In [1]:

```
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 cv2
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
import array
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 nltk.util import ngrams
```

In [2]:

```
import pandas as pd
from tqdm import tqdm
import imageio
asm_df=pd.read_csv('asmoutputfile.csv')
```

In [5]:

```
asm_df.shape
```

Out[5]:

```
(10868, 52)
```

In [6]:

```
asm_df.head()
```

Out[6]:

	ID	HEADER:	.text:	.Pav:	.idata:	.data:	.bss:	.rdata:	.edata:	.rsrc:
0	01kcPWA9K2BOxQeS5Rju	19	744	0	127	57	0	323	0	3
1	1E93CpP60RHFNiT5Qfvn	17	838	0	103	49	0	0	0	3
2	3ekVow2ajZHbTnBcsDfX	17	427	0	50	43	0	145	0	3
3	3X2nY7iQaPBIWDrAZqJe	17	227	0	43	19	0	0	0	3
4	46OZzdsSKDCFV8h7XWxf	17	402	0	59	170	0	0	0	3

5 rows × 52 columns

In [14]:

```
#https://towardsdatascience.com/malware-classification-using-machine-learning-7c648fb1da79
files=os.listdir("asmFiles")
for file in tqdm(files):
    filename = file.split('.')[0]
    f = codecs.open("asmFiles/" +file, 'rb')
    length= os.path.getsize("asmFiles/" +file)
    width = int(length ** 0.5)
    rem = int(length/ width)
    arr = array.array('B')
    arr.frombytes(f.read())
    f.close()
    reshaped = np.reshape(arr[:width * width], (width, width))
    reshaped = np.uint8(reshaped)
    imageio.imsave('asm_image/' + filename + '.png',reshaped)
```

```
100%|██████████| 10868/10868 [3:52:19<00:00,  1.28s/it]
```

In [50]:

```
#https://github.com/sai977/microsoft-malware-detection/blob/master/MicrosoftMalwareDetectio
import cv2
imagefeatures = np.zeros((10868, 200))
```

In [51]:

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

In [52]:

```
type(imagefeatures)
```

Out[52]:

```
numpy.ndarray
```

In [53]:

```
img_features_name = []
for i in range(200):
    img_features_name.append('pix' + str(i))
```

In [54]:

```
imgdf = pd.DataFrame((imagefeatures), columns = img_features_name)
```

In [55]:

```
imgdf.head()
```

Out[55]:

	pix0	pix1	pix2	pix3	pix4	pix5	pix6	pix7	pix8	pix9	...	pix190	pix191	pix192
0	72.0	72.0	72.0	69.0	69.0	69.0	65.0	65.0	65.0	68.0	...	45.0	45.0	45.0
1	46.0	46.0	46.0	116.0	116.0	116.0	101.0	101.0	101.0	120.0	...	45.0	45.0	45.0
2	72.0	72.0	72.0	69.0	69.0	69.0	65.0	65.0	65.0	68.0	...	45.0	45.0	45.0
3	72.0	72.0	72.0	69.0	69.0	69.0	65.0	65.0	65.0	68.0	...	45.0	45.0	45.0
4	72.0	72.0	72.0	69.0	69.0	69.0	65.0	65.0	65.0	68.0	...	45.0	45.0	45.0

5 rows × 200 columns

In [2]:

```
imgdf=pd.read_csv('image_features.csv')
```

In [5]:

```
y=pd.read_csv('trainLabels.csv')
y=y['Class']
```

In [6]:

```
X_train_asm, X_test_asm, y_train_asm, y_test_asm = train_test_split(imgdf,y ,stratify=y,tes
X_train_asm, X_cv_asm, y_train_asm, y_cv_asm = train_test_split(X_train_asm, y_train_asm,st
```

```
#KNN
```

In [7]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train_asm)
scaler.transform(X_train_asm)
scaler.transform(X_cv_asm)
```

Out[7]:

```
array([[-1.0577401 ,  0.48869151,  0.48869151, ...,  0.01199175,
         0.01199175,  0.01199175],
       [ 0.77509524,  0.48869151,  0.48869151, ...,  0.01199175,
         0.01199175,  0.01199175],
       [ 0.7228014 ,  0.48869151,  0.48869151, ...,  0.01199175,
         0.01199175,  0.01199175],
       ...,
       [-1.62181207, -1.78472789, -1.78472789, ...,  0.01199175,
         0.01199175,  0.01199175],
       [ 0.93580314, -1.78472789, -1.78472789, ...,  0.01199175,
         0.01199175,  0.01199175],
       [ 1.47500365,  0.48869151,  0.48869151, ...,  0.01199175,
         0.01199175,  0.01199175]])
```

In [84]:

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/genera
# ------------------------

alpha = [x for x in range(1, 21,2)]
cv_log_error_array=[]
for i in alpha:
    k_cfl=KNeighborsClassifier(n_neighbors=i)
    k_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=k_cfl.classes_, eps=1e-1

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_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(k_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
pred_y=sig_clf.predict(X_test_asm)


predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',log_loss(y_train_asm, predict_y))
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',log_loss(y_cv_asm, predict_y))
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',log_loss(y_test_asm, predict_y))
```

```
log_loss for k =  1 is 1.9007431478587495
log_loss for k =  3 is 1.90021685474741
log_loss for k =  5 is 1.8998562163237904
log_loss for k =  7 is 1.9006330524381227
log_loss for k =  9 is 1.9005240107141173
log_loss for k =  11 is 1.8997854850504061
log_loss for k =  13 is 1.899940808887257
log_loss for k =  15 is 1.900386432750284
log_loss for k =  17 is 1.900817750512503
log_loss for k =  19 is 1.8995922501820257
log loss for train data 1.8986815155936156
log loss for cv data 1.8995922501820257
log loss for test data 1.9004669967069017
```

```
#Logistic regression
```

In [85]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal
# 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 Gradient Desce
# predict(X)    Predict class labels for samples in X.

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


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_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, 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_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(logisticR, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=logisticR.classes
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=logisticR.classes_, eps
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=logisticR.classes_,
```

```
log_loss for c =  1e-05 is 1.9009276538582975
log_loss for c =  0.0001 is 1.9007559853431149
log_loss for c =  0.001 is 1.900194523795631
log_loss for c =  0.01 is 1.9002274021385404
log_loss for c =  0.1 is 1.900333895661907
```

```
log_loss for c =  1 is 1.900272427923307
log_loss for c =  10 is 1.9002277263669891
log_loss for c =  100 is 1.9002473550720096
log_loss for c =  1000 is 1.9002115015225962
log loss for train data 1.8994641717475333
log loss for cv data 1.900194523795631
log loss for test data 1.8999456559801227
```

In [87]:

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

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
# 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/rando
# --------------------------------

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    r_cfl=RandomForestClassifier(n_estimators=i,random_state=42,n_jobs=-1)
    r_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, 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_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(r_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)
predict_y = sig_clf.predict_proba(X_train_asm)
print ('log loss for train data',(log_loss(y_train_asm, predict_y, labels=sig_clf.classes_,
predict_y = sig_clf.predict_proba(X_cv_asm)
print ('log loss for cv data',(log_loss(y_cv_asm, predict_y, labels=sig_clf.classes_, eps=1
predict_y = sig_clf.predict_proba(X_test_asm)
print ('log loss for test data',(log_loss(y_test_asm, predict_y, labels=sig_clf.classes_, e
```

```
log_loss for c =  10 is 1.901305907386374
log_loss for c =  50 is 1.9015514185982654
log_loss for c =  100 is 1.9017623008144102
log_loss for c =  500 is 1.9017019201698366
log_loss for c =  1000 is 1.9018027584219617
log_loss for c =  2000 is 1.901847232429469
log_loss for c =  3000 is 1.901851304214067
log loss for train data 1.906384707650573
log loss for cv data 1.901305907386374
log loss for test data 1.9011828047817003
```

In [8]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

alpha=[10,50,100,500,1000,2000,3000]
cv_log_error_array=[]
for i in alpha:
    x_cfl=XGBClassifier(n_estimators=i,nthread=-1)
    x_cfl.fit(X_train_asm,y_train_asm)
    sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
    sig_clf.fit(X_train_asm, y_train_asm)
    predict_y = sig_clf.predict_proba(X_cv_asm)
    cv_log_error_array.append(log_loss(y_cv_asm, predict_y, labels=x_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()

x_cfl=XGBClassifier(n_estimators=alpha[best_alpha],nthread=-1)
x_cfl.fit(X_train_asm,y_train_asm)
sig_clf = CalibratedClassifierCV(x_cfl, method="sigmoid")
sig_clf.fit(X_train_asm, y_train_asm)

predict_y = sig_clf.predict_proba(X_train_asm)

print ('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_cv_asm)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(X_test_asm)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_
```

```
log_loss for c =  10 is 0.00892940072388406
log_loss for c =  50 is 0.008928101391465378
log_loss for c =  100 is 0.008927848785653378
log_loss for c =  500 is 0.0089289430933055
log_loss for c =  1000 is 0.008928778863411096
log_loss for c =  2000 is 0.008928904895220172
log_loss for c =  3000 is 0.008928290533490849
For values of best alpha =  100 The train log loss is: 0.008358048109044992
For values of best alpha =  100 The cross validation log loss is: 0.00892784
8785653378
For values of best alpha =  100 The test log loss is: 0.00821029874953462
```

In [9]:

```
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_asm,y_train_asm)
print (random_cfl.best_params_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   46.4s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  1.8min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:  4.5min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:  5.7min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed:  6.6min remaining:  1.
5min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 10.7min remaining:   4
0.8s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 10.9min finished

{'colsample_bytree': 0.5, 'max_depth': 3, 'subsample': 0.3, 'learning_rate':
0.15, 'n_estimators': 1000}
```

In [10]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

x_cfl=XGBClassifier(n_estimators=200,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5,
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

```
train loss 0.008262141416707685
cv loss 0.011123518620043514
test loss 0.008197434380798377
```

In [11]:

```
# Training a hyper-parameter tuned Xg-Boost regressor on our train data

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

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

x_cfl=XGBClassifier(n_estimators=1000,subsample=0.5,learning_rate=0.15,colsample_bytree=0.5
x_cfl.fit(X_train_asm,y_train_asm)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_train_asm,y_train_asm)

predict_y = c_cfl.predict_proba(X_train_asm)
print ('train loss',log_loss(y_train_asm, predict_y))
predict_y = c_cfl.predict_proba(X_cv_asm)
print ('cv loss',log_loss(y_cv_asm, predict_y))
predict_y = c_cfl.predict_proba(X_test_asm)
print ('test loss',log_loss(y_test_asm, predict_y))
```

```
train loss 0.008256678409115094
cv loss 0.011090946258662731
test loss 0.008196739825909375
```

```
# Xgboost with asm image features gives us the best log loss 0.008
```