MicrosoftMalwareDetection2

January 16, 2019

1 Microsoft Malware detection

1.Business/Real-world Problem

1.1. What is Malware?

The term malware is a contraction of malicious software. Put simply, malware is any piece of software that was written with the intent of doing harm to data, devices or to people. Source: https://www.avg.com/en/signal/what-is-malware

1.2. Problem Statement

In the past few years, the malware industry has grown very rapidly that, the syndicates invest heavily in technologies to evade traditional protection, forcing the anti-malware groups/communities to build more robust softwares to detect and terminate these attacks. The major part of protecting a computer system from a malware attack is to identify whether a given piece of file/software is a malware.

1.3 Source/Useful Links

Microsoft has been very active in building anti-malware products over the years and it runs it's anti-malware utilities over 150 million computers around the world. This generates tens of millions of daily data points to be analyzed as potential malware. In order to be effective in analyzing and classifying such large amounts of data, we need to be able to group them into groups and identify their respective families. This dataset provided by Microsoft contains about 9 classes of malware. ,

Source: https://www.kaggle.com/c/malware-classification

1.4. Real-world/Business objectives and constraints.

1. Minimize multi-class error.
2. Multi-class probability estimates.
3. Malware detection should not take hours and block the user's computer. It should fininsh in a few seconds or a minute.

2. Machine Learning Problem

2.1. Data

2.1.1. Data Overview

Source : https://www.kaggle.com/c/malware-classification/data

For every malware, we have two files

.asm file (read more: https://www.reviversoft.com/file-extensions/asm)

.bytes file (the raw data contains the hexadecimal representation of the file's binary content, without the PE header)

Total train dataset consist of 200GB data out of which 50Gb of data is .bytes files and 150GB of data is .asm files:

Lots of Data for a single-box/computer.

There are total 10,868 .bytes files and 10,868 asm files total 21,736 files

There are 9 types of malwares (9 classes) in our give data

Types of Malware:

Ramnit

Lollipop

Kelihos_ver3

Vundo

Simda

Tracur

Kelihos_ver1

Obfuscator.ACY

Gatak

2.1.2. Example Data Point

.asm file

.bytes file

2.2. Mapping the real-world problem to an ML problem

2.2.1. Type of Machine Learning Problem

```
There are nine different classes of malware that we need to classify a given a data point =
```

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/malware-classification#evaluation

Metric(s): * Multi class log-loss * Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- Some Latency constraints.

2.3. Train and Test Dataset

Split the dataset randomly into three parts train, cross validation and test with 64%,16%, 20% of data respectively

2.4. Useful blogs, videos and reference papers

http://blog.kaggle.com/2015/05/26/microsoft-malware-winners-interview-1st-place-no-to-overfitting/ https://arxiv.org/pdf/1511.04317.pdf First place solution in Kaggle competition: https://www.youtube.com/watch?v=VLQTRlLGz5Y https://github.com/dchad/malware-detection http://vizsec.org/files/2011/Nataraj.pdf https://www.dropbox.com/sh/gfqzv0ckgs4l1bf/AAB6EelnEjvvuQg2nu_pIB6ua?dl=0 " Cross validation is more trustworthy than domain knowledge."

3. Exploratory Data Analysis

In [0]:
```
import warnings
warnings.filterwarnings("ignore")
import shutil
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
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
```

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 asm_files:
```

```
            if (file.endswith("bytes")):
                shutil.move(source+file,destination)
```

3.1. Distribution of malware classes in whole data set

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

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

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

3.2. Feature extraction
3.2.1 File size of byte files as a feature

```
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())
```

```
   Class                    ID      size
0      9  01azqd4InC7m9JpocGv5  4.234863
1      2  01IsoiSMh5gxyDYTl4CB  5.538818
2      9  01jsnpXSAlgw6aPeDxrU  3.887939
3      1  01kcPWA9K2BOxQeS5Rju  0.574219
4      8  01SuzwMJEIXsK7A8dQbl  0.370850
```

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()
```

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

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

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

files = os.listdir('byteFiles')
filenames=[]
array=[]
for file in files:
    if(f.endswith("bytes")):
        file=file.split('.')[0]
        text_file = open('byteFiles/'+file+".txt", 'w+')
        with open('byteFiles/'+file,"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)
```

```
            text_file.close()

    files = os.listdir('byteFiles')
    filenames2=[]
    feature_matrix = np.zeros((len(files),257),dtype=int)
    k=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.csv")
        print (byte_features.head())
```

```
                     ID       0     1     2     3     4     5     6     7  \
0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242  3650  3201
1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844  8420  7589
2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330  9007  2342
3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   550   523
4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   262   249

      8  ...     f7    f8    f9    fa    fb    fc    fd     fe     ff     ??
0  2965  ...   2804  3687  3101  3211  3097  2758  3099   2759   5753   1824
1  9291  ...    451  6536   439   281   302  7639   518  17001  54902   8588
2  9107  ...   2325  2358  2242  2885  2863  2471  2786   2680  49144    468
3  1078  ...    478   873   485   462   516  1133   471    761   7998  13940
```

```
4   422  ...     847   947   350   209   239   653   221    242   2199   9008

[5 rows x 258 columns]
```

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

```
Out[0]:                     ID       0     1     2     3     4     5     6     7  \
        0  01azqd4InC7m9JpocGv5  601905  3905  2816  3832  3345  3242  3650  3201
        1  01IsoiSMh5gxyDYTl4CB   39755  8337  7249  7186  8663  6844  8420  7589
        2  01jsnpXSAlgw6aPeDxrU   93506  9542  2568  2438  8925  9330  9007  2342
        3  01kcPWA9K2BOxQeS5Rju   21091  1213   726   817  1257   625   550   523
        4  01SuzwMJEIXsK7A8dQbl   19764   710   302   433   559   410   262   249

              8    ...       f9    fa    fb    fc    fd     fe     ff     ??  Class  \
        0  2965    ...     3101  3211  3097  2758  3099   2759   5753   1824      9
        1  9291    ...      439   281   302  7639   518  17001  54902   8588      2
        2  9107    ...     2242  2885  2863  2471  2786   2680  49144    468      9
        3  1078    ...      485   462   516  1133   471    761   7998  13940      1
        4   422    ...      350   209   239   653   221    242   2199   9008      8

               size
        0  4.234863
        1  5.538818
        2  3.887939
        3  0.574219
        4  0.370850

        [5 rows x 260 columns]
```

```
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_v
            return result1
        result = normalize(result)
```

```
In [0]: data_y = result['Class']
        result.head()
```

```
Out[0]:                     ID         0         1         2         3         4  \
        0  01azqd4InC7m9JpocGv5  0.262806  0.005498  0.001567  0.002067  0.002048
        1  01IsoiSMh5gxyDYTl4CB  0.017358  0.011737  0.004033  0.003876  0.005303
        2  01jsnpXSAlgw6aPeDxrU  0.040827  0.013434  0.001429  0.001315  0.005464
```

```
3  01kcPWA9K2BOxQeS5Rju  0.009209  0.001708  0.000404  0.000441  0.000770
4  01SuzwMJEIXsK7A8dQbl  0.008629  0.001000  0.000168  0.000234  0.000342

          5         6         7         8    ...         f9        fa  \
0  0.001835  0.002058  0.002946  0.002638    ...   0.013560  0.013107
1  0.003873  0.004747  0.006984  0.008267    ...   0.001920  0.001147
2  0.005280  0.005078  0.002155  0.008104    ...   0.009804  0.011777
3  0.000354  0.000310  0.000481  0.000959    ...   0.002121  0.001886
4  0.000232  0.000148  0.000229  0.000376    ...   0.001530  0.000853

         fb        fc        fd        fe        ff        ??  Class      size
0  0.013634  0.031724  0.014549  0.014348  0.007843  0.000129      9  0.092219
1  0.001329  0.087867  0.002432  0.088411  0.074851  0.000606      2  0.121236
2  0.012604  0.028423  0.013080  0.013937  0.067001  0.000033      9  0.084499
3  0.002272  0.013032  0.002211  0.003957  0.010904  0.000984      1  0.010759
4  0.001052  0.007511  0.001038  0.001258  0.002998  0.000636      8  0.006233

[5 rows x 260 columns]
```

3.2.4 Multivariate Analysis

In [0]:
```
#multivariate analysis on byte files
#this is with perplexity 50
xtsne=TSNE(perplexity=50)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

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


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

In [0]:
```
#this is with perplexity 30
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(result.drop(['ID','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=data_y, cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```

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

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

2 Train Test split

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]:
```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.shape[0])
```

```
Number of data points in train data: 6955
Number of data points in test data: 2174
Number of data points in cross validation data: 1739
```

In [0]:
```
# it returns a dict, keys as class labels and values as the number of data points in th
train_class_distribution = y_train.value_counts().sortlevel()
test_class_distribution = y_test.value_counts().sortlevel()
cv_class_distribution = y_cv.value_counts().sortlevel()

my_colors = 'rgbkymc'
train_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',train_class_distribution.values[i]


print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()
```

```python
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',test_class_distribution.values[i]

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i],
```

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


<IPython.core.display.HTML object>


Number of data points in class 3 : 1883 ( 27.074 %)
Number of data points in class 2 : 1586 ( 22.804 %)
Number of data points in class 1 : 986 ( 14.177 %)
Number of data points in class 8 : 786 ( 11.301 %)
Number of data points in class 9 : 648 ( 9.317 %)
Number of data points in class 6 : 481 ( 6.916 %)
Number of data points in class 4 : 304 ( 4.371 %)
Number of data points in class 7 : 254 ( 3.652 %)
Number of data points in class 5 : 27 ( 0.388 %)
--------------------------------------------------------------------------------


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>


Number of data points in class 3 : 588 ( 27.047 %)
Number of data points in class 2 : 496 ( 22.815 %)
Number of data points in class 1 : 308 ( 14.167 %)
```

```
Number of data points in class 8 : 246 ( 11.316 %)
Number of data points in class 9 : 203 ( 9.338 %)
Number of data points in class 6 : 150 ( 6.9 %)
Number of data points in class 4 : 95 ( 4.37 %)
Number of data points in class 7 : 80 ( 3.68 %)
Number of data points in class 5 : 8 ( 0.368 %)
--------------------------------------------------------------------------------
```

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

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

```
Number of data points in class 3 : 471 ( 27.085 %)
Number of data points in class 2 : 396 ( 22.772 %)
Number of data points in class 1 : 247 ( 14.204 %)
Number of data points in class 8 : 196 ( 11.271 %)
Number of data points in class 9 : 162 ( 9.316 %)
Number of data points in class 6 : 120 ( 6.901 %)
Number of data points in class 4 : 76 ( 4.37 %)
Number of data points in class 7 : 64 ( 3.68 %)
Number of data points in class 5 : 7 ( 0.403 %)
```

In [0]:
```
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    print("Number of misclassified points ",(len(test_y)-np.trace(C))/len(test_y)*100)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predi

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that colum

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                            [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                            [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
```

```
# C = [[1, 2],
#      [3, 4]]
# C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in
# C.sum(axix =0) = [[4, 6]]
# (C/C.sum(axis=0)) = [[1/4, 2/6],
#                       [3/4, 4/6]]

labels = [1,2,3,4,5,6,7,8,9]
cmap=sns.light_palette("green")
# representing A in heatmap format
print("-"*50, "Confusion matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

print("-"*50, "Precision matrix", "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of columns in precision matrix",B.sum(axis=0))

# representing B in heatmap format
print("-"*50, "Recall matrix"    , "-"*50)
plt.figure(figsize=(10,5))
sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=la
plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()
print("Sum of rows in precision matrix",A.sum(axis=1))
```

4. Machine Learning Models

4.1. Machine Leaning Models on bytes files
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]:
```
# 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.
#-------------------------------------
# ----------------------------
# 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
#-------------------------------------


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]: # 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.

        #-------------------------------



        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).

        # --------------------------------
        -------------------------------

        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-1

        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_lo
        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 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

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


# -------------------------
# 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
```

```
# ------------------------
----------------------

alpha=[10,50,100,500,1000,2000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train,y_train)
    sig_clf = CalibratedClassifierCV(x_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=x_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()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train,y_train)
sig_clf = CalibratedClassifierCV(x_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_
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 0.20615980494
log_loss for c =  50 is 0.123888382365
log_loss for c =  100 is 0.099919437112
log_loss for c =  500 is 0.0931035681289
log_loss for c =  1000 is 0.0933084876012
log_loss for c =  2000 is 0.0938395690309
```

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

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

```
For values of best alpha =  500 The train log loss is: 0.0225231805824
For values of best alpha =  500 The cross validation log loss is: 0.0931035681289
For values of best alpha =  500 The test log loss is: 0.0792067651731
Number of misclassified points  1.24195032199
-------------------------------------------------- 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 with best hyper parameters using RandomSearch

In [0]:
```
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-
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_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,verbose=10,n_jobs=-1,)
        random_cfl1.fit(X_train,y_train)
```

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


[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  5.8min
[Parallel(n_jobs=-1)]: Done  19 out of  30 | elapsed:  9.3min remaining:  5.4min
[Parallel(n_jobs=-1)]: Done  23 out of  30 | elapsed: 10.1min remaining:  3.1min
[Parallel(n_jobs=-1)]: Done  27 out of  30 | elapsed: 14.0min remaining:  1.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 14.2min finished
```

```
Out[0]: RandomizedSearchCV(cv=None, error_score='raise',
                  estimator=XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree
               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_cfl1.best_params_)
```

```
{'subsample': 1, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.05, 'colsample_bytree
```

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


        # -------------------------
        # 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
# -----------------------


x_cfl=XGBClassifier(n_estimators=2000, learning_rate=0.05, colsample_bytree=1, max_dep
x_cfl.fit(X_train,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train,y_train)

predict_y = c_cfl.predict_proba(X_train)
print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_cv)
print ('cv loss',log_loss(y_cv, predict_y))
predict_y = c_cfl.predict_proba(X_test)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.022540976086
cv loss 0.0928710624158
test loss 0.0782688587098
```

```
In [1]: from prettytable import PrettyTable
```

Conclusion

```
In [3]: x = PrettyTable()

        x.field_names = ["Model Name","Log-Loss", "Misclassified Points"]

        x.add_row(["Random","2.485","88.56"])
        x.add_row(["K-NN","0.241","4.507"])
        x.add_row(["Logistic Regression","0.528","12.327"])
        x.add_row(["Random Forest","0.085","2.02"])
        x.add_row(["XGBoost","0.079","1.241"])

        print(x)
```

```
+---------------------+----------+----------------------+
|      Model Name     | Log-Loss | Misclassified Points |
+---------------------+----------+----------------------+
|        Random       |  2.485   |        88.56         |
|         K-NN        |  0.241   |        4.507         |
| Logistic Regression |  0.528   |        12.327        |
|    Random Forest    |  0.085   |         2.02         |
|       XGBoost       |  0.079   |        1.241         |
+---------------------+----------+----------------------+
```

Further work from here

In the next section we are going to use bigram features from byte files and train our model
The main objective of this section will be to bring down log-loss to 0.01
In part 2 of this project we will use asm files