   550   523  ...   516  1133   471    761   7998  13940  0.574219        1
4   262   249  ...   239   653   221    242   2199   9008  0.370850        8

     size_y  Class_y
0  4.234863        9
1  5.538818        2
2  3.887939        9
3  0.574219        1
4  0.370850        8

[5 rows x 263 columns]
```

```
In [0]: # result = result.drop(["Class_y", "size_x", "size_y"], axis = 1)
```

```
In [0]: # data = data.rename(columns={"Area": "place_name"})
        # result = result.rename(columns={"Class_x": "Class"})
```

```
In [0]: # https://stackoverflow.com/a/29651514
        def normalize(df):
            result1 = df.copy()
            for feature_name in df.columns:
                if (str(feature_name) != str('ID') and str(feature_name)!=str('Class')):
                    max_value = df[feature_name].max()
                    min_value = df[feature_name].min()
                    result1[feature_name] = (df[feature_name] - min_value) / (max_value - min_
            return result1
        result = normalize(result)
```

```
In [0]: result=byte_features
```

```
In [0]: data_y = result['Class']
        # split the data into test and train by maintaining same distribution of output varaib
        X_train, X_test, y_train, y_test = train_test_split(result.drop(['ID','Class'], axis=1
        # split the train data into train and cross validation by maintaining same distributio
        X_train, X_cv, y_train, y_cv = train_test_split(X_train, y_train,stratify=y_train,test_
```

```
In [0]: dfasm=pd.read_csv("asmoutputfile.csv")
        Y.columns = ['ID', 'Class']
        result_asm = pd.merge(dfasm, Y,on='ID', how='left')
        result_asm.head()
```

```
Out[0]:                       ID  HEADER:  .text:  .Pav:  .idata:  .data:  .bss:  \
        0  01kcPWA9K2BOxQeS5Rju       19     744      0      127      57      0
        1  1E93CpP6ORHFNiT5Qfvn       17     838      0      103      49      0
        2  3ekVow2ajZHbTnBcsDfX       17     427      0       50      43      0
        3  3X2nY7iQaPBIWDrAZqJe       17     227      0       43      19      0
        4  46OZzdsSKDCFV8h7XWxf       17     402      0       59     170      0
```

```
   .rdata:  .edata:  .rsrc:  ...  edx  esi  eax  ebx  ecx  edi  ebp  esp  eip  \
0      323        0       3  ...   18   66   15   43   83    0   17   48   29   
1        0        0       3  ...   18   29   48   82   12    0   14    0   20   
2      145        0       3  ...   13   42   10   67   14    0   11    0    9   
3        0        0       3  ...    6    8   14    7    2    0    8    0    6   
4        0        0       3  ...   12    9   18   29    5    0   11    0   11   

   Class  
0      1  
1      1  
2      1  
3      1  
4      1  

[5 rows x 53 columns]
```

```
In [0]: #file sizes of byte files

        files=os.listdir('asmFiles')
        filenames=Y['ID'].tolist()
        class_y=Y['Class'].tolist()
        class_bytes=[]
        sizebytes=[]
        fnames=[]
        for file in files:
            # print(os.stat('byteFiles/0A32eTdBKayjCWhZqDOQ.txt'))
            # os.stat_result(st_mode=33206, st_ino=1125899906874507, st_dev=3561571700, st_nli
            # st_size=3680109, st_atime=1519638522, st_mtime=1519638522, st_ctime=1519638522)
            # read more about os.stat: here https://www.tutorialspoint.com/python/os_stat.htm
            statinfo=os.stat('asmFiles/'+file)
            # split the file name at '.' and take the first part of it i.e the file name
            file=file.split('.')[0]
            if any(file == filename for filename in filenames):
                i=filenames.index(file)
                class_bytes.append(class_y[i])
                # converting into Mb's
                sizebytes.append(statinfo.st_size/(1024.0*1024.0))
                fnames.append(file)
        asm_size_byte=pd.DataFrame({'ID':fnames,'size':sizebytes,'Class':class_bytes})
        print (asm_size_byte.head())
```

```
                     ID       size  Class
0  C9hHuINUVJqk1zo5pTQX   8.265899      1
1  16cTMtKIjH5SbyovWuBq  36.532800      2
2  0Phtp2LVcsCFMKkGgmRH  58.857293      9
3  4Z17fDwSIlGLtyoPv6Fp   0.172276      3
4  c8BfFP6iYEIRaUxdGtmX  37.294192      2
```

```
In [0]: # add the file size feature to previous extracted features
        print(result_asm.shape)
        print(asm_size_byte.shape)
        result_asm = pd.merge(result_asm, asm_size_byte.drop(['Class'], axis=1),on='ID', how='
        result_asm.head()
```

```
(10868, 53)
(10868, 3)
```

```
Out[0]:                       ID  HEADER:  .text:  .Pav:  .idata:  .data:  .bss:  \
        0  01kcPWA9K2BOxQeS5Rju       19     744      0      127      57      0
        1  1E93CpP6ORHFNiT5Qfvn       17     838      0      103      49      0
        2  3ekVow2ajZHbTnBcsDfX       17     427      0       50      43      0
        3  3X2nY7iQaPBIWDrAZqJe       17     227      0       43      19      0
        4  46OZzdsSKDCFV8h7XWxf       17     402      0       59     170      0

           .rdata:  .edata:  .rsrc:  ...  esi  eax  ebx  ecx  edi  ebp  esp  eip  \
        0      323        0       3  ...   66   15   43   83    0   17   48   29
        1        0        0       3  ...   29   48   82   12    0   14    0   20
        2      145        0       3  ...   42   10   67   14    0   11    0    9
        3        0        0       3  ...    8   14    7    2    0    8    0    6
        4        0        0       3  ...    9   18   29    5    0   11    0   11

           Class      size
        0      1  0.078190
        1      1  0.063400
        2      1  0.041695
        3      1  0.018757
        4      1  0.037567

        [5 rows x 54 columns]
```

```
In [0]: asm_y = result_asm['Class']
        asm_x = result_asm.drop(['ID','Class','.BSS:','rtn','.CODE'], axis=1)
```

```
In [0]: X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(asm_x,asm_y ,strati
        X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_as
```

```
In [0]: result_x = pd.merge(result,result_asm.drop(['Class'], axis=1),on='ID', how='left')
        result_y = result_x['Class']
        result_x = result_x.drop(['ID','rtn','.BSS:','.CODE','Class'], axis=1)
        result_x.head()
```

```
Out[0]:    Unnamed: 0       0     1     2     3     4     5     6     7     8  ...  \
        0           0  601905  3905  2816  3832  3345  3242  3650  3201  2965  ...
        1           1   39755  8337  7249  7186  8663  6844  8420  7589  9291  ...
        2           2   93506  9542  2568  2438  8925  9330  9007  2342  9107  ...
        3           3   21091  1213   726   817  1257   625   550   523  1078  ...
```

```
4           4   19764   710   302   433   559   410   262   249   422  ...

   edx   esi   eax   ebx  ecx  edi  ebp  esp  eip     size_y
0  808  2290  1281   587  701    0   15   14  456  56.229886
1  260  1090   391   905  420    0   24   22  227  13.999378
2    5   547     5   451   56    0   27    0  117   8.507785
3   18    66    15    43   83    0   17   48   29   0.078190
4   18  1228    24  1546  107    0   15    0   76   0.996723

[5 rows x 308 columns]
```

```
In [0]: result_y.head()
```

```
Out[0]: 0    9
        1    2
        2    9
        3    1
        4    8
        Name: Class, dtype: int64
```

```
In [0]: X_train, X_test_merge, y_train, y_test_merge = train_test_split(result_x, result_y,stra
        X_train_merge, X_cv_merge, y_train_merge, y_cv_merge = train_test_split(X_train, y_trai
```

```
In [0]: result_x['ID'] = result.ID
```

0.1 Bi-grams

```
In [0]: byte_vocab = "00,01,02,03,04,05,06,07,08,09,0a,0b,0c,0d,0e,0f,10,11,12,13,14,15,16,17,
```

```
In [0]: byte_bigram_vocab = []

        def byte_bigram():
            for i, v in enumerate(byte_vocab.split(',')):
                for j in range(0, len(byte_vocab.split(','))):
                    byte_bigram_vocab.append(v + ' ' +byte_vocab.split(',')[j])
            len(byte_bigram_vocab)
```

```
In [0]: byte_bigram()
```

```
In [0]: byte_bigram_vocab[:5]
```

```
Out[0]: ['00 00', '00 01', '00 02', '00 03', '00 04']
```

```
In [0]: len(byte_bigram_vocab)
```

```
Out[0]: 66049
```

```
In [0]: byte_trigram_vocab = []

        def byte_trigram():
                    for i, v in enumerate(byte_vocab.split(',')):
                        for j in range(0, len(byte_vocab.split(','))):
                            for k in range(0, len(byte_vocab.split(','))):
                                byte_trigram_vocab.append(v + ' ' +byte_vocab.split(',')[j]+' '+byte_v
                    len(byte_trigram_vocab)
```

```
In [0]: byte_trigram()
```

```
In [0]: import pickle

        filename = 'trigram'
        outfile = open(filename,'wb')

        pickle.dump(byte_trigram_vocab,outfile)
        outfile.close()
```

```
In [0]: infile = open('trigram','rb')
        byte_trigram_vocab = pickle.load(infile)
        infile.close()
```

```
In [0]: byte_trigram_vocab[:5]
```

```
Out[0]: ['00 00 00', '00 00 01', '00 00 02', '00 00 03', '00 00 04']
```

```
In [0]: len(byte_trigram_vocab)
```

```
Out[0]: 16974593
```

```
In [0]: from tqdm import tqdm
        from sklearn.feature_extraction.text import CountVectorizer



        vect = CountVectorizer(lowercase=False,ngram_range=(2,2), vocabulary=byte_bigram_vocab
        byte_bigram_vect = scipy.sparse.csr_matrix((10868, 66049))
        for i, file in tqdm(enumerate(os.listdir('./byteFiles'))):
            f = open('./byteFiles/' + file)
            a[i : ]+= scipy.sparse.csr_matrix(vect.fit_transform([f.read().replace('\n', ' ').
            f.close()

        scipy.sparse.save_npz('bytebigram.npz', byte_bigram_vect)
```

```
0it [00:00, ?it/s]

1it [00:01,  1.01s/it]

2it [00:01,  1.28it/s]
```

```
3it [00:03,  1.15s/it]

4it [00:05,  1.40s/it]

5it [00:05,  1.19s/it]

6it [00:06,  1.11s/it]

7it [00:06,  1.23it/s]

8it [00:08,  1.02it/s]

9it [00:08,  1.38it/s]

10it [00:10,  1.10s/it]

11it [00:12,  1.27s/it]

12it [00:14,  1.45s/it]

13it [00:15,  1.51s/it]

14it [00:16,  1.22s/it]

15it [00:16,  1.09it/s]

16it [00:17,  1.09it/s]

17it [00:17,  1.38it/s]

18it [00:17,  1.70it/s]

19it [00:19,  1.30it/s]

20it [00:20,  1.20it/s]

21it [00:20,  1.61it/s]
```

```
In [0]: scipy.sparse.save_npz('bytebigram.npz', byte_bigram_vect)
```

```
In [0]: import scipy
        from sklearn.preprocessing import normalize
        byte_bigram_vect = normalize(scipy.sparse.load_npz('bytebigram.npz'), axis = 0)
```

1 N-Grams

```
In [0]: #Ref https://www.edwardraff.com/publications/what_can_ngrams_learn.pdf
        #Ref https://github.com/melanieihuei/Malware-Classification

        opcodes = ['jmp', 'mov', 'retf', 'push', 'pop', 'xor', 'retn', 'nop', 'sub', 'inc', 'de
```

```
In [0]: asm_bigram = []
        def asmopcodebigram():
            for i, v in enumerate(opcodes):
                for j in range(0, len(opcodes)):
                    asm_bigram.append(v + ' ' + opcodes[j])
```

```
In [0]: asmopcodebigram()
        len(asm_bigram)
```

```
Out[0]: 676
```

```
In [0]: asm_trigram = []
        def asmopcodetrigram():
            for i, v in enumerate(opcodes):
                for j in range(0, len(opcodes)):
                    for k in range(0, len(opcodes)):
                        asm_trigram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k])
```

```
In [0]: asmopcodetrigram()
        len(asm_trigram)
```

```
Out[0]: 17576
```

```
In [0]: asm_4gram = []
        for i, v in enumerate(opcodes):
            for j in range(0, len(opcodes)):
                for k in range(0, len(opcodes)):
                    for l in range(0, len(opcodes)):
                        asm_4gram.append(v + ' ' + opcodes[j] + ' ' + opcodes[k] + ' ' + opcode
        len(asm_4gram)
```

```
Out[0]: 456976
```

```
In [0]: def opcode_collect():
            op_file = open("opcode_file.txt", "w+")
            for asmfile in os.listdir('asmFiles'):
                opcode_str = ""
                with codecs.open('asmFiles/' + asmfile, encoding='cp1252', errors ='replace') a
                    for lines in fli:
                        line = lines.rstrip().split()
                        for li in line:
                            if li in opcodes:
```

```
                        opcode_str += li + ' '
        op_file.write(opcode_str + "\n")
    op_file.close()
opcode_collect()
```

```
In [0]: vect = CountVectorizer(ngram_range=(2, 2), vocabulary = asm_bigram)
        bigram_vect = scipy.sparse.csr_matrix((10868, len(asm_bigram)))
        raw_opcode = open('opcode_file.txt').read().split('\n')

        for i in range(10868):
            bigram_vect[i, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[i]]))
```

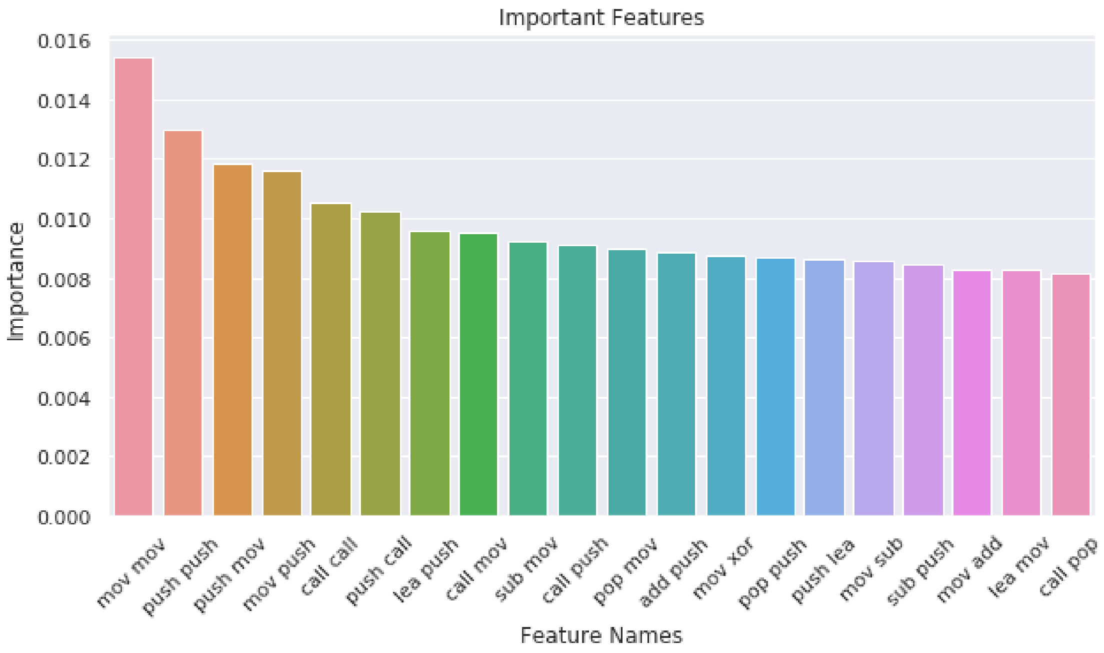
```
In [0]: scipy.sparse.save_npz('op_bigram.npz', opcodebivect)
```

```
In [0]: vect = CountVectorizer(ngram_range=(3, 3), vocabulary = asm_trigram)
        trigram_vect = scipy.sparse.csr_matrix((10868, len(asm_trigram)))
        raw_opcode = open('opcode_file.txt').read().split('\n')

        for i in range(10868):
            trigram_vect[i, :] += scipy.sparse.csr_matrix(vect.transform([raw_opcode[i]]))
```

```
In [0]: scipy.sparse.save_npz('op_trigram.npz', opcodetrivect)
```

```
In [0]: opcodebivect=scipy.sparse.load_npz('op_bigram.npz')
        opcodetrivect=scipy.sparse.load_npz('op_trigram.npz')
```

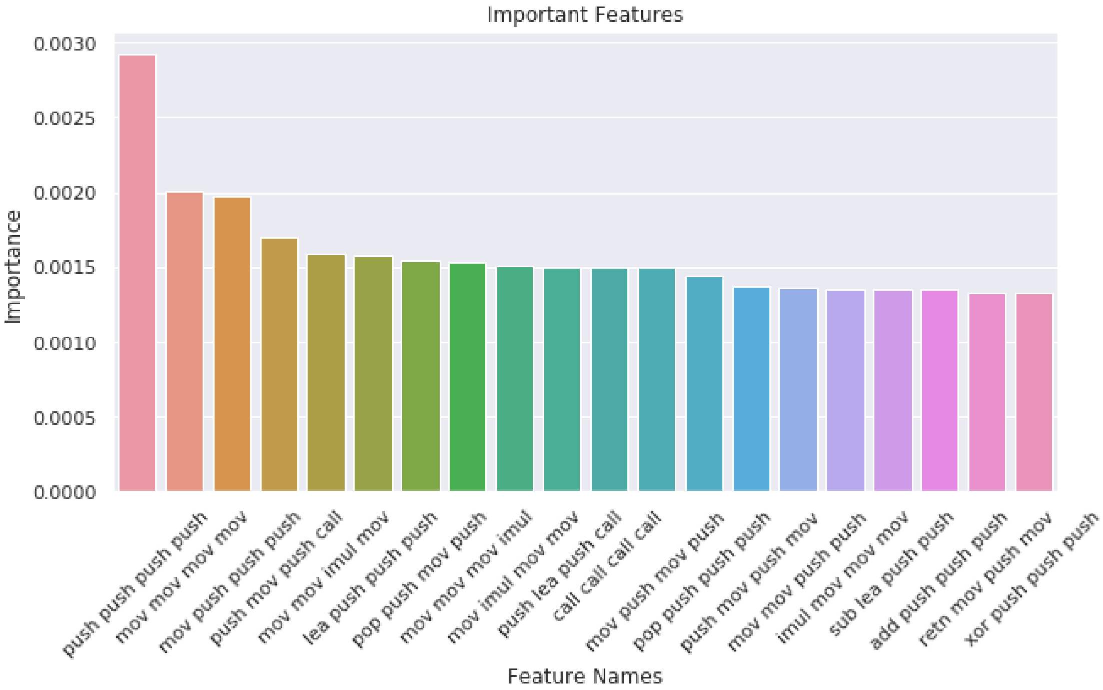
```
In [0]: bi_imp_feat = imp_features(normalize(bigram_vect, axis = 0), asm_bigram, 200)
```



```
In [0]: op_bigram = pd.read_csv('op_bigram.csv').drop('Unnamed: 0', axis = 1).fillna(0)
        op_trigram = pd.read_csv('op_trigram.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

```
In [0]: opcode_tetra_vect = scipy.sparse.load_npz('opcode_tetragram.npz')
```

```
In [0]: n_imp_feat = imp_features(normalize(n_vect, axis = 0), asm_4gram, 5000)
```



```
In [0]: op_tetragram = pd.SparseDataFrame(normalize(n_vect, axis = 0), columns = asm_4gram)
        op_tetragram = op_tetragram.loc[:, np.intersect1d(op_tetragram.columns, np.take(asm_4g
```

```
In [0]: op_tetra_df.to_dense().to_csv('op_tetra_filtered.csv')
```

```
In [0]: op_tetragram = pd.read_csv('op_n_final.csv').drop('Unnamed: 0', axis = 1).fillna(0)
```

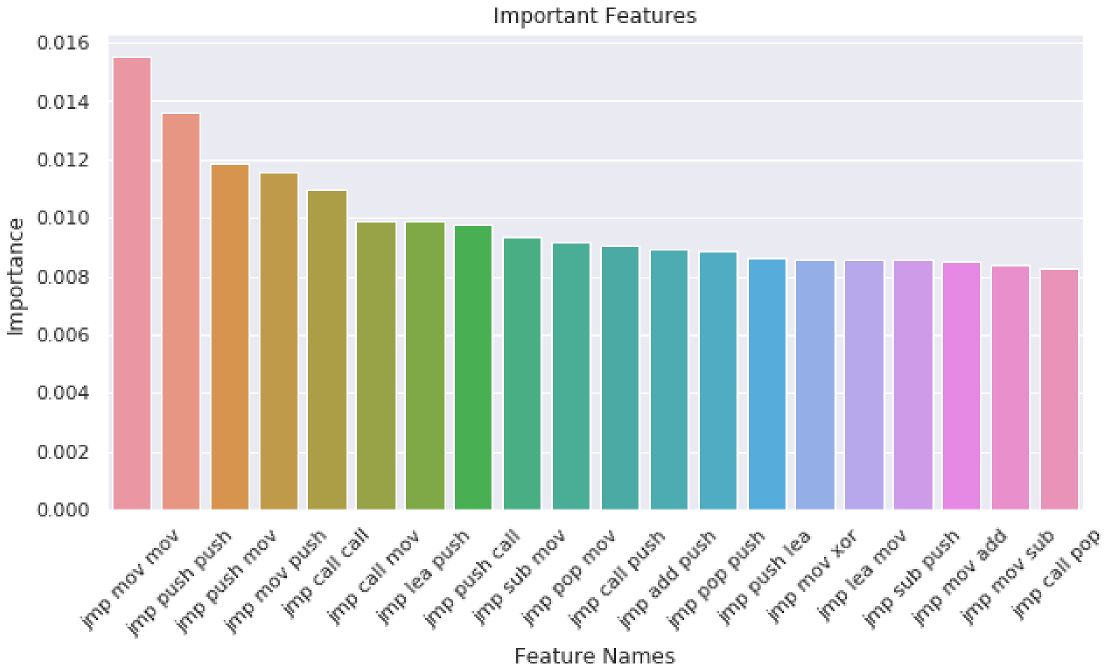
```
In [0]: op_tetragram['ID'] = result.ID
        op_tetragram.head()
```

```
Out[0]:    add add add add  add add add cmp  add add add dec  add add add jmp  \
        0         0.000443         0.024936              0.0              0.0
        1         0.000000         0.000000              0.0              0.0
        2         0.000000         0.000000              0.0              0.0
        3         0.000000         0.000000              0.0              0.0
        4         0.000000         0.000000              0.0              0.0

           add add add jz  add add add mov  add add add or  add add add pop  \
0             0.0        0.003054             0.0              0.0
1             0.0        0.000000             0.0              0.0
2             0.0        0.000000             0.0              0.0
3             0.0        0.000000             0.0              0.0
4             0.0        0.000000             0.0              0.0

   add add add push  add add add retn  ...  xor xor push push  \
0          0.008047               0.0  ...                0.0
1          0.000000               0.0  ...                0.0
2          0.000000               0.0  ...                0.0
3          0.000000               0.0  ...                0.0
4          0.000000               0.0  ...                0.0

   xor xor push sub  xor xor push xor  xor xor sub mov  xor xor sub push  \
0               0.0               0.0              0.0               0.0
1               0.0               0.0              0.0               0.0
2               0.0               0.0              0.0               0.0
3               0.0               0.0              0.0               0.0
4               0.0               0.0              0.0               0.0

   xor xor xchg mov  xor xor xor mov  xor xor xor sub  xor xor xor xor  \
0               0.0         0.000216              0.0          0.00111
1               0.0         0.000000              0.0          0.00000
2               0.0         0.000000              0.0          0.00000
3               0.0         0.000000              0.0          0.00000
4               0.0         0.000000              0.0          0.00000

                     ID
0  01azqd4InC7m9JpocGv5
1  01IsoiSMh5gxyDYTl4CB
2  01jsnpXSAlgw6aPeDxrU
3  01kcPWA9K2BOxQeS5Rju
4  01SuzwMJEIXsK7A8dQbl

[5 rows x 5001 columns]
```

```
In [0]: def imp_features(data, features, keep):
            rf = RandomForestClassifier(n_estimators = 100, n_jobs = -1)
            rf.fit(data, result_y)
            imp_feature_indx = np.argsort(rf.feature_importances_)[::-1]
            imp_value = np.take(rf.feature_importances_, imp_feature_indx[:20])
            imp_feature_name = np.take(features, imp_feature_indx[:20])
            sns.set()
            plt.figure(figsize = (10, 5))
            ax = sns.barplot(x = imp_feature_name, y = imp_value)
            ax.set_xticklabels(labels = imp_feature_name, rotation = 45)
            sns.set_palette(reversed(sns.color_palette("husl", 10)), 10)
            plt.title('Important Features')
```

```
    plt.xlabel('Feature Names')
    plt.ylabel('Importance')
    return imp_feature_indx[:keep]
```

```
In [0]: from sklearn.preprocessing import normalize

        tri_imp_feat = imp_features(normalize(bigram_vect, axis = 0), asm_trigram, 300)
```



```
In [0]: op_bigram = pd.SparseDataFrame(normalize(bigram_vect, axis = 0), columns = asm_bigram)
        for col in op_bigram.columns:
            if col not in np.take(asm_bigram, bi_imp_feat):
                op_bigram.drop(col, axis = 1, inplace = True)
```

```
In [0]: op_bigram = pd.read_csv('op_bigram_final.csv').drop('Unnamed: 0', axis = 1).fillna(0)
        op_bigram['ID'] = result.ID
        op_bigram.head()
```

```
Out[0]:     jmp jmp   jmp mov  jmp push   jmp pop   jmp xor   jmp sub   jmp add  \
        0  0.002169  0.016612  0.015480  0.002994  0.017598  0.025732  0.022664
        1  0.009038  0.001400  0.002101  0.000374  0.006425  0.000000  0.000000
        2  0.031815  0.003894  0.000420  0.000000  0.002374  0.008950  0.016752
        3  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000  0.000000
        4  0.009255  0.001095  0.002101  0.000374  0.005587  0.000000  0.000000

            jmp cmp  jmp call   jmp lea  ...    lea add    lea or   lea cmp  lea call  \
        0  0.022002  0.013257  0.005995  ...  0.013914  0.006827  0.017680  0.003501
```

```
1  0.002806  0.001128  0.000521  ...  0.001113  0.002276  0.003584  0.000400
2  0.000112  0.000564  0.000391  ...  0.002226  0.007396  0.001911  0.000833
3  0.000000  0.000000  0.000000  ...  0.001113  0.002276  0.000478  0.000000
4  0.000449  0.001410  0.000521  ...  0.000000  0.000000  0.001911  0.000333

     lea jz   lea lea  movzx mov  movzx sub  movzx add                    ID
0  0.010045  0.007178   0.015875   0.056518   0.004574  01azqd4InC7m9JpocGv5
1  0.000000  0.000479   0.000772   0.036972   0.000286  01IsoiSMh5gxyDYTl4CB
2  0.002318  0.006102   0.000000   0.000000   0.000000  01jsnpXSAlgw6aPeDxrU
3  0.000000  0.000000   0.000000   0.000000   0.000000  01kcPWA9K2BOxQeS5Rju
4  0.000000  0.000120   0.000000   0.036266   0.000000  01SuzwMJEIXsK7A8dQbl

[5 rows x 201 columns]
```

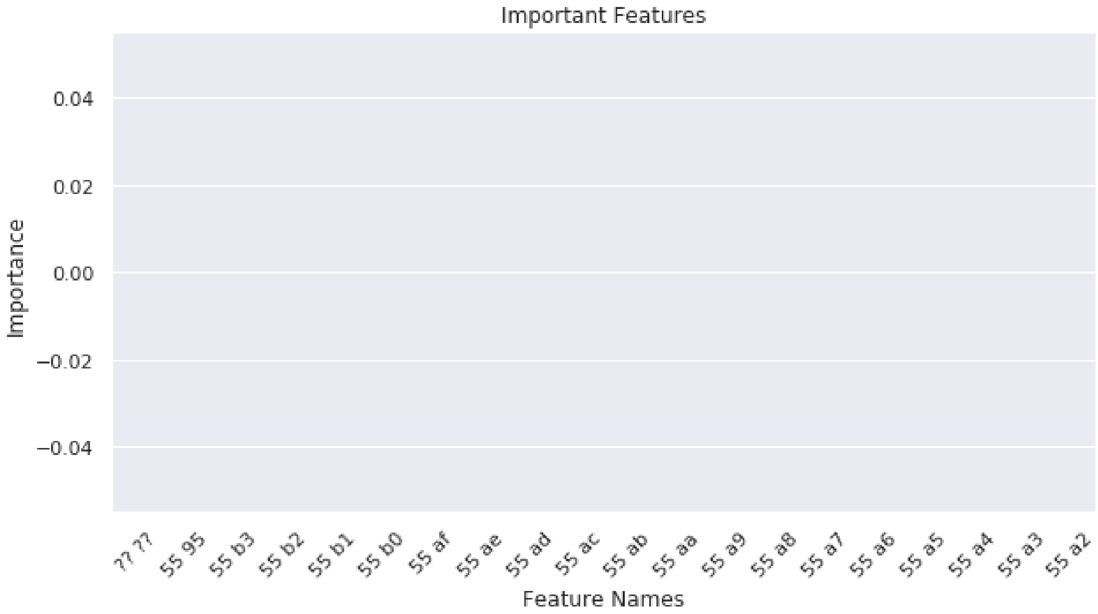
```
In [0]: op_trigram = pd.SparseDataFrame(normalize(trigram_vect, axis = 0), columns = asm_trigra
        op_trigram = op_trigram.loc[:, np.intersect1d(op_trigram.columns, np.take(asm_trigram,
```

```
In [0]: op_trigram = pd.read_csv('op_trigram_final.csv').drop('Unnamed: 0', axis = 1).fillna(0)
        op_trigram['ID'] = result.ID
        op_trigram.head()
```

```
Out[0]:    jmp add add  jmp add call  jmp add cmp  jmp add dec  jmp add imul  \
        0          0.0      0.033763     0.026559          0.0           0.0
        1          0.0      0.000000     0.000000          0.0           0.0
        2          0.0      0.000000     0.013280          0.0           0.0
        3          0.0      0.000000     0.000000          0.0           0.0
        4          0.0      0.000000     0.000000          0.0           0.0

           jmp add inc  jmp add jmp  jmp add jz  jmp add lea  jmp add mov  ...  \
        0     0.042737     0.000000    0.013761     0.033191     0.023325  ...
        1     0.000000     0.000000    0.000000     0.000000     0.000000  ...
        2     0.000000     0.005556    0.000000     0.000000     0.026069  ...
        3     0.000000     0.000000    0.000000     0.000000     0.000000  ...
        4     0.000000     0.000000    0.000000     0.000000     0.000000  ...

           jmp xor lea  jmp xor mov  jmp xor or  jmp xor pop  jmp xor push  \
        0     0.006909     0.007039    0.018413     0.010029      0.011196
        1     0.006909     0.011856    0.000000     0.002006      0.000000
        2     0.000000     0.001852    0.000000     0.000000      0.001120
        3     0.000000     0.000000    0.000000     0.000000      0.000000
        4     0.000000     0.011856    0.000000     0.002006      0.000000

           jmp xor retn  jmp xor sub  jmp xor xchg  jmp xor xor                    ID
        0      0.000000          0.0           0.0     0.008479  01azqd4InC7m9JpocGv5
        1      0.003717          0.0           0.0     0.012718  01IsoiSMh5gxyDYTl4CB
        2      0.003717          0.0           0.0     0.004239  01jsnpXSAlgw6aPeDxrU
        3      0.000000          0.0           0.0     0.000000  01kcPWA9K2BOxQeS5Rju
        4      0.003717          0.0           0.0     0.004239  01SuzwMJEIXsK7A8dQbl
```

```
[5 rows x 301 columns]
```

```
In [0]: byte_imp_feat = imp_features(normalize(byte_vect, axis = 0), byte_bigram_vocab, 300)
```



```
In [0]: byte_imp_feat = np.load('byte_imp_feat.npy')

        byte_imp = np.zeros((10868, 0))
        for i in byte_imp:
            sliced = byte_vect[:, i].todense()
            byte_imp = np.hstack([byte_imp, sliced])

In [0]: byte_imp = pd.SparseDataFrame(byte_imp, columns = np.take(byte_bigram_vocab, byte_imp_i

In [0]: byte_bigram = pd.read_csv('byte_bigram.csv').drop('Unnamed: 0', axis = 1).fillna(0)

        byte_bigram['ID'] = result.ID

        byte_bigram.head()
```

```
Out[0]:    ?? ??  55 95  55 b3  55 b2  55 b1  55 b0  55 af  55 ae  55 ad  55 ac  ...  \
        0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  ...
        1    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  ...
        2    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  ...
        3    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  ...
        4    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  ...
```

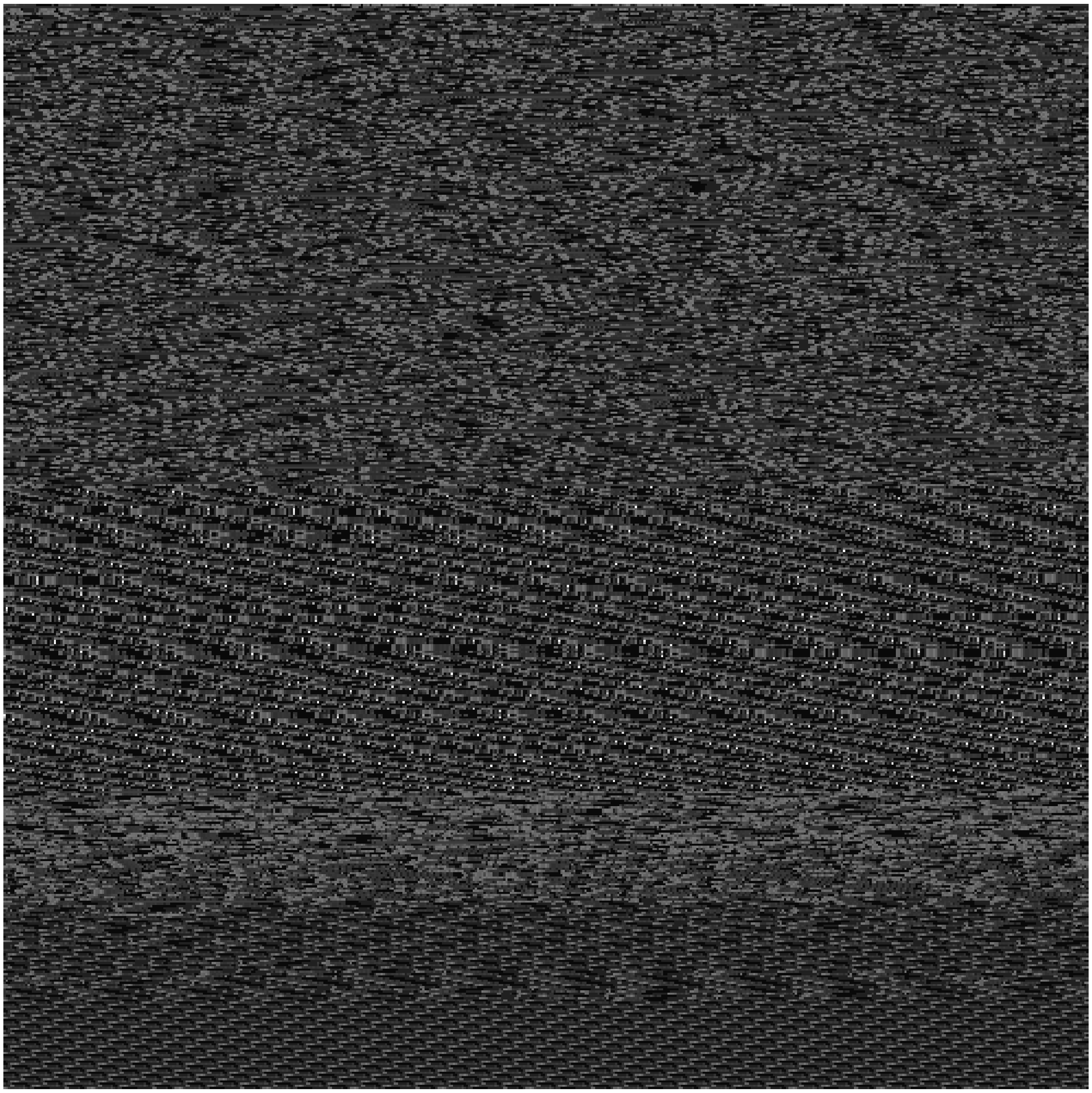
```
   54 b3  54 b4  54 c4  54 d1  54 d0  54 cf  54 ce  54 cd  54 cc  \
0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
3    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
4    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

                     ID
0  01azqd4InC7m9JpocGv5
1  01IsoiSMh5gxyDYTl4CB
2  01jsnpXSAlgw6aPeDxrU
3  01kcPWA9K2BOxQeS5Rju
4  01SuzwMJEIXsK7A8dQbl

[5 rows x 301 columns]
```

1.1 Top 200 Image Features

```
In [0]: #Ref https://github.com/adeya99/Microsoft-Malware-Detection/blob/master/Malware%20Clas
        import array

        def collect_img_asm():
            #pix_file = open("../pixels.txt", "w+")
            for asmfile in os.listdir("./asmFiles"):
                file_name = asmfile.split('.')[0]
                file = codecs.open("./asmFiles/" + asmfile, 'rb')
                file_len = os.path.getsize("./asmFiles/" + asmfile)
                width = int(file_len ** 0.5)
                rem = int(file_len / width)
                arr = array.array('B')
                arr.frombytes(file.read())
                file.close()
                reshaped = np.reshape(arr[:width * width], (width, width))
                reshaped = np.uint8(reshaped)
                scipy.misc.imsave('./asm_image/' + file_name + '.png',reshaped)


        collect_img_asm()
```

```
In [0]: from IPython.display import Image
        Image(filename='asm_image/8vJiQURcq15ZtmEdHOIp.png')
```

```
Out[0]:
```



```
In [0]: import cv2
        image_features = np.zeros((10868, 200))

        for i, asmfile in enumerate(os.listdir("asmFiles")):
            img = cv2.imread("asm_image/" + asmfile.split('.')[0] + '.png')
            img_arr = img.flatten()[:200]
            image_features[i, :] += img_arr
```

```
In [0]: from sklearn.preprocessing import normalize

        img_feat = []
        for i in range(200):
            img_feat.append('pix' + str(i))
        img_final = pd.DataFrame(normalize(image_features, axis = 0), columns = img_feat)
```

```
In [0]: img_final['ID'] = result.ID
        img_final.head()
```

```
Out[0]:         pix0      pix1      pix2      pix3      pix4      pix5      pix6  \
        0  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        1  0.006560  0.006560  0.006560  0.013504  0.013504  0.013504  0.012927
        2  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        3  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        4  0.006560  0.006560  0.006560  0.013504  0.013504  0.013504  0.012927

                pix7      pix8      pix9  ...    pix191    pix192    pix193    pix194  \
        0  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        1  0.012927  0.012927  0.013963  ...  0.009593  0.009593  0.009593  0.009593
        2  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        3  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        4  0.012927  0.012927  0.013963  ...  0.009593  0.009593  0.009593  0.009593

              pix195    pix196    pix197    pix198    pix199                    ID
        0  0.009593  0.009593  0.009593  0.009593  0.009593  01azqd4InC7m9JpocGv5
        1  0.009593  0.009593  0.009593  0.009593  0.009593  01IsoiSMh5gxyDYTl4CB
        2  0.009593  0.009593  0.009593  0.009593  0.009593  01jsnpXSAlgw6aPeDxrU
        3  0.009593  0.009593  0.009593  0.009593  0.009593  01kcPWA9K2BOxQeS5Rju
        4  0.009593  0.009593  0.009593  0.009593  0.009593  01SuzwMJEIXsK7A8dQbl

        [5 rows x 201 columns]
```

```
In [0]: img_final.head()
```

```
Out[0]:         pix0      pix1      pix2      pix3      pix4      pix5      pix6  \
        0  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        1  0.006560  0.006560  0.006560  0.013504  0.013504  0.013504  0.012927
        2  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        3  0.010268  0.010268  0.010268  0.008033  0.008033  0.008033  0.008320
        4  0.006560  0.006560  0.006560  0.013504  0.013504  0.013504  0.012927

                pix7      pix8      pix9  ...    pix191    pix192    pix193    pix194  \
        0  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        1  0.012927  0.012927  0.013963  ...  0.009593  0.009593  0.009593  0.009593
        2  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        3  0.008320  0.008320  0.007913  ...  0.009593  0.009593  0.009593  0.009593
        4  0.012927  0.012927  0.013963  ...  0.009593  0.009593  0.009593  0.009593

              pix195    pix196    pix197    pix198    pix199                    ID
        0  0.009593  0.009593  0.009593  0.009593  0.009593  01azqd4InC7m9JpocGv5
        1  0.009593  0.009593  0.009593  0.009593  0.009593  01IsoiSMh5gxyDYTl4CB
        2  0.009593  0.009593  0.009593  0.009593  0.009593  01jsnpXSAlgw6aPeDxrU
        3  0.009593  0.009593  0.009593  0.009593  0.009593  01kcPWA9K2BOxQeS5Rju
        4  0.009593  0.009593  0.009593  0.009593  0.009593  01SuzwMJEIXsK7A8dQbl
```

```
[5 rows x 201 columns]
```

```
In [0]: # final_data = pd.concat([result_x, op_bi_df, op_tri_df, byte_bi_df, img_df], axis = 1
        final_data = pd.concat([result_x, op_bigram, op_trigram,op_tetragram, byte_bigram, img_
```

```
In [0]: final_data = final_data.drop('ID', axis = 1)
        final_data.head()
```

```
Out[0]:    Unnamed: 0       0     1     2     3     4     5     6     7     8  ...  \
        0           0  601905  3905  2816  3832  3345  3242  3650  3201  2965  ...
        1           1   39755  8337  7249  7186  8663  6844  8420  7589  9291  ...
        2           2   93506  9542  2568  2438  8925  9330  9007  2342  9107  ...
        3           3   21091  1213   726   817  1257   625   550   523  1078  ...
        4           4   19764   710   302   433   559   410   262   249   422  ...

             pix190    pix191    pix192    pix193    pix194    pix195    pix196  \
        0  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593
        1  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593
        2  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593
        3  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593
        4  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593  0.009593

             pix197    pix198    pix199
        0  0.009593  0.009593  0.009593
        1  0.009593  0.009593  0.009593
        2  0.009593  0.009593  0.009593
        3  0.009593  0.009593  0.009593
        4  0.009593  0.009593  0.009593

        [5 rows x 6308 columns]
```

```
In [0]: # final_data = pd.read_csv('final_data_img.csv')
        final_data = pd.read_csv('final_data_img4.csv')
```

```
In [0]: X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(final_data
        X_trn_final, X_cv_final, y_trn_final, y_cv_final = train_test_split(X_train_final, y_t
```

4. Machine Learning Models

Without 2,3 or 4 gram

```
In [0]: # final_data = pd.read_csv('final_data_img.csv')
        final_data = pd.read_csv('final_data.csv')
```

```
In [0]: X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(final_data
        X_trn_final, X_cv_final, y_trn_final, y_cv_final = train_test_split(X_train_final, y_t
```

4.1.1. Random Model

In [0]:
```
# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039

test_data_len = X_test.shape[0]
cv_data_len = X_cv.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicte


# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=

predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)
```

```
Log loss on Cross Validation Data using Random Model 2.45615644965
Log loss on Test Data using Random Model 2.48503905509
Number of misclassified points  88.5004599816
-------------------------------------------------- Confusion matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


-------------------------------------------------- Precision matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Sum of columns in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
-------------------------------------------------- Recall matrix -----------------------------
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.2. K Nearest Neighbour Classification

```
In [0]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/g
        # -------------------------
        # default parameter
        # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, p
        # metric=minkowski, metric_params=None, n_jobs=1, **kwargs)

        # methods of
        # fit(X, y) : Fit the model using X as training data and y as target values
        # predict(X):Predict the class labels for the provided data
        # predict_proba(X):Return probability estimates for the test data X.
        #------------------------------------
        # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
        #------------------------------------


        # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/module
        # ----------------------------
        # default paramters
        # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=3
        #
        # some of the methods of CalibratedClassifierCV()
        # fit(X, y[, sample_weight])        Fit the calibrated model
        # get_params([deep])        Get parameters for this estimator.
        # predict(X)        Predict the target of new samples.
        # predict_proba(X)        Posterior probabilities of classification
        #------------------------------------
        # video link:
        #------------------------------------

        alpha = [x for x in range(1, 15, 2)]
        cv_log_error_array=[]
        for i in alpha:
            k_cfl=KNeighborsClassifier(n_neighbors=i)
            k_cfl.fit(X_train,y_train)
            sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
            sig_clf.fit(X_train, y_train)
            predict_y = sig_clf.predict_proba(X_cv)
            cv_log_error_array.append(log_loss(y_cv, predict_y, labels=k_cfl.classes_, eps=1e-
for i in range(len(cv_log_error_array)):
    print ('log_loss for k = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

k_cfl=KNeighborsClassifier(n_neighbors=alpha[best_alpha])
k_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(X_cv)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for k =  1 is 0.225386237304
log_loss for k =  3 is 0.230795229168
log_loss for k =  5 is 0.252421408646
log_loss for k =  7 is 0.273827486888
log_loss for k =  9 is 0.286469181555
log_loss for k =  11 is 0.29623391147
log_loss for k =  13 is 0.307551203154


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


For values of best alpha =  1 The train log loss is: 0.0782947669247
For values of best alpha =  1 The cross validation log loss is: 0.225386237304
For values of best alpha =  1 The test log loss is: 0.241508604195
Number of misclassified points  4.50781968721
```

```
-------------------------------------------------- Confusion matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


-------------------------------------------------- Precision matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Sum of columns in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
-------------------------------------------------- Recall matrix -----------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.3. Logistic Regression

```
In [0]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
        # -------------------------------
        # default parameters
        # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
        # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=opt
        # class_weight=None, warm_start=False, average=False, n_iter=None)

        # some of methods
        # fit(X, y[, coef_init, intercept_init, ])        Fit linear model with Stochastic Gra
        # predict(X)        Predict class labels for samples in X.

        #-------------------------------
        # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
        #------------------------------

        alpha = [10 ** x for x in range(-5, 4)]
        cv_log_error_array=[]
```

```
for i in alpha:
    logisticR=LogisticRegression(penalty='l2',C=i,class_weight='balanced')
    logisticR.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=logisticR.classes_, eps

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])

best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

logisticR=LogisticRegression(penalty='l2',C=alpha[best_alpha],class_weight='balanced')
logisticR.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train, y_train)
pred_y=sig_clf.predict(X_test)

predict_y = sig_clf.predict_proba(X_train)
print ('log loss for train data',log_loss(y_train, predict_y, labels=logisticR.classes
predict_y = sig_clf.predict_proba(X_cv)
print ('log loss for cv data',log_loss(y_cv, predict_y, labels=logisticR.classes_, eps
predict_y = sig_clf.predict_proba(X_test)
print ('log loss for test data',log_loss(y_test, predict_y, labels=logisticR.classes_,
plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  1e-05 is 1.56916911178
log_loss for c =  0.0001 is 1.57336384417
log_loss for c =  0.001 is 1.53598598273
log_loss for c =  0.01 is 1.01720972418
log_loss for c =  0.1 is 0.857766083873
log_loss for c =  1 is 0.711154393309
log_loss for c =  10 is 0.583929522635
log_loss for c =  100 is 0.549929846589
log_loss for c =  1000 is 0.624746769121


<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>


log loss for train data 0.498923428696
log loss for cv data 0.549929846589
log loss for test data 0.528347316704
Number of misclassified points  12.3275068997
-------------------------------------------------- Confusion matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


-------------------------------------------------- Precision matrix --------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Sum of columns in precision matrix [  1.   1.   1.   1.  nan   1.   1.   1.   1.]
-------------------------------------------------- Recall matrix -----------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.4. Random Forest Classifier

```
In [0]: # --------------------------------
        # default parameters
        # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
        # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
        # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
        # class_weight=None)

        # Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])        Fit the SVM model according to the given training 
# predict(X)        Perform classification on samples in X.
# predict_proba (X)        Perform classification on samples in X.

# some of attributes of  RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
train_log_error_array=[]
from sklearn.ensemble import RandomForestClassifier
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_cv)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=r_cfl.classes_, eps=1e-

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
r_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
        predict_y = sig_clf.predict_proba(X_cv)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(X_test)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_lo
        plot_confusion_matrix(y_test, sig_clf.predict(X_test))
```

```
log_loss for c =  10 is 0.106357709164
log_loss for c =  50 is 0.0902124124145
log_loss for c =  100 is 0.0895043339776
log_loss for c =  500 is 0.0881420869288
log_loss for c =  1000 is 0.0879849524621
log_loss for c =  2000 is 0.0881566647295
log_loss for c =  3000 is 0.0881318948443
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
For values of best alpha =  1000 The train log loss is: 0.0266476291801
For values of best alpha =  1000 The cross validation log loss is: 0.0879849524621
For values of best alpha =  1000 The test log loss is: 0.0858346961407
Number of misclassified points  2.02391904324
-------------------------------------------------- Confusion matrix --------------------------
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
-------------------------------------------------- Precision matrix --------------------------
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Sum of columns in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
-------------------------------------------------- Recall matrix -----------------------------
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
Sum of rows in precision matrix [ 1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

4.1.5. XgBoost Classification

4.5.4. Random Forest Classifier on final features

```
In [0]: # --------------------------------
        # default parameters
        # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
        # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
        # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=Non
        # class_weight=None)

        # Some of methods of RandomForestClassifier()
        # fit(X, y, [sample_weight])        Fit the SVM model according to the given training
        # predict(X)        Perform classification on samples in X.
        # predict_proba (X)        Perform classification on samples in X.

        # some of attributes of  RandomForestClassifier()
        # feature_importances_ : array of shape = [n_features]
        # The feature importances (the higher, the more important the feature).

        # --------------------------------
        # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
        # --------------------------------

        alpha=[10,50,100,500,1000,2000,3000]
        cv_log_error_array=[]
        from sklearn.ensemble import RandomForestClassifier
        for i in alpha:
            r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
            r_cfl.fit(X_train_merge,y_train_merge)
            sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
            sig_clf.fit(X_train_merge, y_train_merge)
            predict_y = sig_clf.predict_proba(X_cv_merge)
            cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=r_cfl.classes_, ep

        for i in range(len(cv_log_error_array)):
            print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


        best_alpha = np.argmin(cv_log_error_array)

        fig, ax = plt.subplots()
        ax.plot(alpha, cv_log_error_array,c='g')
        for i, txt in enumerate(np.round(cv_log_error_array,3)):
            ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
        plt.grid()
        plt.title("Cross Validation Error for each alpha")
        plt.xlabel("Alpha i's")
        plt.ylabel("Error measure")
        plt.show()


        r_cfl=RandomForestClassifier(n_estimators=alpha[best_alpha],random_state=42,n_jobs=-1)
        r_cfl.fit(X_train_merge,y_train_merge)
        sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
        sig_clf.fit(X_train_merge, y_train_merge)

        predict_y = sig_clf.predict_proba(X_train_merge)
        print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
        predict_y = sig_clf.predict_proba(X_cv_merge)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(X_test_merge)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
log_loss for c =  10 is 0.0461221662017
log_loss for c =  50 is 0.0375229563452
log_loss for c =  100 is 0.0359765822455
log_loss for c =  500 is 0.0358291883873
log_loss for c =  1000 is 0.0358403093496
log_loss for c =  2000 is 0.0357908022178
log_loss for c =  3000 is 0.0355909487962
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
For values of best alpha =  3000 The train log loss is: 0.0166267614753
For values of best alpha =  3000 The cross validation log loss is: 0.0355909487962
For values of best alpha =  3000 The test log loss is: 0.0401141303589
```

4.5.5. XgBoost Classifier on final features

```
In [0]: # Training a hyper-parameter tuned Xg-Boost regressor on our train data

        # find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/
        # -------------------------
        # default paramters
        # class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent
# objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_
# max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
# scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwarg

# some of methods of RandomForestRegressor()
# fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds
# get_params([deep])        Get parameters for this estimator.
# predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This fu
# get_score(importance_type='weight') -> get the feature importance
# -----------------------
# video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons
# -----------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i)
    x_cfl.fit(X_train_merge,y_train_merge)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_merge, y_train_merge)
    predict_y = sig_clf.predict_proba(X_cv_merge)
    cv_log_error_array.append(log_loss(y_cv_merge, predict_y, labels=x_cfl.classes_, ep

for i in range(len(cv_log_error_array)):
    print ('log_loss for c = ',alpha[i],'is',cv_log_error_array[i])


best_alpha = np.argmin(cv_log_error_array)

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

x_cfl=XGBClassifier(n_estimators=3000,nthread=-1)
x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_merge, y_train_merge)

predict_y = sig_clf.predict_proba(X_train_merge)
print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
predict_y = sig_clf.predict_proba(X_cv_merge)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(X_test_merge)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
```

```
log_loss for c =  10 is 0.0898979446265
log_loss for c =  50 is 0.0536946658041
log_loss for c =  100 is 0.0387968186177
log_loss for c =  500 is 0.0347960327293
log_loss for c =  1000 is 0.0334668083237
log_loss for c =  2000 is 0.0316569078846
log_loss for c =  3000 is 0.0315972694477
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.HTML object>
```

```
For values of best alpha =  3000 The train log loss is: 0.0111918809342
For values of best alpha =  3000 The cross validation log loss is: 0.0315972694477
For values of best alpha =  3000 The test log loss is: 0.0323978515915
```

4.5.5. XgBoost Classifier on final features with best hyper parameters using Random search

```
In [0]: x_cfl=XGBClassifier()

        prams={
            'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
             'n_estimators':[100,200,500,1000,2000],
             'max_depth':[3,5,10],
            'colsample_bytree':[0.1,0.3,0.5,1],
            'subsample':[0.1,0.3,0.5,1]
        }
        random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
        random_cfl.fit(X_train_merge, y_train_merge)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  2.2min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  4.5min remaining:  2.6min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed:  5.8min remaining:  1.8min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed:  6.7min remaining:   44.5s
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  7.4min finished
```

```
Out[0]: RandomizedSearchCV(cv=None, error_score='raise',
                  estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytre
               gamma=0, learning_rate=0.1, max_delta_step=0, max_depth=3,
               min_child_weight=1, missing=None, n_estimators=100, nthread=-1,
               objective='binary:logistic', reg_alpha=0, reg_lambda=1,
               scale_pos_weight=1, seed=0, silent=True, subsample=1),
                  fit_params=None, iid=True, n_iter=10, n_jobs=-1,
                  param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n_
                  pre_dispatch='2*n_jobs', random_state=None, refit=True,
                  return_train_score=True, scoring=None, verbose=10)
```

```
In [0]: print (random_cfl.best_params_)
```

```
{'subsample': 1, 'n_estimators': 1000, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytre
```

```
In [0]: # find more about XGBClassifier function here http://xgboost.readthedocs.io/en/latest/p
        # -------------------------
        # default paramters
        # class xgboost.XGBClassifier(max_depth=3, learning_rate=0.1, n_estimators=100, silent=
        # objective='binary:logistic', booster='gbtree', n_jobs=1, nthread=None, gamma=0, min_
        # max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0,
        # scale_pos_weight=1, base_score=0.5, random_state=0, seed=None, missing=None, **kwarg

        # some of methods of RandomForestRegressor()
        # fit(X, y, sample_weight=None, eval_set=None, eval_metric=None, early_stopping_rounds=
        # get_params([deep])       Get parameters for this estimator.
        # predict(data, output_margin=False, ntree_limit=0) : Predict with data. NOTE: This fu
        # get_score(importance_type='weight') -> get the feature importance
        # -----------------------
        # video link2: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
        # -----------------------

        x_cfl=XGBClassifier(n_estimators=1000,max_depth=10,learning_rate=0.15,colsample_bytree=
        x_cfl.fit(X_train_merge,y_train_merge,verbose=True)
        sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
        sig_clf.fit(X_train_merge, y_train_merge)

        predict_y = sig_clf.predict_proba(X_train_merge)
        print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_l
        predict_y = sig_clf.predict_proba(X_cv_merge)
        print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss
        predict_y = sig_clf.predict_proba(X_test_merge)
        print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_los
        plot_confusion_matrix(y_test_asm,sig_clf.predict(X_test_merge))
```

```
For values of best alpha =  3000 The train log loss is: 0.0121922832297
For values of best alpha =  3000 The cross validation log loss is: 0.0344955487471
For values of best alpha =  3000 The test log loss is: 0.0317041132442
```

1.1.1 With Image Features and 2,3,4 -grams

```
In [0]: # final_data = pd.read_csv('final_data_img.csv')
        final_data = pd.read_csv('final_data_img4.csv')
```

```
In [0]: X_train_final, X_test_final, y_train_final, y_test_final = train_test_split(final_data
        X_trn_final, X_cv_final, y_trn_final, y_cv_final = train_test_split(X_train_final, y_t
```

```
In [0]: x_cfl = XGBClassifier()

        prams={
            'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
             'n_estimators':[100,200,500,1000,2000],
             'max_depth':[3,5,10],
            'colsample_bytree':[0.1,0.3,0.5,1],
            'subsample':[0.1,0.3,0.5,1]
        }
        random_cfl=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
        random_cfl.fit(X_trn_final, y_trn_final)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits


[Parallel(n_jobs=-1)]: Using backend LokyBackend with 24 concurrent workers.
[Parallel(n_jobs=-1)]: Done   3 out of  30 | elapsed:   55.2s remaining:  8.3min
[Parallel(n_jobs=-1)]: Done   7 out of  30 | elapsed:  2.8min remaining:  9.1min
[Parallel(n_jobs=-1)]: Done  11 out of  30 | elapsed:  3.0min remaining:  5.2min
[Parallel(n_jobs=-1)]: Done  15 out of  30 | elapsed:  3.0min remaining:  3.0min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  8.7min remaining:  5.0min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 11.7min remaining:  3.6min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 13.2min remaining:  1.5min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 23.5min finished
```

```
Out[0]: RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                  estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
               colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
               max_delta_step=0, max_depth=3, min_child_weight=1, missing=None,
               n_estimators=100, n_jobs=1, nthread=None,
               objective='binary:logistic', random_state=0, reg_alpha=0,
               reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
               subsample=1, verbosity=1),
                  fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
                  param_distributions={'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2], 'n
                  pre_dispatch='2*n_jobs', random_state=None, refit=True,
                  return_train_score='warn', scoring=None, verbose=10)
```

```
In [0]: random_cfl.best_params_
```

```
Out[0]: {'subsample': 0.5,
         'n_estimators': 500,
         'max_depth': 5,
         'learning_rate': 0.03,
         'colsample_bytree': 0.5}
```

```
In [4]: x_cfl=XGBClassifier(n_estimators=500,max_depth=5,learning_rate=0.03,colsample_bytree=0
        x_cfl.fit(X_trn_final,y_trn_final,verbose=True)
        sig_clf = CalibratedClassifierCV(X_cfl, method="sigmoid")
        sig_clf.fit(X_trn_final, y_trn_final)

        # filename = 'finalized_model_clf.sav'
        # pickle.dump(sig_clf, open(filename, 'wb'))
        predict_y = sig_clf.predict_proba(X_trn_final)
        # print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",lo
        print ("The train log loss is:", log_loss(y_trn_final, predict_y))

        predict_y = sig_clf.predict_proba(X_cv_final)
        # print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
        print("The cross validation log loss is:", log_loss(y_cv_final, predict_y))

        predict_y = sig_clf.predict_proba(X_test_final)
        # print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
        print("The test log loss is:", log_loss(y_test_final, predict_y))

        # plot_confusion_matrix(y_test_asm,sig_clf.predict(x_test_final))
```

```
The train log loss is: 0.009331682518494231
The cross validation log loss is: 0.01211256309293162
The test log loss is: 0.01623229063638662
```

```
In [5]: from prettytable import PrettyTable

        x = PrettyTable()

        x.field_names = ["Algorithm", "Train log loss", "Test log loss"]

        x.add_row(["Random", 2.45615644965, 2.48503905509])
        x.add_row(["KNN", 0.0782947669247, 0.241508604195 ])
        x.add_row(["Logistic Regression", 0.498923428696,0.528347316704 ])
        x.add_row(["Random Forest", 0.0166267614753, 0.0401141303589 ])
        x.add_row(["XGBOOST", 0.0111918809342, 0.0323978515915 ])
        x.add_row(["XGBOOST-Random Search", 0.0121922832297, 0.0317041132442 ])
        x.add_row(["XGBOOST-(Image features/N-Grams)",  0.009331682518494231,0.0162322906363866
print(x)
```

```
+----------------------------------+----------------------+---------------------+
|            Algorithm             |    Train log loss    |     Test log loss   |
+----------------------------------+----------------------+---------------------+
|              Random              |    2.45615644965     |     2.48503905509   |
|               KNN                |   0.0782947669247    |     0.241508604195  |
|       Logistic Regression        |    0.498923428696    |     0.528347316704  |
|          Random Forest           |   0.0166267614753    |    0.0401141303589  |
|             XGBOOST              |   0.0111918809342    |    0.0323978515915  |
|      XGBOOST-Random Search       |   0.0121922832297    |    0.0317041132442  |
| XGBOOST-(Image features/N-Grams) | 0.009331682518494231 | 0.01623229063638662 |
+----------------------------------+----------------------+---------------------+
```