Microsoft Malware Detection

December 30, 2018

```
In [1]: # Segregating .asm and .byte files
        import os
        import shutil

        files = os.listdir('./train')

        for f in files:
            if os.path.splitext(f)[1] == '.asm':
                shutil.move('./train/'+f,'./asm')
            else:
                shutil.move('./train/'+f,'./byte')
```

```
In [4]: import os
        files = os.listdir('./asm')
        # list of all keywords like 'mov', 'pop' ,'rdata' etc
        f = open('./key_words.txt','r')
        op = f.read().split('\n')
        op.pop(-1)
        f.close()
        op = ' '.join(op)
        op.replace('\t','')
        op = op.lower().split()
```

```
In [2]: len(op)

Out[2]: 397
```

```
In [3]: len(files)

Out[3]: 10868
```

extraction of data was done using flashtext which uses trie algorithm

```
In [5]: from flashtext import KeywordProcessor

        kp = KeywordProcessor()
        for w in op:
            kp.add_keyword(w.lower())
```

```
# returns key values pairs of words and their counts
def extract(file):
    f = open('./asm/'+file, 'rb')
    name = file.split('.')[0]
    data = str(f.read())
    f.close()
    text = kp.extract_keywords(data)
    pairs = dict(zip(op,[0]*len(op)))
    for w in text:
        try:pairs[w]+=1
        except:pass
    return name,pairs,' '.join(text)
```

In [22]:
```
from joblib import Parallel as p, delayed as jdl
```

In [7]:
```
%%time
rows = p(n_jobs=-1)(jdl(extract)(f) for f in files)
```

```
CPU times: user 3min 10s, sys: 34 s, total: 3min 44s
Wall time: 1h 18min 8s
```

In [8]:
```
len(rows)
```

Out[8]:
```
10868
```

In [9]:
```
from sys import getsizeof as size
```

In [10]:
```
# in bytes
size(rows)
```

Out[10]:
```
87624
```

In [11]:
```
# saving processed data to disk
from pickle import dump

with open('rows.pkl','wb') as f:
    dump(rows,f)
```

In [12]:
```
from pandas import DataFrame
df= DataFrame()
names=[]
for i in range(len(rows)):
    df = df.append(rows[i][1],ignore_index=True)
    names.append(rows[i][0])

df['Id']=names
df.head()
```

```
Out[12]:        00       01       02       03      04      05      06      07      08  \
         0  38425.0  23642.0  23361.0  21382.0  2886.0  2006.0  2309.0  1936.0  2828.0
         1   1381.0     24.0     15.0     18.0    44.0    12.0    10.0     9.0    15.0
         2  15859.0    942.0    653.0    471.0   865.0   579.0   516.0   419.0   887.0
         3   8081.0   1259.0    439.0    477.0  1083.0   285.0   479.0   370.0  2502.0
         4   1697.0     17.0     14.0     14.0    19.0    15.0    11.0    10.0    17.0

                09          ...            stosb  stosw    sub    test     text  tls  \
         0  2059.0          ...              0.0    0.0  280.0   494.0  14751.0  0.0
         1     4.0          ...              0.0    0.0   23.0     1.0   1013.0  0.0
         2   355.0          ...              0.0    0.0  249.0   205.0  11328.0  0.0
         3   184.0          ...              5.0    3.0  511.0  1319.0  67136.0  0.0
         4     9.0          ...              0.0    0.0   55.0     6.0   2116.0  0.0

            xchg  xlatb    xor                    Id
         0   0.0    0.0  409.0  acxojmFTMUAL2HuNfeQd
         1   1.0    0.0   44.0  cqHlrY9oAVpyWMKJ8mOF
         2   0.0    0.0  230.0  FmUz8pwNlXgbS7DW5yre
         3   1.0    0.0  635.0  AywPluRjT8DYXBFS7m2h
         4   0.0    0.0   47.0  5mr4z8KW9nvdyVEY301J

         [5 rows x 396 columns]
```

```
In [30]: feats = list(df.columns)
         feats.pop(-1)# poping 'Id' column
         len(feats)
```

```
395
```

```
In [17]: # this idea was mentioned by the first prize winner of this competition,
         # i.e keep those features(opcodes and segment codes)
         # that occur more than 200 times atleast in one file
         # they found around 165 1-gram features with this

         reduced_feats = [f for f in feats if (df[f]>200).any()]
         len(reduced_feats)
```

```
Out[17]: 344
```

```
In [18]: df[reduced_feats].head()
```

```
Out[18]:        00       01       02       03      04      05      06      07      08  \
         0  38425.0  23642.0  23361.0  21382.0  2886.0  2006.0  2309.0  1936.0  2828.0
         1   1381.0     24.0     15.0     18.0    44.0    12.0    10.0     9.0    15.0
         2  15859.0    942.0    653.0    471.0   865.0   579.0   516.0   419.0   887.0
         3   8081.0   1259.0    439.0    477.0  1083.0   285.0   479.0   370.0  2502.0
         4   1697.0     17.0     14.0     14.0    19.0    15.0    11.0    10.0    17.0
```

```
       09  ...    shl   shr  stc   std    sub    test     text  tls  xchg  \
0  2059.0  ...   84.0  35.0  0.0  78.0  280.0   494.0  14751.0  0.0   0.0
1     4.0  ...    0.0   0.0  0.0   0.0   23.0     1.0   1013.0  0.0   1.0
2   355.0  ...    0.0   0.0  0.0   0.0  249.0   205.0  11328.0  0.0   0.0
3   184.0  ...   32.0  15.0  0.0   0.0  511.0  1319.0  67136.0  0.0   1.0
4     9.0  ...    0.0   0.0  0.0   0.0   55.0     6.0   2116.0  0.0   0.0

     xor
0  409.0
1   44.0
2  230.0
3  635.0
4   47.0

[5 rows x 344 columns]
```

In [21]:
```
red_df = df[reduced_feats+['Id']]
red_df.to_csv('red_df.csv')
```

In [16]:
```
from pandas import read_csv
red_df = read_csv('red_df.csv').drop(columns='Unnamed: 0')
labels = read_csv('trainLabels.csv')
labels.head()
```

Out[16]:
```
                     Id  Class
0  01kcPWA9K2BOxQeS5Rju      1
1  04EjIdbPV5e1XroFOpiN      1
2  05EeG39MTRrI6VY21DPd      1
3  05rJTUWYAKNegBk2wE8X      1
4  0AnoOZDNbPXIr2MRBSCJ      1
```

In [17]:
```
from pandas import merge
final_df = merge(red_df,labels,on='Id')
final_df.shape
```

Out[17]:
```
(10868, 346)
```

In [18]:
```
final_df.head()
```

Out[18]:
```
        00       01       02       03      04      05      06      07      08  \
0  38425.0  23642.0  23361.0  21382.0  2886.0  2006.0  2309.0  1936.0  2828.0
1   1381.0     24.0     15.0     18.0    44.0    12.0    10.0     9.0    15.0
2  15859.0    942.0    653.0    471.0   865.0   579.0   516.0   419.0   887.0
3   8081.0   1259.0    439.0    477.0  1083.0   285.0   479.0   370.0  2502.0
4   1697.0     17.0     14.0     14.0    19.0    15.0    11.0    10.0    17.0

       09  ...  stc   std    sub   test     text  tls  xchg    xor  \
0  2059.0  ...  0.0  78.0  280.0  494.0  14751.0  0.0   0.0  409.0
1     4.0  ...  0.0   0.0   23.0    1.0   1013.0  0.0   1.0   44.0
```

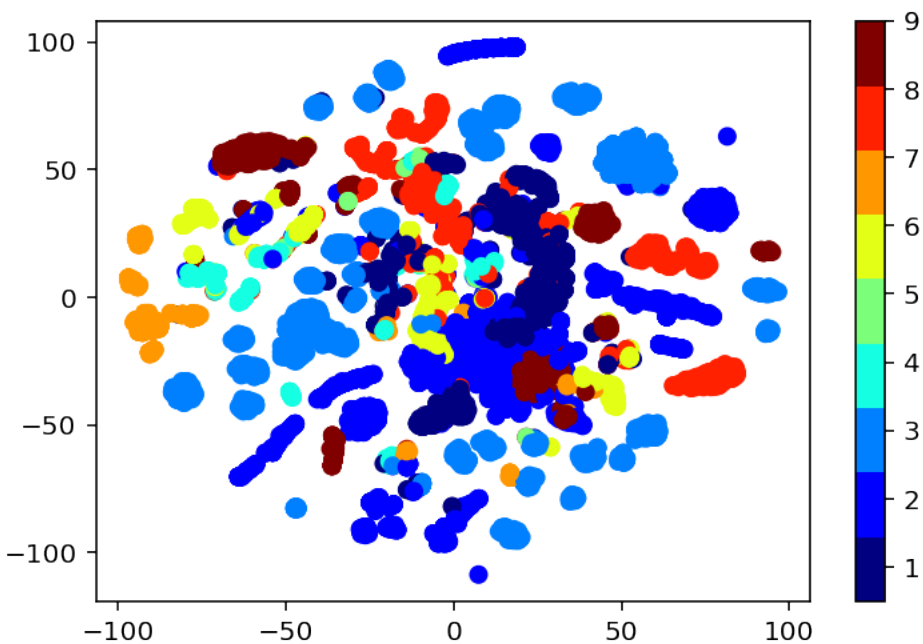
```
2    355.0  ...    0.0    0.0  249.0   205.0  11328.0  0.0    0.0  230.0
3    184.0  ...    0.0    0.0  511.0  1319.0  67136.0  0.0    1.0  635.0
4      9.0  ...    0.0    0.0   55.0     6.0   2116.0  0.0    0.0   47.0

                     Id  Class
0  acxojmFTMUAL2HuNfeQd      2
1  cqHlrY9oAVpyWMKJ8mOF      3
2  FmUz8pwNlXgbS7DW5yre      9
3  AywPluRjT8DYXBFS7m2h      1
4  5mr4z8KW9nvdyVEY301J      3

[5 rows x 346 columns]
```

In [19]:
```
final_df.to_csv('fin_df.csv')
```

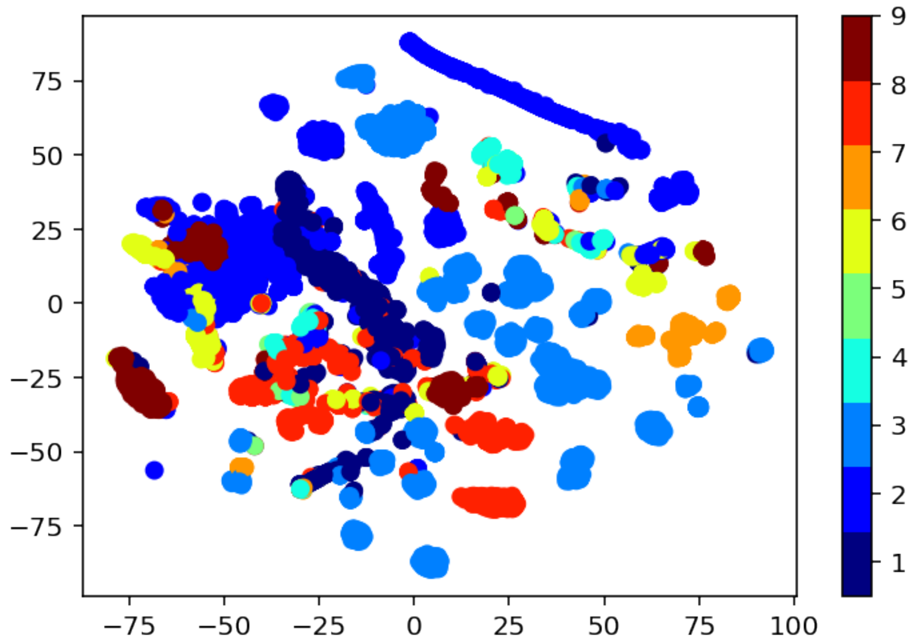
In [40]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

0.1 *tSNE visualization of 1 grams features:*

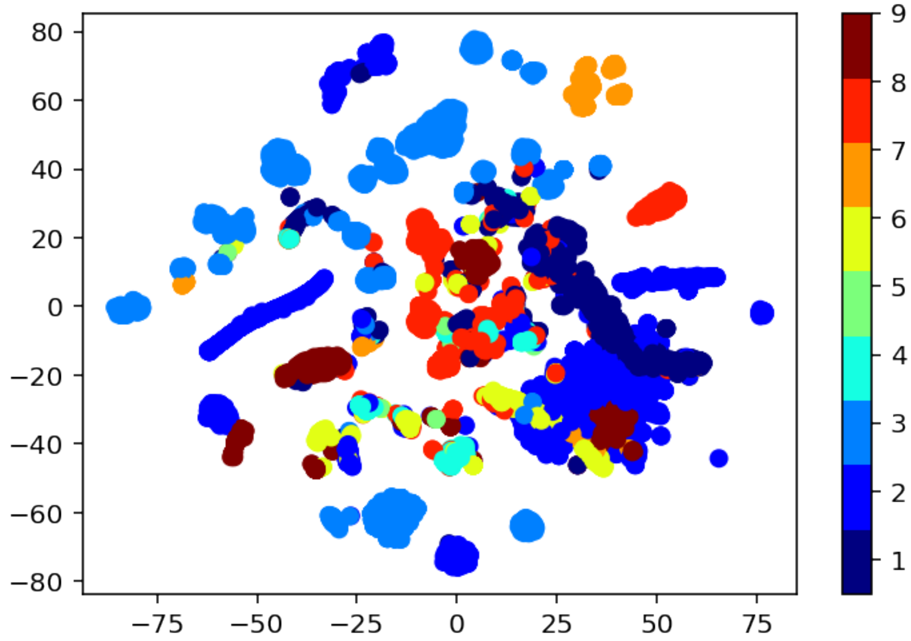
In [26]:
```
xtsne=TSNE(perplexity=30)
results=xtsne.fit_transform(final_df.drop(['Id','Class'], axis=1))
vis_x = results[:, 0]
vis_y = results[:, 1]
plt.scatter(vis_x, vis_y, c=final_df['Class'], cmap=plt.cm.get_cmap("jet", 9))
plt.colorbar(ticks=range(10))
plt.clim(0.5, 9)
plt.show()
```



```
In [27]: xtsne=TSNE(perplexity=50)
         results=xtsne.fit_transform(final_df.drop(['Id','Class'], axis=1))
         vis_x = results[:, 0]
         vis_y = results[:, 1]
         plt.scatter(vis_x, vis_y, c=final_df['Class'], cmap=plt.cm.get_cmap("jet", 9))
         plt.colorbar(ticks=range(10))
         plt.clim(0.5, 9)
         plt.show()
```




```
In [28]: xtsne=TSNE(perplexity=70)
         results=xtsne.fit_transform(final_df.drop(['Id','Class'], axis=1))
         vis_x = results[:, 0]
         vis_y = results[:, 1]
         plt.scatter(vis_x, vis_y, c=final_df['Class'], cmap=plt.cm.get_cmap("jet", 9))
         plt.colorbar(ticks=range(10))
         plt.clim(0.5, 9)
         plt.show()
```



In [1]:
```
from pickle import load

with open('rows.pkl','rb') as f:
    rows=load(f)

f.close()
Id, counts , text = zip(*rows)
```

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [4]:
```
from numpy import int32,int16
```

In [ ]:
```
# min_df , ignores those terms/words which doesn't occur atleast 200 times
# first place solution got nearly 70K+ featured ngrams of most frequent opcodes and segm
```

In [17]:
```
%%time
# I'll just take 30K features for 2 grams

cnt_2g = CountVectorizer(ngram_range=(2,2),min_df=200,max_features=30000,dtype=int32)
twograms = cnt_2g.fit_transform(text)
```

```
CPU times: user 2h 6min 37s, sys: 1min 46s, total: 2h 8min 24s
Wall time: 2h 8min 21s
```

In [18]:
```
twograms.shape
```

```
Out[18]: (10868, 30000)
```

```
In [20]: from scipy.sparse import save_npz
         save_npz('./twograms.npz',twograms)
```

```
In [5]: %%time
        # I'll just take 25k features for 3 grams

        cnt_3g = CountVectorizer(ngram_range=(3,3),min_df=200,max_features=25000,dtype=int32)
        threegrams = cnt_3g.fit_transform(text)

CPU times: user 2h 26min 31s, sys: 2min 5s, total: 2h 28min 37s
Wall time: 2h 28min 37s
```

```
In [6]: from scipy.sparse import save_npz
        save_npz('./threegrams.npz',threegrams)
```

0.2 *extracting features from ASM images:*

```
In [1]: import os
        import numpy as np
        from imageio import imwrite
        from array import array as arr
        from joblib import Parallel as p, delayed as jdl
```

```
In [2]: # code derived from here,
        # https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/
        # /Saynotooverfitting.pdf
        asm = os.listdir('./asm')

        def asm_pixel(af):
            f = open('./asm/'+af,'rb')
            ln = os.path.getsize('./asm/'+af)
            width = int(ln**.5)
            rem = ln%width
            a = arr('B')
            a.fromfile(f,ln-rem)
            f.close()
            g = np.reshape(a,(len(a)//width,width))
            g = np.uint8(g)
            return af.split('.')[0],g.ravel()[:1000]
```

```
In [9]: %%time
        pixel_features = p(n_jobs=22,backend='multiprocessing')(jdl(asm_pixel)(fil) for fil in a

CPU times: user 5.02 s, sys: 1.3 s, total: 6.32 s
Wall time: 26min 45s
```

```
In [20]: Id , pix = zip(*pixel_features)
```

```
In [39]: labels = read_csv('trainLabels.csv')
         red_df = read_csv('red_df.csv').drop(columns=['Unnamed: 0'])
         final_df = merge(red_df,labels,on='Id')
```

```
In [26]: from scipy.sparse import load_npz

         twograms = load_npz('twograms.npz')
         threegrams = load_npz('threegrams.npz')

         twograms.shape,threegrams.shape

Out[26]: ((10868, 30000), (10868, 25000))
```

```
In [43]: from scipy.sparse import csr_matrix as csr

         data = csr(final_df.drop(columns=['Id','Class']).values)
         pixels = csr(np.array(pix))

         data.shape,pixels.shape

Out[43]: ((10868, 344), (10868, 1000))
```

```
In [44]: from scipy.sparse import hstack

         ngrams = hstack((data,twograms,threegrams))
         ngrams.shape

Out[44]: (10868, 55344)
```

```
In [45]: from scipy.sparse import save_npz
         save_npz('./ngrams.npz',ngrams)
         save_npz('./pixels.npx',pixels)
```

```
In [1]: from scipy.sparse import load_npz,save_npz,hstack,vstack
        ngrams = load_npz('./ngrams.npz').tocsr()
        pixels = load_npz('./pixels.npz').tocsr()
```

```
In [2]: ngrams.shape,pixels.shape

Out[2]: ((10868, 55344), (10868, 1000))
```

```
In [4]: from pandas import read_csv
        final_df = read_csv('fin_df.csv').drop(columns='Unnamed: 0')
        final_df.shape

Out[4]: (10868, 346)
```

0.3 *Feature Selection by RandomForest:*

```
In [6]: from sklearn.ensemble import RandomForestClassifier
        import numpy as np
```

```
In [21]: from sklearn.utils.class_weight import compute_class_weight
         cls_wt = compute_class_weight('balanced',list(range(1,10)),final_df.Class.values)
         cls_wt = dict(zip(list(range(1,10)),cls_wt))
         cls_wt
```

```
Out[21]: {1: 0.7836181411781671,
          2: 0.48731055510716526,
          3: 0.4104539617795906,
          4: 2.542222222222222,
          5: 28.75132275132275,
          6: 1.6079301671844948,
          7: 3.0340591848129534,
          8: 0.9833514296055013,
          9: 1.192058791269058}
```

```
In [27]: rf = RandomForestClassifier(n_jobs=-1,warm_start=True,class_weight=cls_wt)
```

```
In [28]: # using warm start to make use of whole data while sampling data points as well.
         from sklearn.model_selection import train_test_split
         y = final_df.Class
         l = list(range(1,5))[::-1]
         for i in l:
             x_trn, x_tst, y_trn, y_tst = train_test_split(ngrams, y,stratify=y ,test_size=i/10)
             rf.fit(x_trn,y_trn)
```

```
/usr/local/lib/python3.5/site-packages/sklearn/ensemble/forest.py:305: UserWarning: Warm-start f
  warn("Warm-start fitting without increasing n_estimators does not "
```

```
In [58]: top_feat_indices = np.argpartition(f, -3000)[-3000:]
         indices = np.sort(top_feat_indices)
```

```
In [64]: reduced_ngrams = ngrams[:,indices]
         reduced_ngrams.shape
```

```
Out[64]: (10868, 3000)
```

```
In [65]: # stacking ngram features and asm pixels features
         ngrm_pix = hstack((reduced_ngrams,pixels))
         ngrm_pix.shape
```

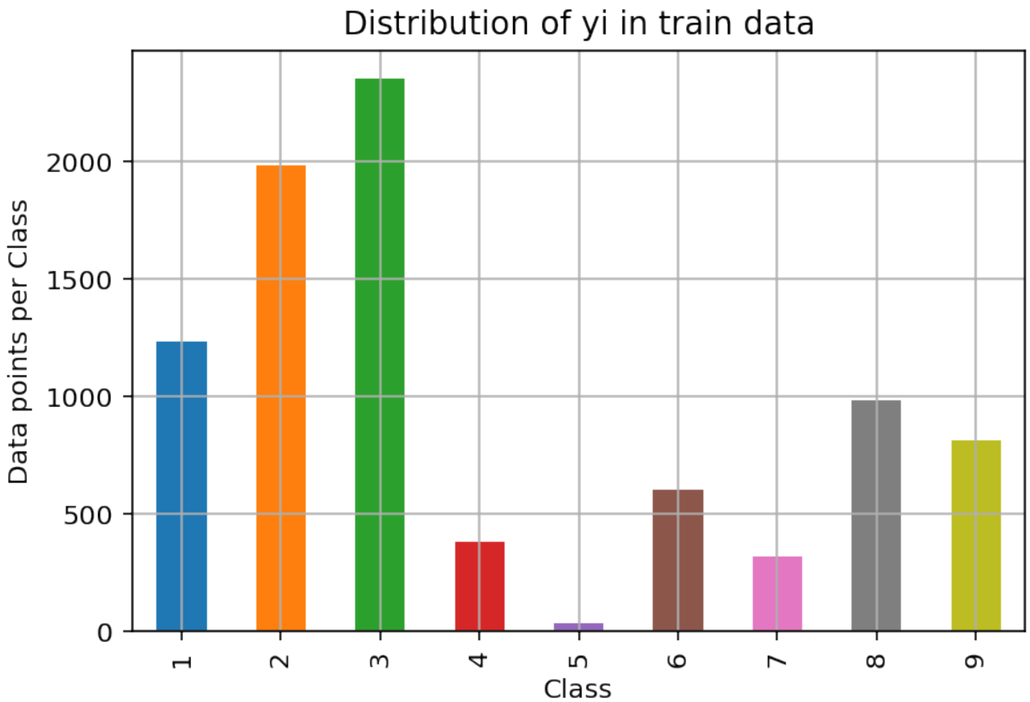
```
Out[65]: (10868, 4000)
```

0.4 *Train Test Split:*

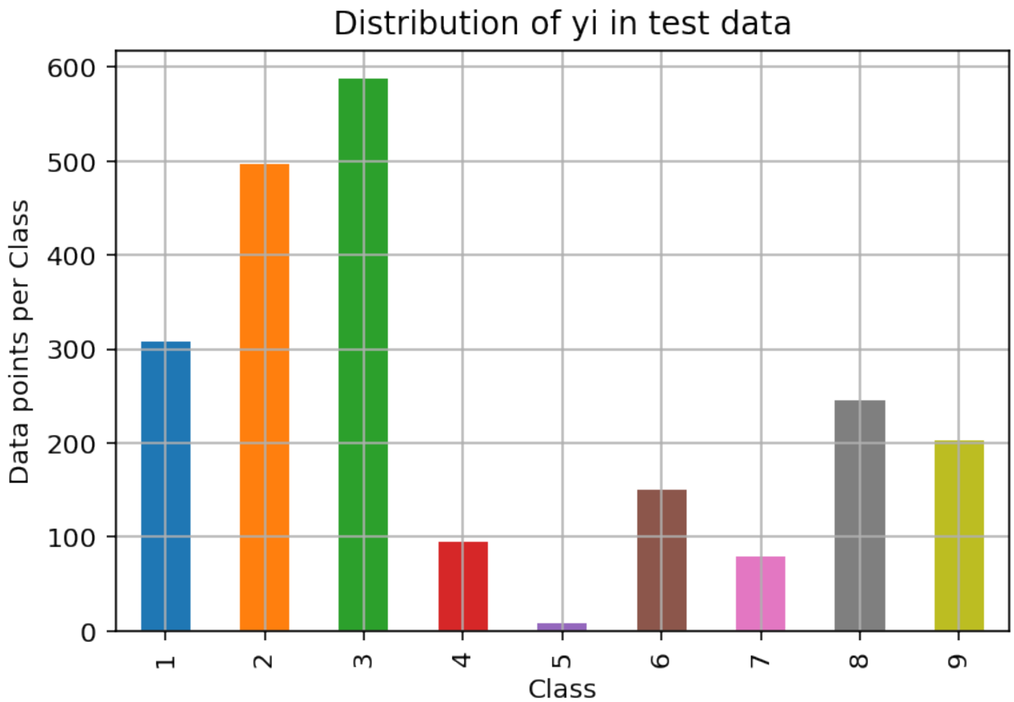
In [66]:
```
# stratify: to maintain same distribution
x_train, x_test, y_train, y_test = train_test_split(ngrm_pix, y,stratify=y,test_size=0.

print('Number of data points in train data:', x_train.shape[0])
print('Number of data points in test data:', x_test.shape[0])
```

```
Number of data points in train data: 8694
Number of data points in test data: 2174
```

In [67]:
```
# 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()

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

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)

test_class_distribution.plot(kind='bar')
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

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],
```




```
Number of data points in class 3 : 2354 ( 27.076 %)
Number of data points in class 2 : 1982 ( 22.797 %)
Number of data points in class 1 : 1233 ( 14.182 %)
Number of data points in class 8 : 982 ( 11.295 %)
Number of data points in class 9 : 810 ( 9.317 %)
Number of data points in class 6 : 601 ( 6.913 %)
Number of data points in class 4 : 380 ( 4.371 %)
Number of data points in class 7 : 318 ( 3.658 %)
Number of data points in class 5 : 34 ( 0.391 %)
--------------------------------------------------------------------------------
```




```
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 %)
```

0.5 *Modeling:*

```
In [44]: import seaborn as sns
         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)

             A =(((C.T)/(C.sum(axis=1))).T)

             B =(C/C.sum(axis=0))

             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 recall matrix",A.sum(axis=1))
```

In [45]:
```
from lightgbm import LGBMClassifier as lgb
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import log_loss,confusion_matrix
```

In [46]:
```
def frange(start, end, step=1.0,rnd=3):
    while start < end:
        yield round(start,rnd)
        start += step
```

In [69]:
```
params={
    'learning_rate':list(frange(0.001,0.3,.01)),
    'num_leaves':list(range(10,120)),
    'n_estimators':list(range(30,150)),
    'min_child_weight':list(frange(0.0001,0.01,.0001,rnd=4)),
    'reg_lambda':list(frange(0.0001,0.03,.0001,rnd=5)),
    'colsample_bytree':list(frange(0.1,1,.1)),
    'subsample':list(frange(0.1,1,.1))
}
```

In [50]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [70]:
```
rndcv = RandomizedSearchCV(lgb(objective='multi:softprob',class_weight='balanced'),cv=4
                           param_distributions=params,scoring='log_loss',n_jobs=10)
rndcv.fit(x_train,y_train)
-rndcv.best_score_
```

Out[70]: 0.01081910286050876

In [71]:
```
rndcv.best_params_
```

Out[71]:
```
{'colsample_bytree': 0.7,
 'learning_rate': 0.121,
 'min_child_weight': 0.0072,
 'n_estimators': 82,
 'num_leaves': 52,
 'reg_lambda': 0.0106,
 'subsample': 0.9}
```

In [74]:
```
clf = lgb(objective='multi:softprob',
          class_weight='balanced',
          colsample_bytree= 0.7,
          learning_rate= 0.121,
          min_child_weight= 0.0072,
          n_estimators= 82,
          num_leaves= 52,
          reg_lambda= 0.0106,
          subsample= 0.9)
clf.fit(x_train,y_train)
c_cfl=CalibratedClassifierCV(clf,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_test)
print ('test loss',log_loss(y_test, predict_y))
```

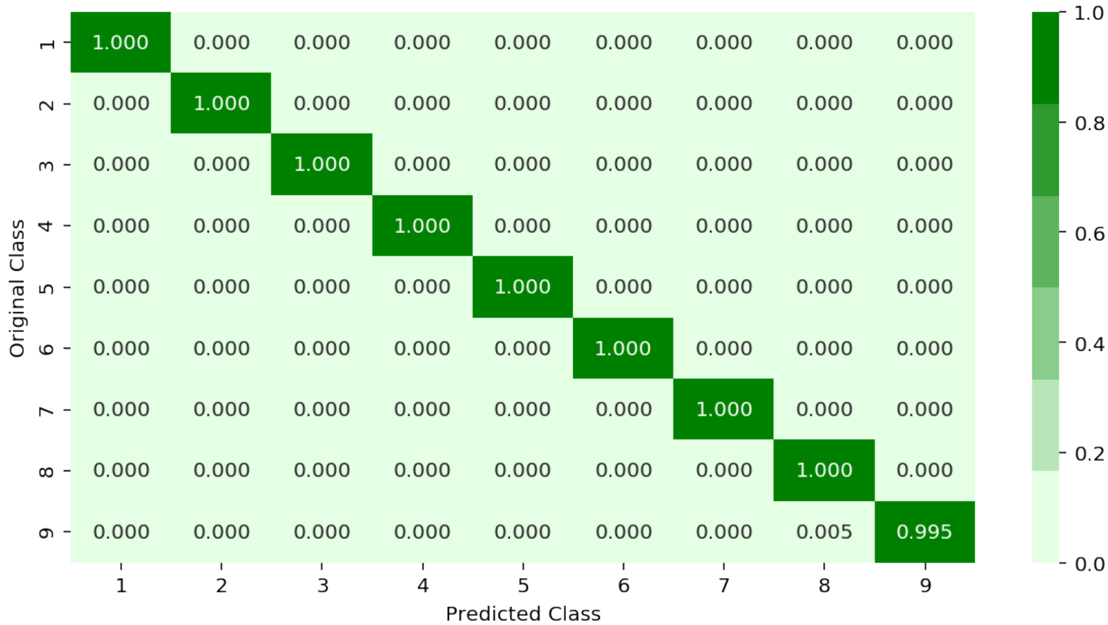
```
train loss 0.007660557247164744
test loss 0.009122368282205633
```

In [77]:
```
plot_confusion_matrix(y_test,c_cfl.predict(x_test))
```

```
Number of misclassified points  0.045998160073597055
-------------------------------------------------- Confusion matrix ----------------------------
```


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


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




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

0.6 *Conclusion:*

```
I followed the approach of first place winner(xiaozhouwang's team)

the things I did to achieve the above score are,
    1. used some opcodes, segement codes which seemed important
       like 'mov', 'pop', 'rdata', '00' , 'A9' etc.
    2. counted 1-grams, 2-grams, 3-grams from those words, tried to count 4-grams,
       but it was consuming too much of RAM so I had to leave 4-grams.
    3. converted .asm files to images and picked first 1000 pixel intensities.
    4. used randomforest to select 3000 features out of all ngram features.
       If I took too few features from ngrams models where overfitting.
    5. finally,tuned LightGBMClassifier using RandomizedSearchCV.
       and fit a calibrated model on top of it which gave,
       train loss: 0.007660557247164744
       test loss:  0.009122368282205633
```