Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context: Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462

Problem statement : Classify the given genetic variations/mutations based on evidence from text-based clinical liter

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-us/#2a44ee2f6b25
2. https://www.youtube.com/watch?v=UwbuW7oK8rk
3. https://www.youtube.com/watch?v=qxXRKVompI8

1.3. Real-world/Business objectives and constraints.

- No low-latency requirement.
- Interpretability is important.
- Errors can be very costly.
- Probability of a data-point belonging to each class is needed

2.1. Data

2.1.1. Data Overview

- Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/data
- We have two data files: one conatins the information about the genetic mutations and the other contains the experts/pathologists use to classify the genetic mutations.
- Both these data files are have a common column called ID

- Data file's information:
  - training_variants (ID , Gene, Variations, Class)
  - training_text (ID, Text)

2.1.2. Example Data Point

training_variants

ID,Gene,Variation,Class

0,FAM58A,Truncating Mutations,1

1,CBL,W802*,2

2,CBL,Q249E,2 ...

training_text

ID,Text

0||Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one
cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases
homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The
ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cycl
an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome,
features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome
interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxife
CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome
derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results rev
ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlyi
(CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome cont
considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to b
almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap
biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor ir
ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity throu
th

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

2.2.1. Type of Machine Learning Problem

There are nine different classes a genetic mutation can be classified into => Multi class classification problem

2.2.2. Performance Metric

Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment#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:

- Interpretability
- Class probabilities are needed.
- Penalize the errors in class probabilites => Metric is Log-loss.
- No Latency constraints.

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
#from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
```



```
from sklearn.model_selection import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
#from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier
```

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

```
Number of data points :  3321
Number of features :  4
Features :  ['ID' 'Gene' 'Variation' 'Class']
```

	ID	Gene	Variation	Class
0	0	FAM58A	Truncating Mutations	1
1	1	CBL	W802*	2
2	2	CBL	Q249E	2
3	3	CBL	N454D	3
4	4	CBL	L399V	4

training/training_variants is a comma separated file containing the description of the genetic mutations used for tra

- ID : the id of the row used to link the mutation to the clinical evidence
- Gene : the gene where this genetic mutation is located
- Variation : the aminoacid change for this mutations
- Class : 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data


```
print('Number of features : ', data_text.shape[1])
print('Features : ', data_text.columns.values)
data_text.head()
```

```
Number of data points :  3321
Number of features :  2
Features :  ['ID' 'TEXT']
```

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...

3.1.3. Preprocessing of text

```
# loading stop words from nltk library
stop_words = set(stopwords.words('english'))


def nlp_preprocessing(total_text, index, column):
    if type(total_text) is not int:
        string = ""
        # replace every special char with space
        total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
        # replace multiple spaces with single space
        total_text = re.sub('\s+',' ', total_text)
        # converting all the chars into lower-case.
        total_text = total_text.lower()

        for word in total_text.split():
        # if the word is a not a stop word then retain that word from the data
            if not word in stop_words:
                string += word + " "

        data_text[column][index] = string


#text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    if type(row['TEXT']) is str:
        nlp_preprocessing(row['TEXT'], index, 'TEXT')
    else:
        print("there is no text description for id:",index)
print('Time took for preprocessing the text :',time.clock() - start_time, "seconds")
```

```
there is no text description for id: 1109
there is no text description for id: 1277
Time took for preprocessing the text : 208.84804406897806 seconds
```



```
#merging both gene_variations and text data based on ID
result = pd.merge(data, data_text,on='ID', how='left')
result.head()
```

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...

```
result[result.isnull().any(axis=1)]
```

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	NaN
1277	1277	ARID5B	Truncating Mutations	1	NaN
1407	1407	FGFR3	K508M	6	NaN
1639	1639	FLT1	Amplification	6	NaN
2755	2755	BRAF	G596C	7	NaN

```
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
result[result['ID']==1109]
```

	ID	Gene	Variation	Class	TEXT
1109	1109	FANCA	S1088F	1	FANCA S1088F

3.1.4. Test, Train and Cross Validation Split

3.1.4.1. Splitting data into train, test and cross validation (64:20:16)

```
y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
```

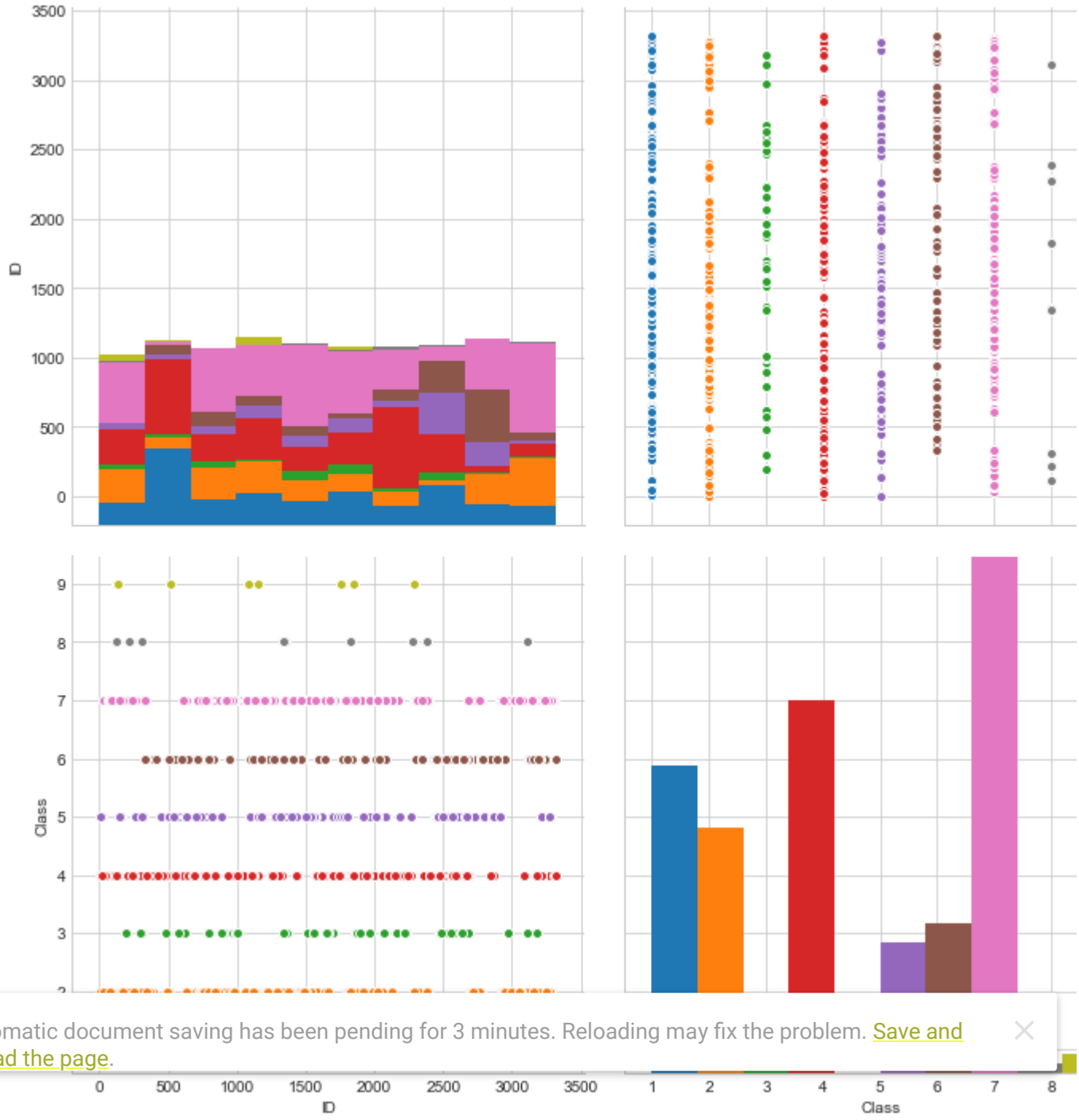


```
print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
Number of data points in train data: 2124
Number of data points in test data: 665
Number of data points in cross validation data: 532
```

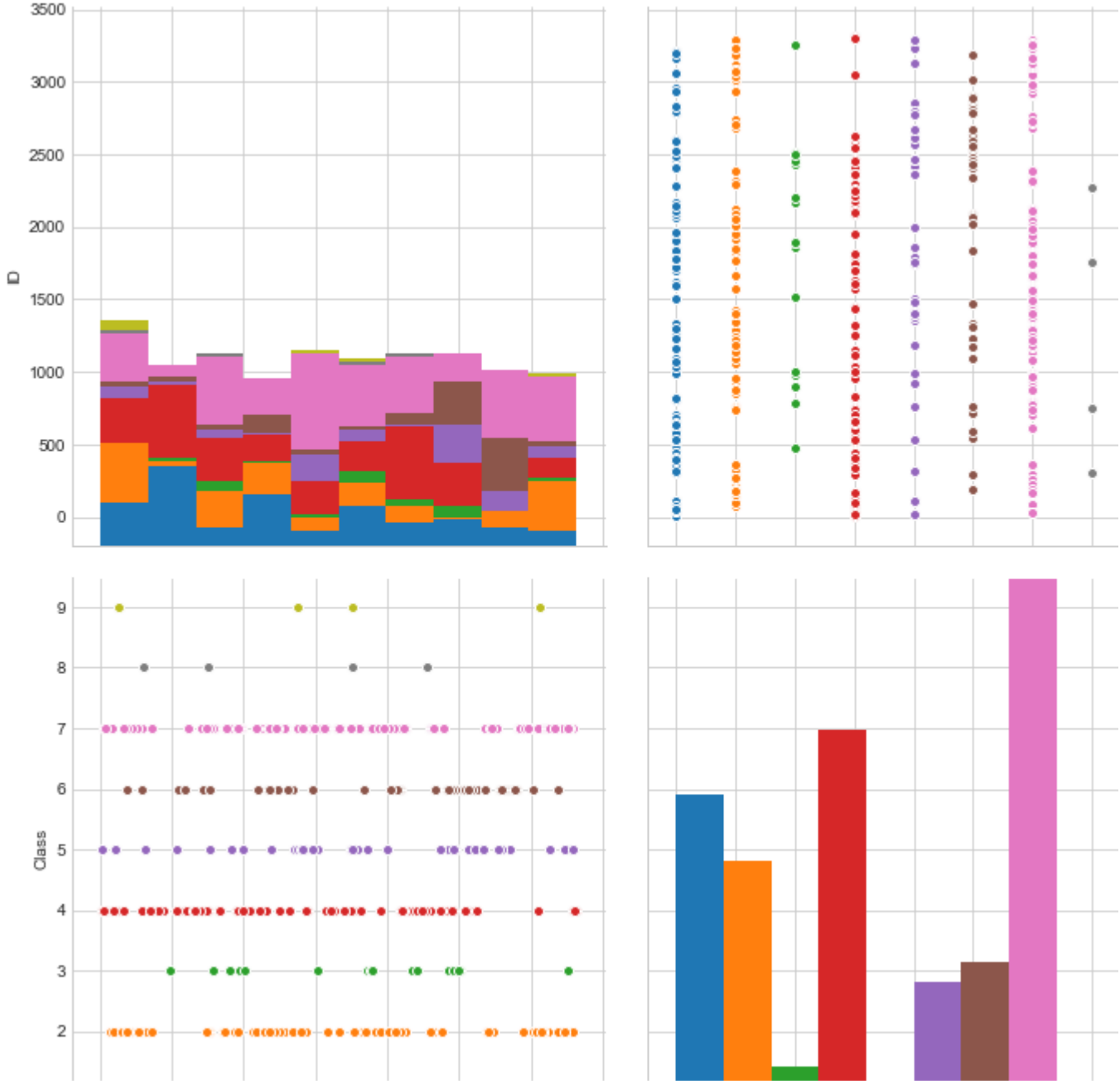
```
#PAIR-PLOTS On train data
plt.close();
sns.set_style("whitegrid");
sns.pairplot(train_df,hue="Class",size=5)
```

```
<seaborn.axisgrid.PairGrid at 0x196a53a4ac8>
```



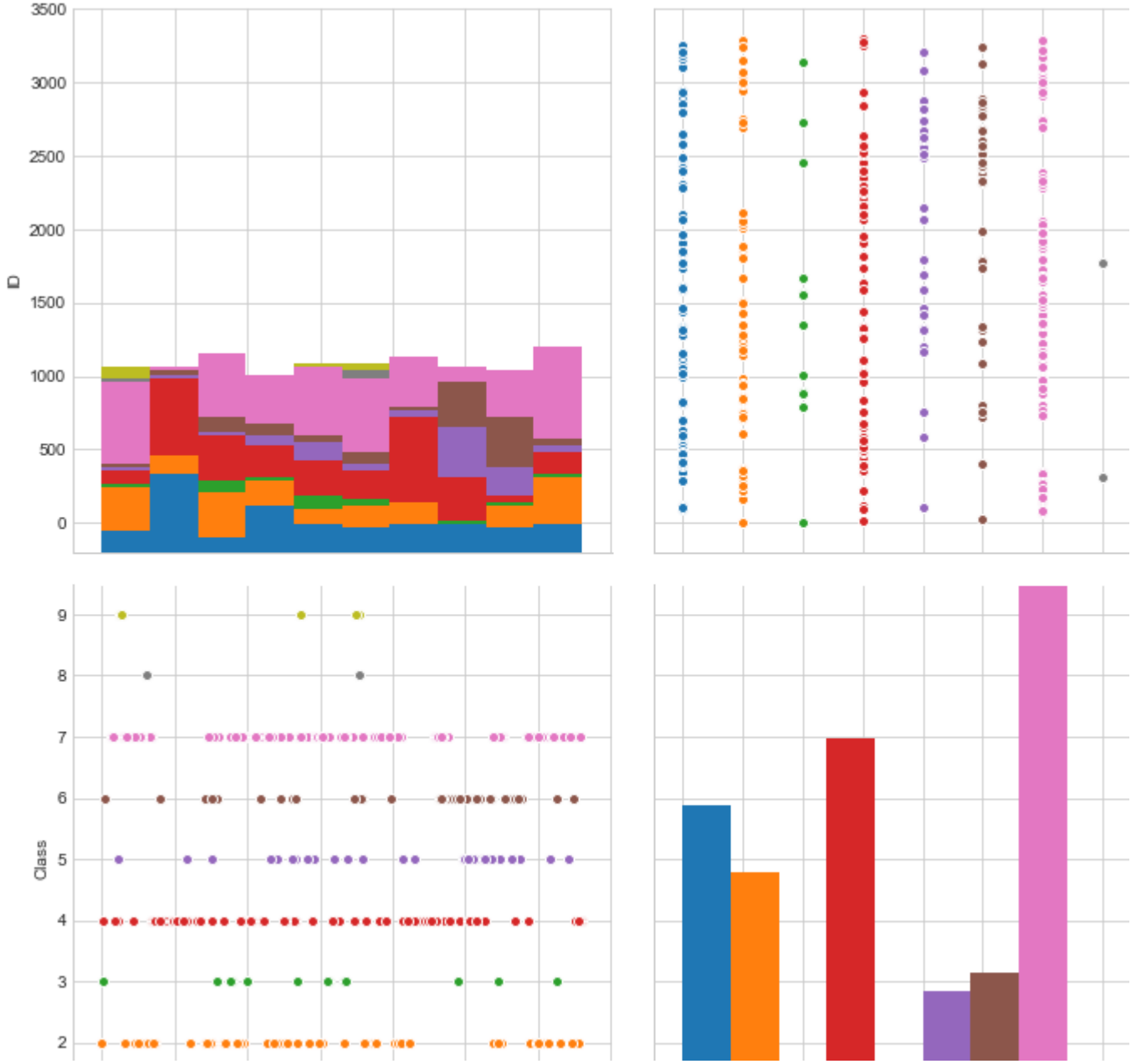

```
#PAIR-PLOTS on test data
plt.close();
sns.set_style("whitegrid");
sns.pairplot(test_df,hue="Class",size=5)
```

```
<seaborn.axisgrid.PairGrid at 0x196c1932588>
```




```
#PAIR-PLOTS on cross validation data
plt.close();
```

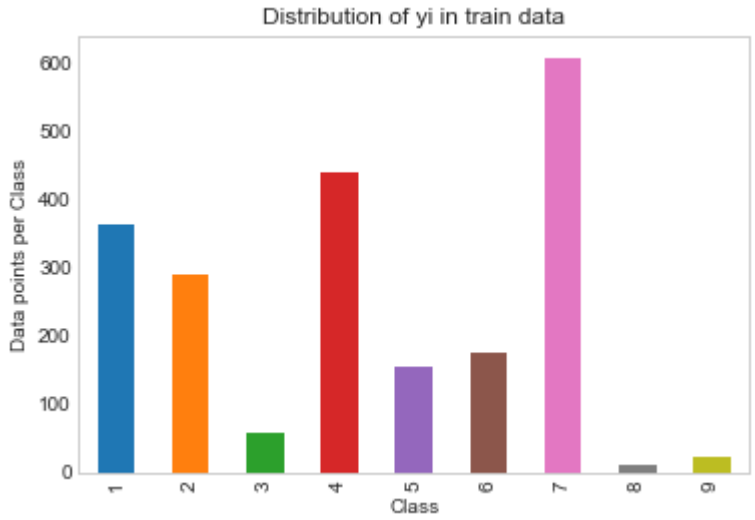
```
<seaborn.axisgrid.PairGrid at 0x196a5461d68>
```



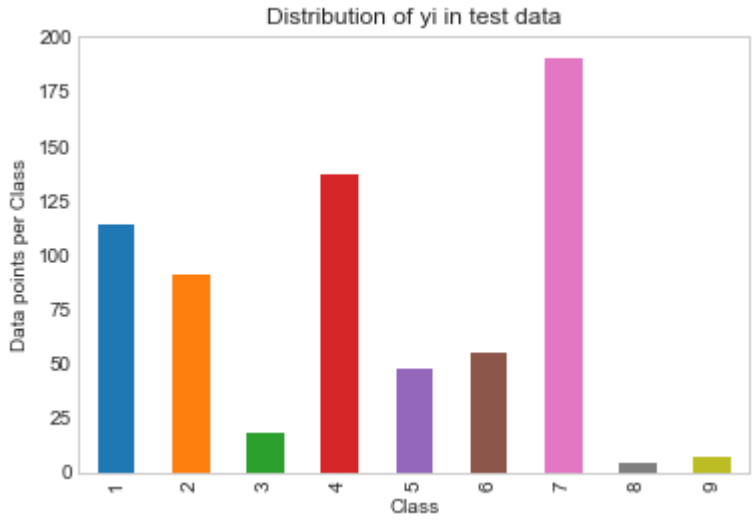
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets


```
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

my_colors = 'rgbkymc'
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()

# 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], '(', np.rou


print('-'*80)
my_colors = 'rgbkymc'
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()

# 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], '(', np.roun

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar')
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], '(', np.round(
```



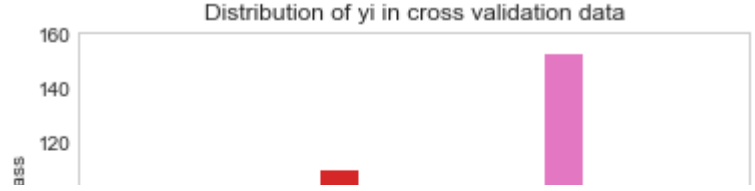

```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
Number of data points in class 3 : 57 ( 2.684 %)
Number of data points in class 9 : 24 ( 1.13 %)
Number of data points in class 8 : 12 ( 0.565 %)
------------------------------------------------------------------------------
```

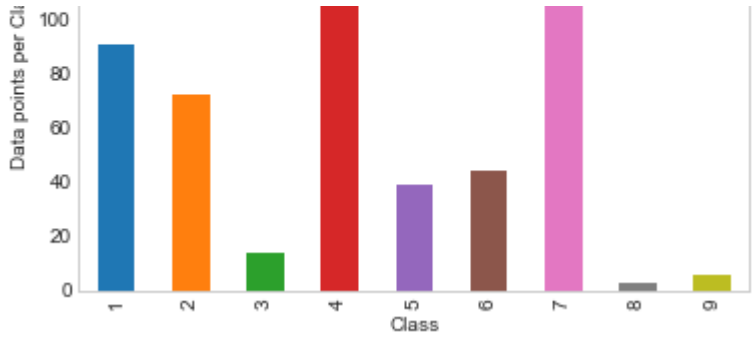



```
Number of data points in class 7 : 191 ( 28.722 %)
```



```
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
------------------------------------------------------------------------------
```






```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
```

```
import pandas as pd

# appending label to the 2d projected data
#new_coordinates = np.vstack((new_coordinates, labels)).T

# creating a new data frame for ploting the labeled points.
dataframe = pd.DataFrame(data=train_df, columns=("1st_principal", "2nd_principal", "label"))
print(dataframe.head())
```

```
      1st_principal  2nd_principal  label
98              NaN            NaN    NaN
2440            NaN            NaN    NaN
2482            NaN            NaN    NaN
1525            NaN            NaN    NaN
2220            NaN            NaN    NaN
```

3.2 Prediction using a 'Random' Model

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j

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

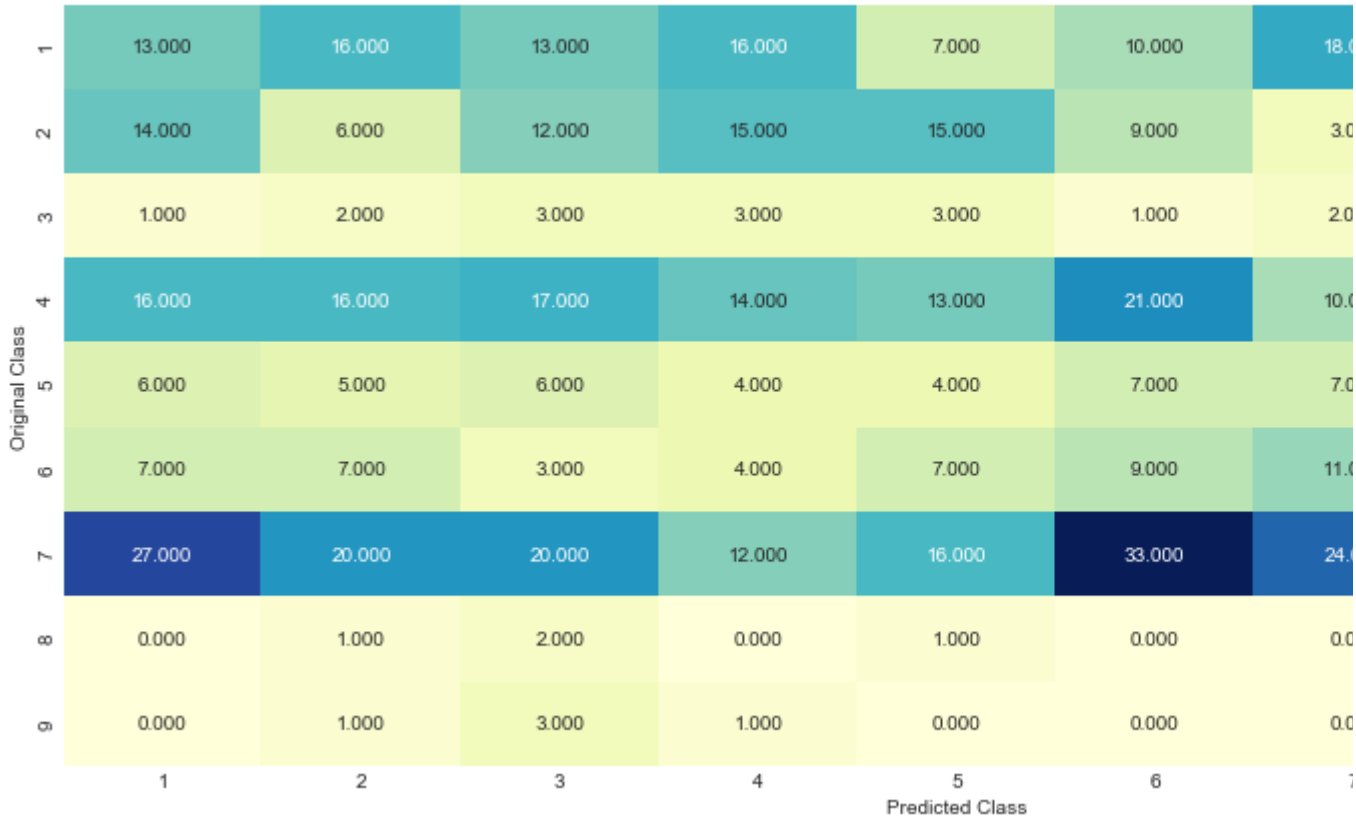


```
    #       [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamension
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((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]]
```

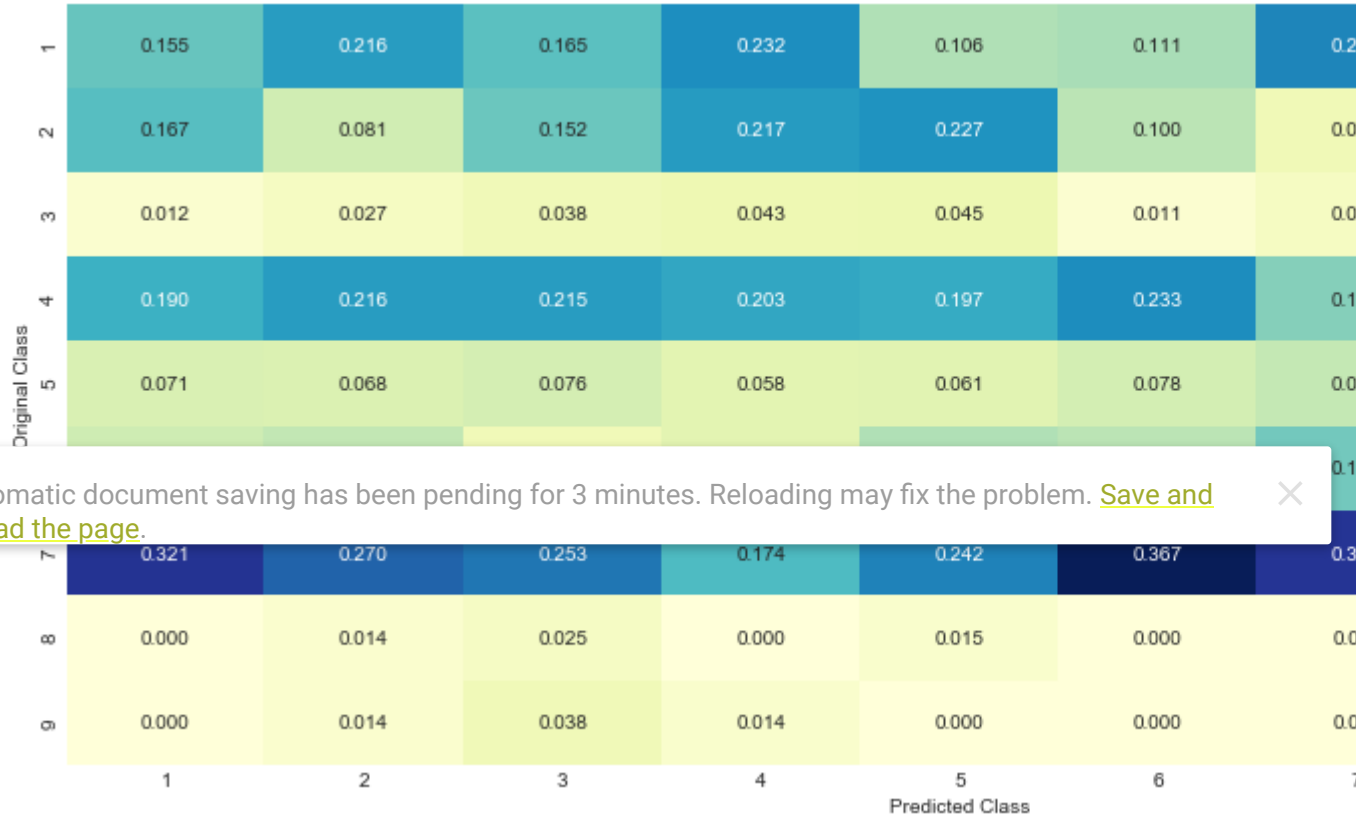
```
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two diamension
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]

    labels = [1,2,3,4,5,6,7,8,9]
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    print("-"*20, "Precision matrix (Columm Sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    # representing B in heatmap format
    print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()
```

```
# 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 = test_df.shape[0]
cv_data_len = cv_df.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_predicted_y, eps=1e-15


# 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=1e-15))
```

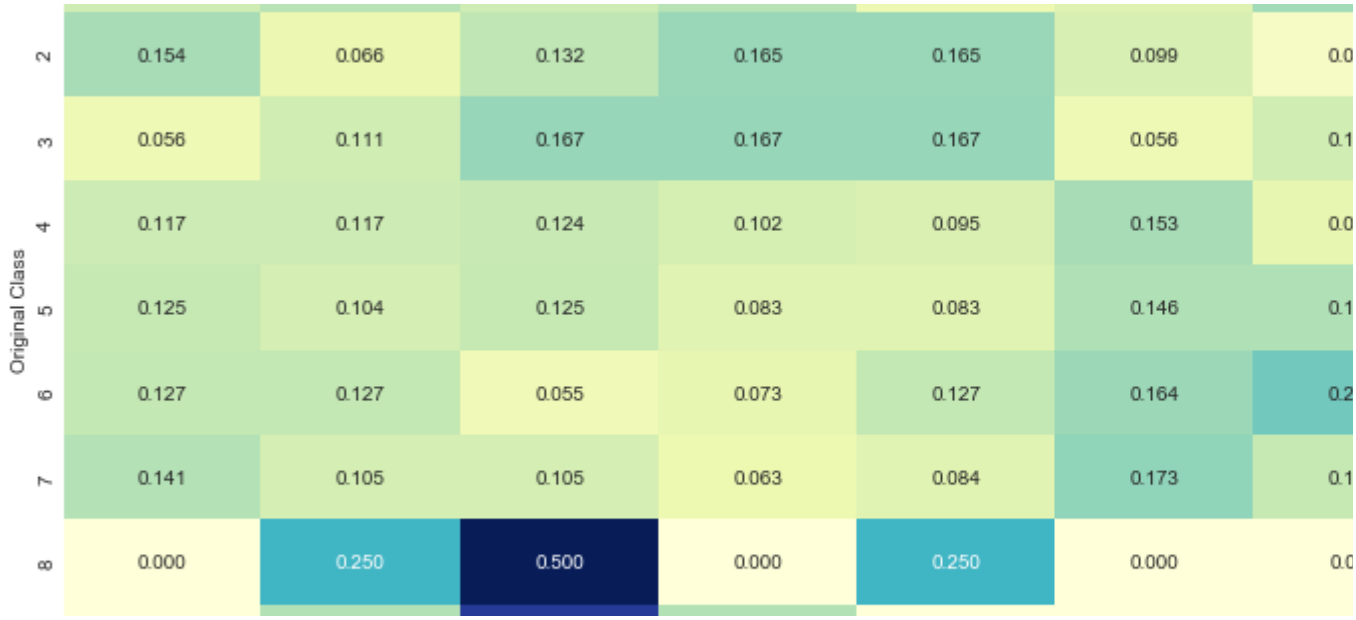
```
Log loss on Cross Validation Data using Random Model 2.501695266541365
Log loss on Test Data using Random Model 2.4868120403912277
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



3.3 Univariate Analysis

```
# code for response coding with Laplace smoothing.
# alpha : used for laplace smoothing
# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# ----------
# Consider all unique values and the number of occurances of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*alpha / numb
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
```

```
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

    # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole data
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticular class
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
            #         ID   Gene             Variation  Class
            # 2470  2470  BRCA1               S1715C      1
            # 2486  2486  BRCA1               S1841R      1
            # 2614  2614  BRCA1                  M1R      1
            # 2432  2432  BRCA1               L1657P      1
            # 2567  2567  BRCA1               T1685A      1
            # 2583  2583  BRCA1               E1660G      1
            # 2634  2634  BRCA1               W1718L      1
            # cls_cnt.shape[0] will return the number of rows

            cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

            # cls_cnt.shape[0](numerator) will contain the number of time that particular feature oc
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

        # we are adding the gene/variation to the dict as key and vec as value
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1363636363636
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177, 0.06818
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078787878787
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284768211920
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733333333333
    #      ...
    #     }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene variation feature, it will contain the feature for each feature value in the data

    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```


3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

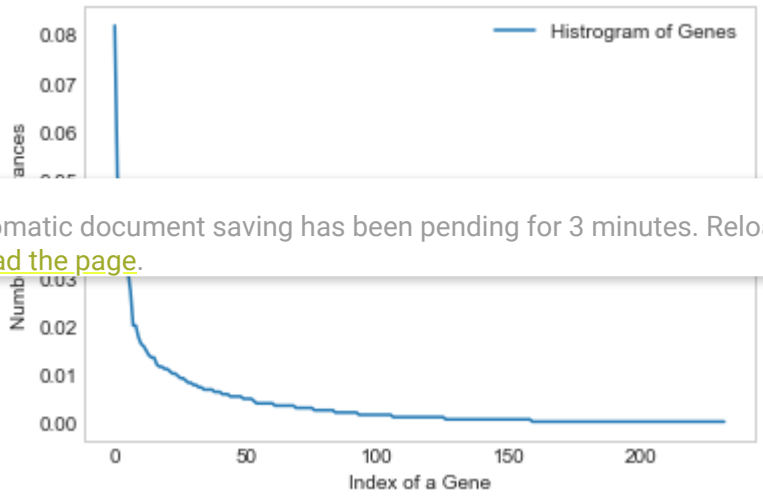
```
unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])
# the top 10 genes that occured most
print(unique_genes.head(10))
```

```
Number of Unique Genes : 233
BRCA1     174
TP53      109
EGFR       86
PTEN       86
BRCA2      82
KIT        67
BRAF       58
ALK        43
ERBB2      43
PDGFRA     38
Name: Gene, dtype: int64
```

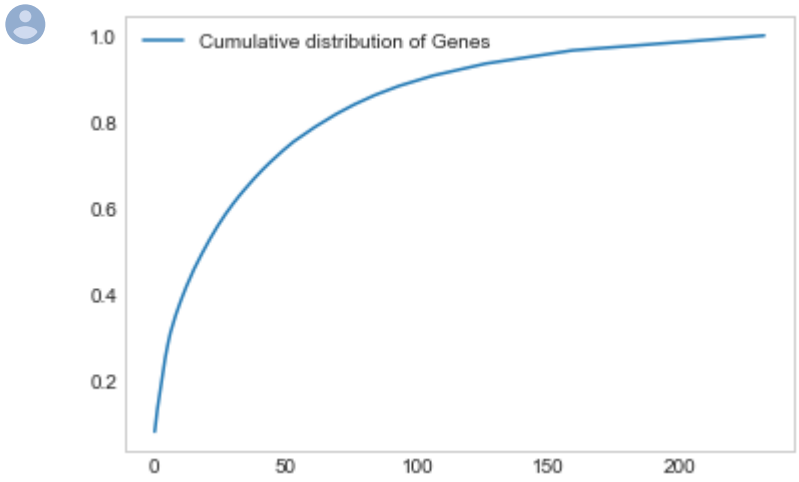
```
print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the train data, and
```

```
Ans: There are 233 different categories of genes in the train data, and they are distibu
```

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
plt.plot(h, label="Histrogram of Genes")
plt.xlabel('Index of a Gene')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



```
c = np.cumsum(h)
plt.plot(c,label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/cours categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classifi


```
#response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
print("train_gene_feature_responseCoding is converted feature using respone coding method. The shape
```

train_gene_feature_responseCoding is converted feature using respone coding method. The

```
# one-hot encoding of Gene feature.
gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
train_df['Gene'].head()
```

```
98      TGFBR2
2440     BRCA1
2482     BRCA1
1525       ALK
2220      PTEN
Name: Gene, dtype: object
```

```
gene_vectorizer.get_feature_names()
```

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1a',
 'arid1b',
 'arid2',
 'asxl2',
 'atm',
 'atrx',
 'aurka',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bard1',
 'bcl10',
 'bcl2',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
```



```
 'cdkn2a',
 'cdkn2b',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3a',
```

```
        'dnmt3b',
        'dusp4',
        'egfr',
        'eif1ax',
        'elf3',
        'ep300',
        'epas1',
        'epcam',
        'erbb2',
        'erbb3',
        'erbb4',
        'ercc2',
        'ercc3',
        'ercc4',
        'erg',
        'errfi1',
        'esr1',
        'etv1',
        'etv6',
        'ewsr1',
        'ezh2',
        'fam58a',
        'fanca',
        'fat1',
        'fbxw7',
        'fgf19',
        'fgf3',
        'fgfr1',
        'fgfr2',
        'fgfr3',
        'fgfr4',
        'flt1',
        'flt3',
        'foxa1',
        'foxl2',
        'foxo1',
        'foxp1',
        'gata3',
        'gnaq',
        'gnas',
        'h3f3a',
        'hist1h1c',
        'hla',
```



```
        'idh2',
        'igf1r',
        'ikbke',
        'ikzf1',
        'il7r',
        'jak1',
        'jak2',
        'kdm5a',
        'kdm5c',
        'kdm6a',
        'kdr',
        'keap1',
```

```
        'kit',
        'kmt2a',
        'kmt2c',
        'kmt2d',
        'knstrn',
        'kras',
        'lats2',
        'map2k1',
        'map2k2',
        'map2k4',
        'map3k1',
        'mapk1',
        'mdm2',
        'mdm4',
        'med12',
        'mef2b',
        'met',
        'mga',
        'mlh1',
        'mpl',
        'msh2',
        'msh6',
        'mtor',
        'myc',
        'mycn',
        'myd88',
        'ncor1',
        'nf1',
        'nf2',
        'nfe2l2',
        'nfkbia',
        'nkx2',
        'notch1',
        'notch2',
        'nras',
        'nsd1',
        'ntrk1',
        'ntrk2',
        'ntrk3',
        'nup93',
        'pak1',
        'pbrm1',
```



```
        'pik3cb',
        'pik3r1',
        'pik3r2',
        'pik3r3',
        'pim1',
        'pms1',
        'pms2',
        'pole',
        'ppm1d',
        'ppp2r1a',
        'ppp6c',
        'ptch1',
```

```
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rit1',
 'ros1',
 'rras2',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'srsf2',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
```



```
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tsc1',
 'tsc2',
 'u2af1',
 'vegfa',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
```

```
        'xrcc2',
        'yap1']
```

```
print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape
```

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The
```

Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good methods is to build a prop we will build a logistic regression model using only Gene feature (one hot encoded) to predict y_i.

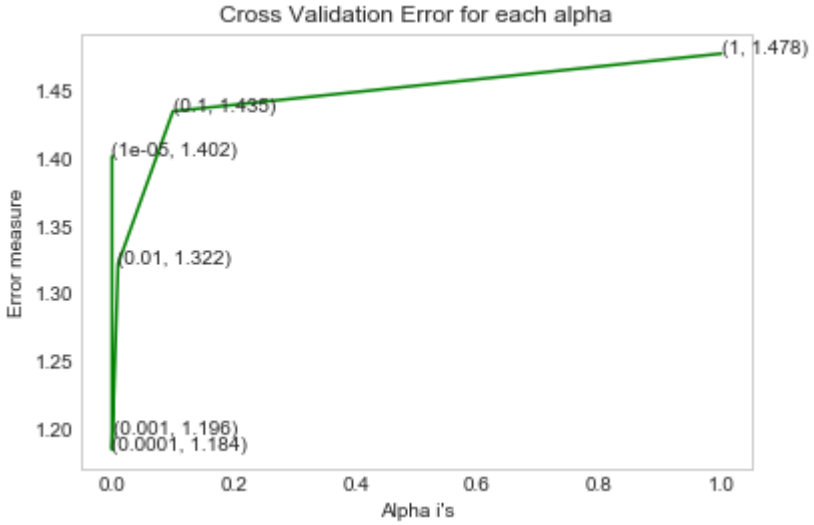
```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

#-------------------------------
# video link:
#-------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_gene_feature_onehotCoding, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_gene_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.class

fig, ax = plt.subplots()
```



```
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_gene_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
```

```
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
For values of alpha =  1e-05 The log loss is: 1.401587888330622
For values of alpha =  0.0001 The log loss is: 1.184278219828928
For values of alpha =  0.001 The log loss is: 1.1957739019271167
For values of alpha =  0.01 The log loss is: 1.3218183238224142
For values of alpha =  0.1 The log loss is: 1.4348431832433786
For values of alpha =  1 The log loss is: 1.4775244210764817
```




```
For values of best alpha =  0.0001 The train log loss is: 1.0526755177006817
For values of best alpha =  0.0001 The cross validation log loss is: 1.184278219828928
For values of best alpha =  0.0001 The test log loss is: 1.2306094742973166
```

Q5. Is the Gene feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Yes, it is. Otherwise, the CV and Test errors would be significantly more than train error.

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0],

test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.sh
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.sh
```



```
Ans
1. In test data 642 out of 665 : 96.54135338345866
2. In cross validation data 514 out of  532 : 96.61654135338345
```

3.2.2 Univariate Analysis on Variation Feature

Q7. Variation, What type of feature is it ?

Ans. Variation is a categorical variable

Q8. How many categories are there?

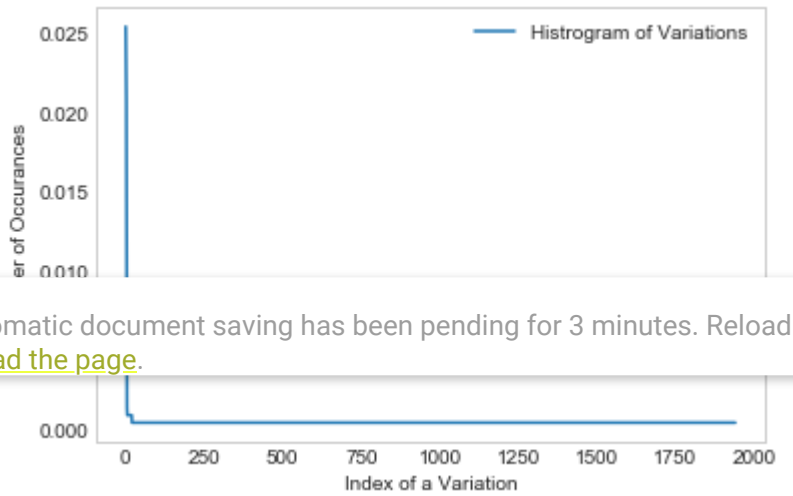
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1942
Truncating_Mutations    54
Amplification           48
Deletion                44
Fusions                 23
G12V                     3
P130S                    2
G13D                     2
Q61R                     2
G12C                     2
A146T                    2
Name: Variation, dtype: int64
```

```
print("Ans: There are", unique_variations.shape[0] ,"different categories of variations in the train
```

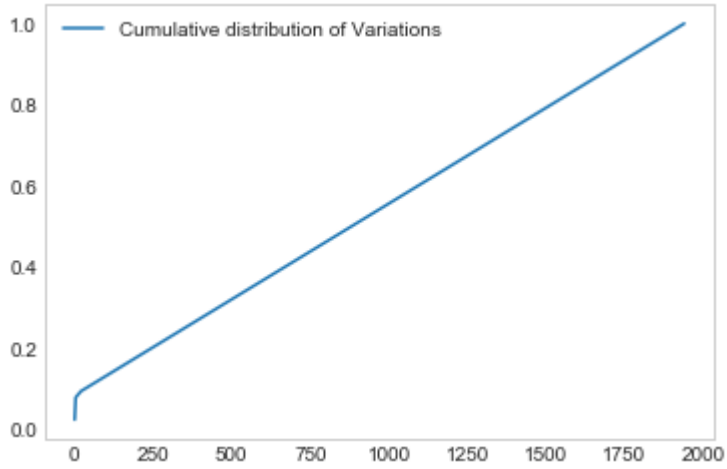
```
Ans: There are 1942 different categories of variations in the train data, and they are d
```

```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histrogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```




```
c = np.cumsum(h)
print(c)
plt.plot(c,label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()
```

```
[0.02542373 0.0480226  0.06873823 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans.There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/cours categorical-and-numerical-features/

1. One hot Encoding
2. Response coding

We will be using both these methods to featurize the Variation Feature

```
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

```
print("train_variation_feature_responseCoding is a converted feature using the response coding metho
```

```
train_variation_feature_responseCoding is a converted feature using the response coding
```

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
```



```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method
```

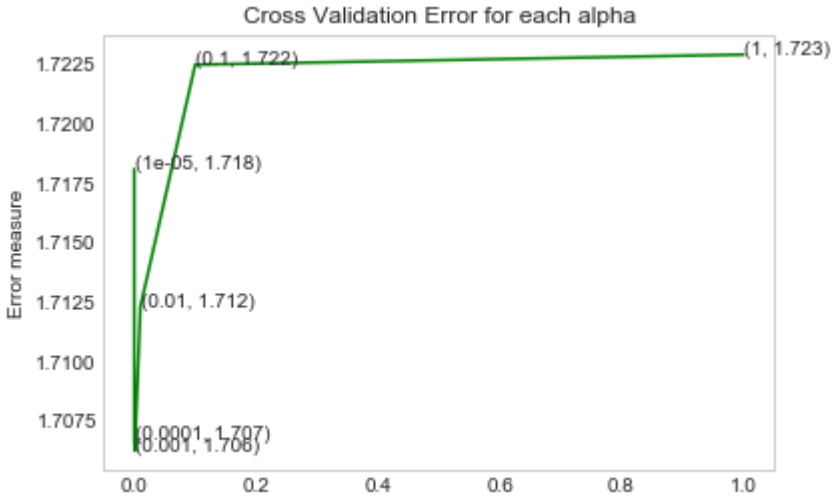
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding m
```

Q10. How good is this Variation feature in predicting y_i?

Let's build a model just like the earlier!

```
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

#-------------------------------
# video link:
#-------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_variation_feature_onehotCoding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)

    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.class

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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_variation_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_variation_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
```

```
For values of alpha =  1e-05 The log loss is: 1.7180739726855425
For values of alpha =  0.0001 The log loss is: 1.7067425142073074
For values of alpha =  0.001 The log loss is: 1.706252903318476
For values of alpha =  0.01 The log loss is: 1.7122807075616635
For values of alpha =  0.1 The log loss is: 1.7224470208555887
For values of alpha =  1 The log loss is: 1.7228706891106815
```



Q11. Is the Variation feature stable across all the data sets (Test, Train, Cross validation)?

Ans. Not sure! But lets be very sure using the below analysis.

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.sh
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverage/cv_df.sh
```

```
Q12. How many data points are covered by total  1942  genes in test and cross validation
Ans
1. In test data 85 out of 665 : 12.781954887218044
2. In cross validation data 54 out of  532 : 10.150375939849624
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?

4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

```
# cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
```

```
            for word in row['TEXT'].split():
                dictionary[word] +=1
        return dictionary


    import math
    #https://stackoverflow.com/a/1602964
    def get_text_responsecoding(df):
        text_feature_responseCoding = np.zeros((df.shape[0],9))
        for i in range(0,9):
            row_index = 0
            for index, row in df.iterrows():
                sum_prob = 0
                for word in row['TEXT'].split():
                    sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
                text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
                row_index += 1
        return text_feature_responseCoding


    # building a unigram CountVectorizer with all the words that occured minimum 3 times in train data
    text_unigram_vectorizer = CountVectorizer(min_df=3)
    train_text_unigram_feature_onehotCoding = text_unigram_vectorizer.fit_transform(train_df['TEXT'])
    # getting all the feature names (words)
    train_text_unigram_features= text_unigram_vectorizer.get_feature_names()

    # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of feature
    train_text_unigram_fea_counts = train_text_unigram_feature_onehotCoding.sum(axis=0).A1

    # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
    text_unigram_fea_dict = dict(zip(list(train_text_unigram_features),train_text_unigram_fea_counts))


    print("Total number of unique words in train data :", len(train_text_unigram_features))
```

```
Total number of unique words in train data : 52295
```

```
    # building a  bigram CountVectorizer with all the words that occured minimum 3 times in train data
    text_bigram_vectorizer = CountVectorizer(ngram_range=(1,2),min_df=3)
    train_text_bigram_feature_onehotCoding = text_bigram_vectorizer.fit_transform(train_df['TEXT'])
    # getting all the feature names (words)
    train_text_bigram_features= text_bigram_vectorizer.get_feature_names()

    # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of feature
    train_text_bigram_fea_counts = train_text_bigram_feature_onehotCoding.sum(axis=0).A1

    # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
    text_bigram_fea_dict = dict(zip(list(train_text_bigram_features),train_text_bigram_fea_counts))
```



```
Total number of unique words in train data : 761463
```

```
    # building a TfidfVectorizer with top 1000 words in train data
    text_vectorizer = TfidfVectorizer(max_features=1000, min_df=3, ngram_range=(1,4) )
    train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
    # getting all the feature names (words)
    train_text_features= text_vectorizer.get_feature_names()

    # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of feature
    train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

    # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
```

```
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 1000
```

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = train_df[train_df['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(train_df)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)


#response coding of text features
train_text_feature_responseCoding  = get_text_responsecoding(train_df)
test_text_feature_responseCoding  = get_text_responsecoding(test_df)
cv_text_feature_responseCoding  = get_text_responsecoding(cv_df)


# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_response
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCod
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.su


# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
```



```
# we use the same vectorizer that was trained on train data
cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)


#https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))



# Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

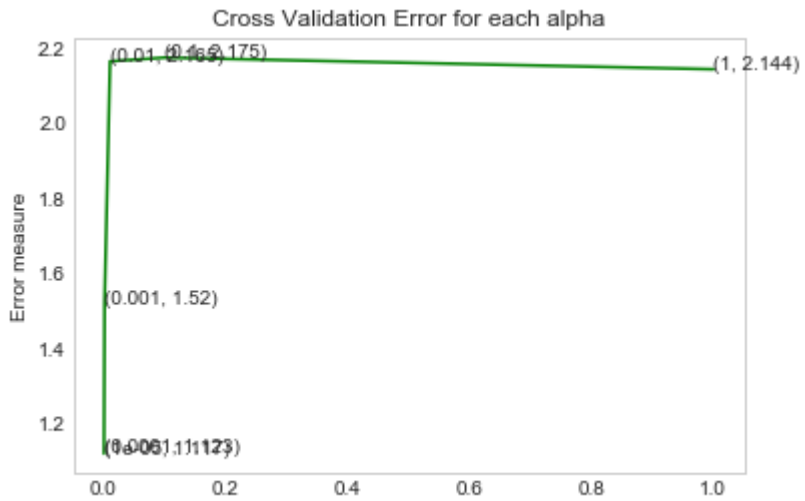
```
Counter({11.723334768592883: 2, 11.24635566555236: 2, 9.876068576812518: 2, 247.70852716
```

```
# Train a Logistic regression+Calibration model using text features whicha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

#--------------------------------
# video link:
#--------------------------------


cv_log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_text_feature_onehotCoding, y_train)

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_text_feature_onehotCoding, y_train)
    predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
    cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, labels=clf.class

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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_text_feature_onehotCoding, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_text_feature_onehotCoding, y_train)
```



```
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
For values of alpha =  1e-05 The log loss is: 1.1174521168006166
For values of alpha =  0.0001 The log loss is: 1.1234119274296015
For values of alpha =  0.001 The log loss is: 1.5201925965786458
For values of alpha =  0.01 The log loss is: 2.1653077906767053
For values of alpha =  0.1 The log loss is: 2.1752103152319333
For values of alpha =  1 The log loss is: 2.1440847454792604
```




```
def get_intersec_text(df):
    df_text_vec = CountVectorizer(min_df=3)
    df_text_fea = df_text_vec.fit_transform(df['TEXT'])
    df_text_features = df_text_vec.get_feature_names()

    df_text_fea_counts = df_text_fea.sum(axis=0).A1
    df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
    len1 = len(set(df_text_features))
    len2 = len(set(train_text_features) & set(df_text_features))
    return len1,len2
```

```
len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appeared in train data")
```

```
3.194 % of word of test data appeared in train data
3.437 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

Automatic document saving has been pending for 3 minutes. Reloading may fix the problem. Save and reload the page.

```
#Data preparation for ML models.

#Misc. functionns for ML models


def predict_and_plot_confusion_matrix(train_x, train_y,test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    pred_y = sig_clf.predict(test_x)

    # for calculating log_loss we willl provide the array of probabilities belongs to each class
    print("Log loss :",log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
```

```
    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)


def report_log_loss(train_x, train_y, test_x, test_y,  clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
    sig_clf_probs = sig_clf.predict_proba(test_x)
    return log_loss(test_y, sig_clf_probs, eps=1e-15)


# this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer()
    var_count_vec = CountVectorizer()
    text_count_vec = CountVectorizer(min_df=3)

    gene_vec = gene_count_vec.fit(train_df['Gene'])
    var_vec  = var_count_vec.fit(train_df['Variation'])
    text_vec = text_count_vec.fit(train_df['TEXT'])

    fea1_len = len(gene_vec.get_feature_names())
    fea2_len = len(var_count_vec.get_feature_names())

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}] present in test data point [{}]".format(word,yes_no))
        elif (v < fea1_len+fea2_len):
            word = var_vec.get_feature_names()[v-(fea1_len)]
            yes_no = True if word == var else False
            if yes_no:
                word_present += 1
                print(i, "variation feature [{}] present in test data point [{}]".format(word,yes_no
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
                print(i, "Text feature [{}] present in test data point [{}]".format(word,yes_no))

    print("Out of the top ",no_features," features ", word_present, "are present in query point")
```



```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehot
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCod
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))
```

```
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr(
train_y = np.array(list(train_df['Class']))

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(test_df['Class']))

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(cv_df['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_re
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_response

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_responseCoding
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
```

```
print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_onehotCoding.s
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3205)
(number of data points * number of features) in test data =  (665, 3205)
(number of data points * number of features) in cross validation data = (532, 3205)
```

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

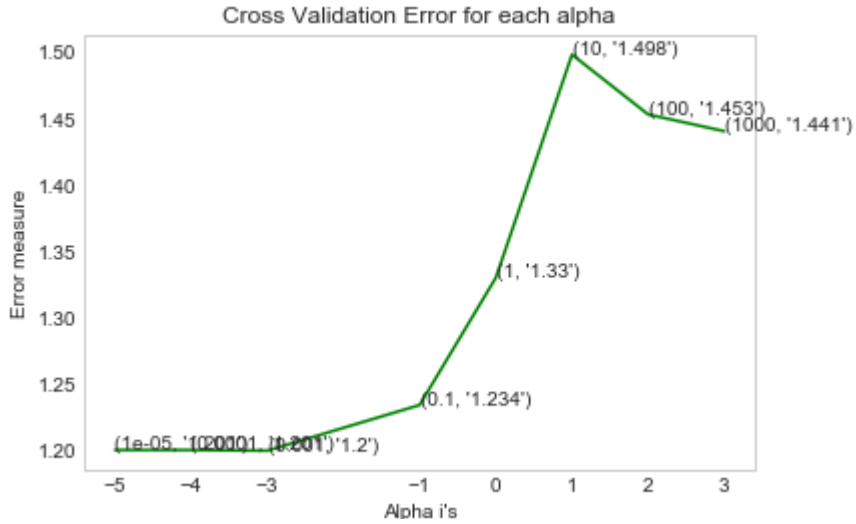
4.1. Base Line Model

4.1.1.1. Hyper parameter tuning

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/genera
# ------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])  Fit Naive Bayes classifier according to X, y
# predict(X)  Perform classification on an array of test vectors X.
# predict_log_proba(X)  Return log-probability estimates for the test vector X.
# -----------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-al
```

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-al
# ----------------------


alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = MultinomialNB(alpha=i)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i],str(txt)), (np.log10(alpha[i]),cv_log_error_array[i]))
plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)


predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
```

```
for alpha = 1e-05
Log Loss : 1.200619719237425
for alpha = 0.0001
Log Loss : 1.2005988333380475
for alpha = 0.001
Log Loss : 1.2001179867102985
for alpha = 0.1
Log Loss : 1.234466432610462
for alpha = 1
Log Loss : 1.3302877989366166
for alpha = 10
Log Loss : 1.4981834977664836
for alpha = 100
Log Loss : 1.4531333396014572
for alpha = 1000
Log Loss : 1.440534055324923
```



4.1.1.2. Testing the model with best hyper paramters

```
for values of best alpha =  0.001 the test log loss is: 1.2167518142139051
```

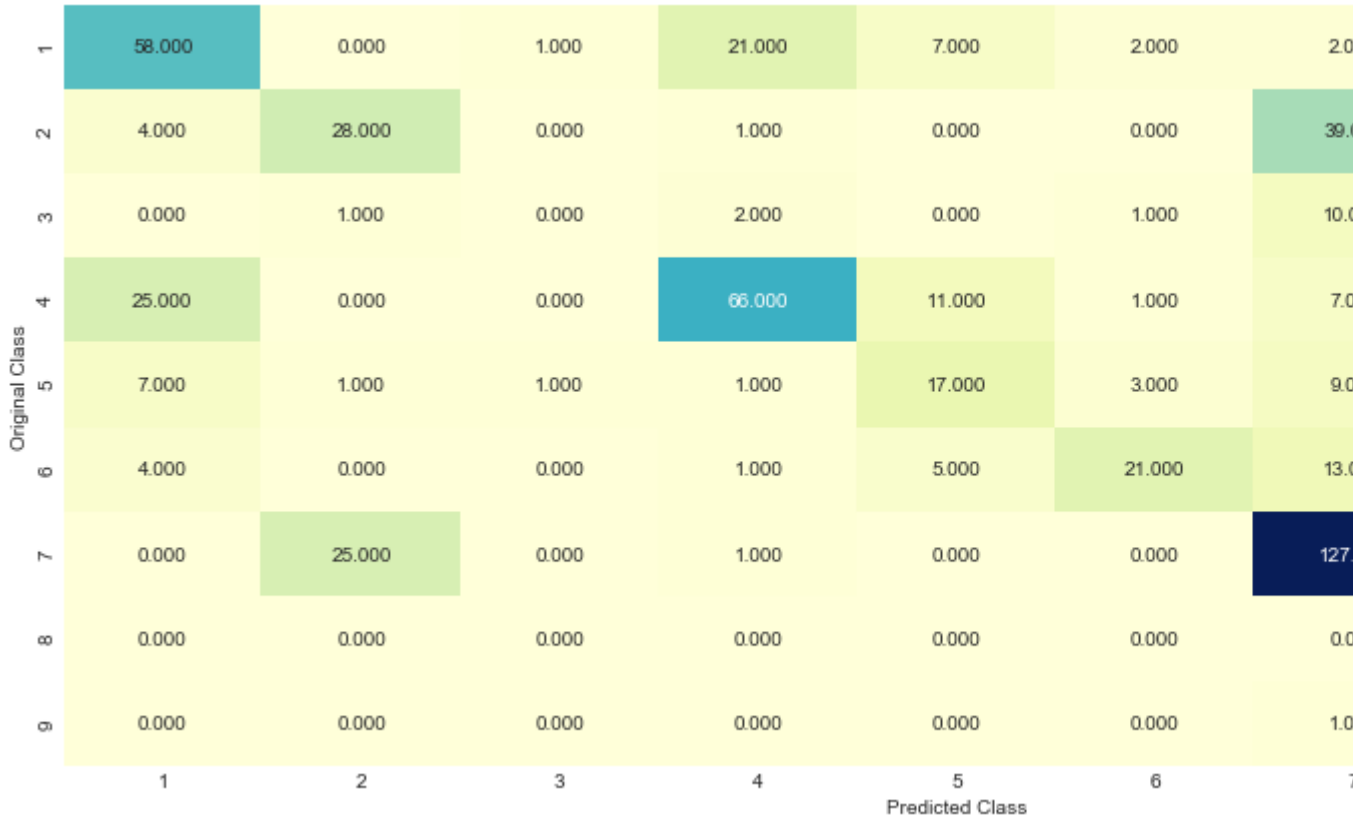
```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/genera
# -------------------------
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])  Fit Naive Bayes classifier according to X, y
# predict(X)  Perform classification on an array of test vectors X.
```



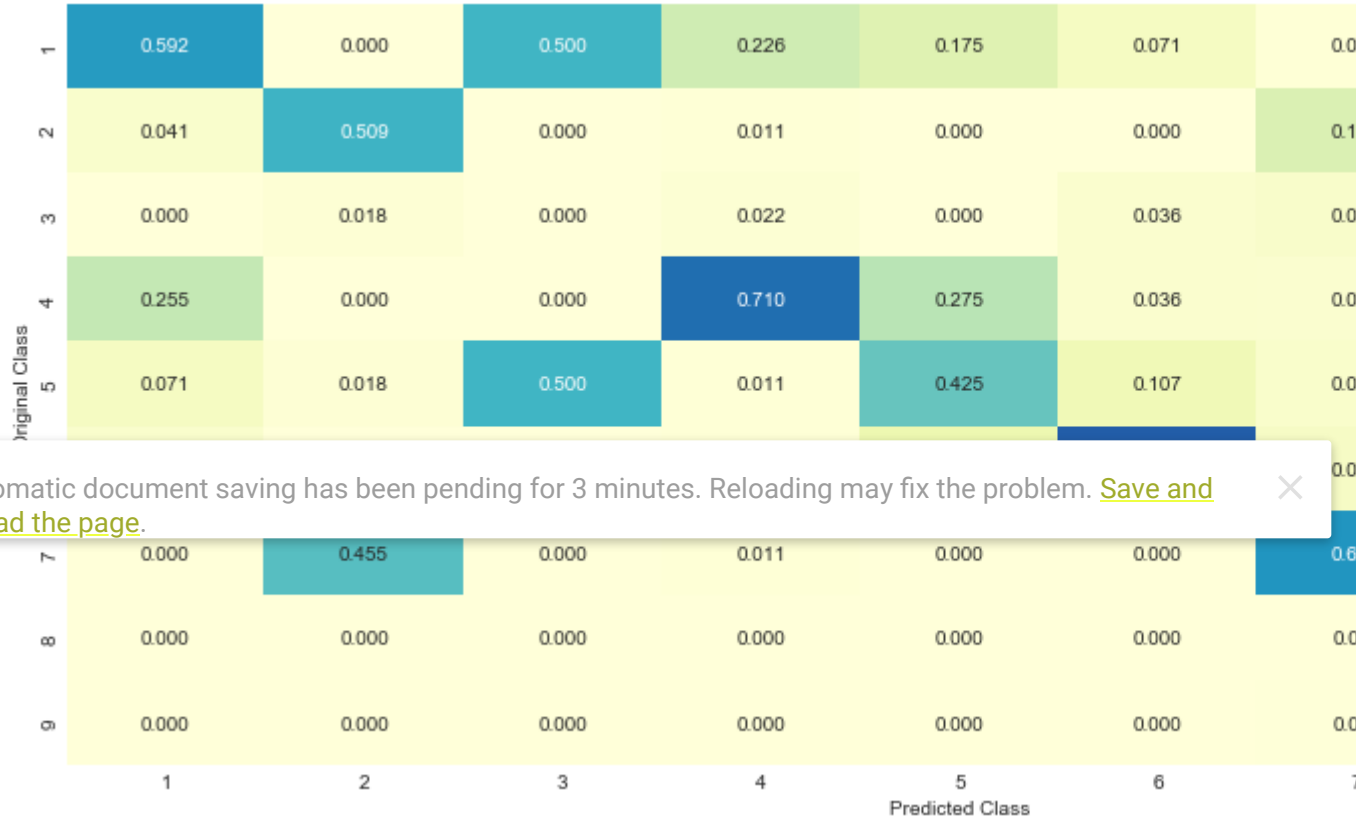
```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, 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
# ----------------------------
```

```
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilites we use log-probability estimates
print("Log Loss :",log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding)- cv_y
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

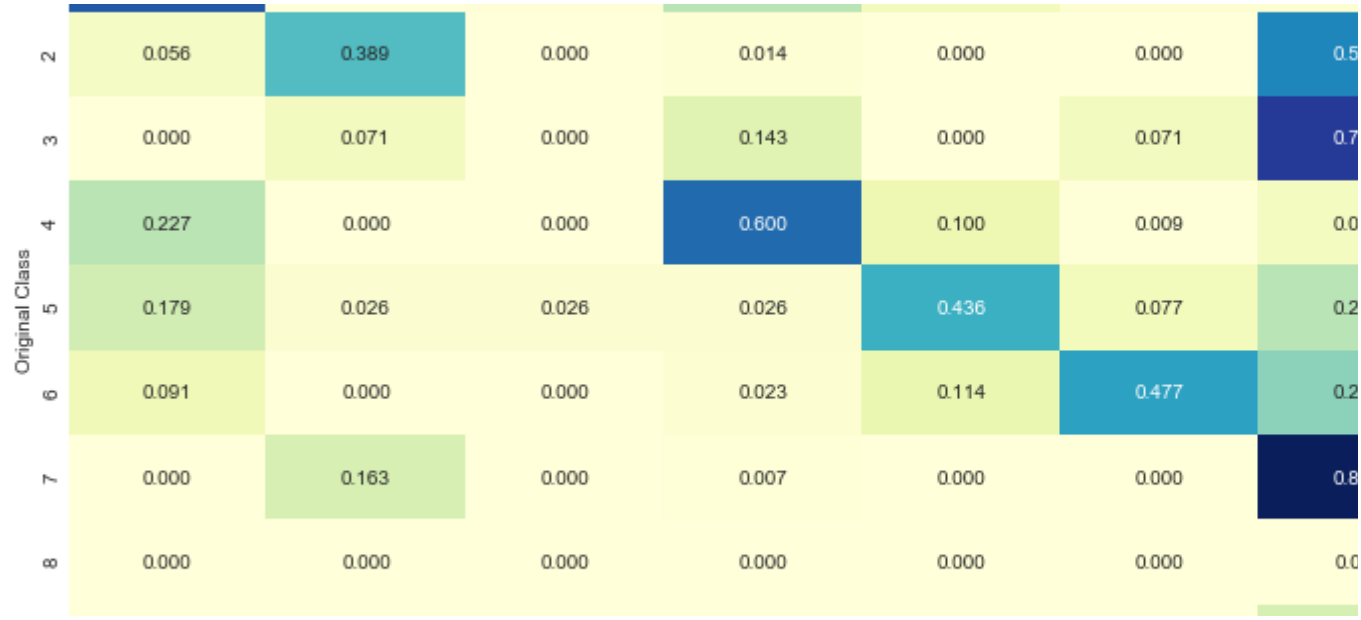
```
Log Loss : 1.2001179867102985
Number of missclassified point : 0.39285714285714285
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
results=pd.DataFrame(columns=['Model','Featuraization', 'Train loss','CV loss','Test loss','Misscla
```

```
1 2 3 4 5 6
```

```
new = ['Naive Bayes','tfidf',0.5215,1.1689,1.200,0.3928]
results.loc[0] = new
```

4.1.1.3. Feature Importance, Correctly classified point

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0602 0.0517 0.0105 0.06   0.0329 0.029  0.7502 0.0034
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```


4.1.1.4. Feature importance, Incorrectly classified point

```
test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.06   0.0449 0.0105 0.0599 0.0328 0.0289 0.7577 0.0035
Actual Class : 7
--------------------------------------------------
48 Text feature [1010] present in test data point [True]
Out of the top  100  features  1 are present in query point
```

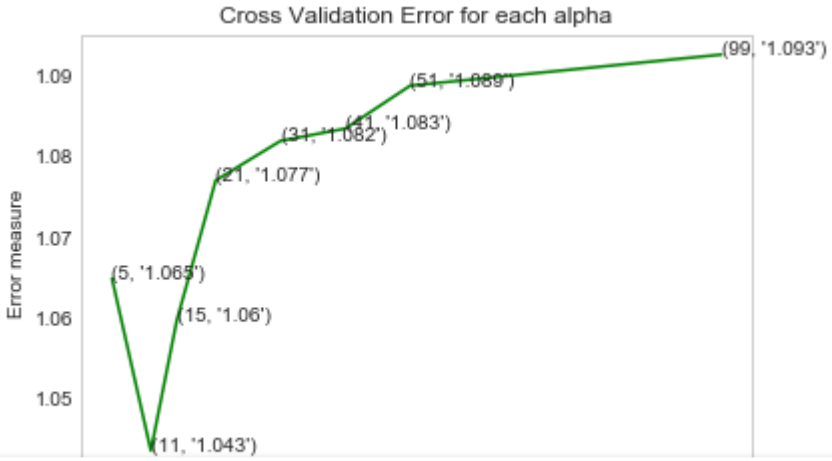
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# 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/k-nearest-neig
#-------------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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 = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (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()
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 5
Log Loss : 1.0648012948062784
for alpha = 11
Log Loss : 1.0433556213407689
for alpha = 15
Log Loss : 1.059590982726817
for alpha = 21
Log Loss : 1.077022885028216
for alpha = 31
Log Loss : 1.0819429304582002
for alpha = 41
Log Loss : 1.083476016398296
for alpha = 51
Log Loss : 1.0888233561158065
for alpha = 99
Log Loss : 1.0926915075149923
```



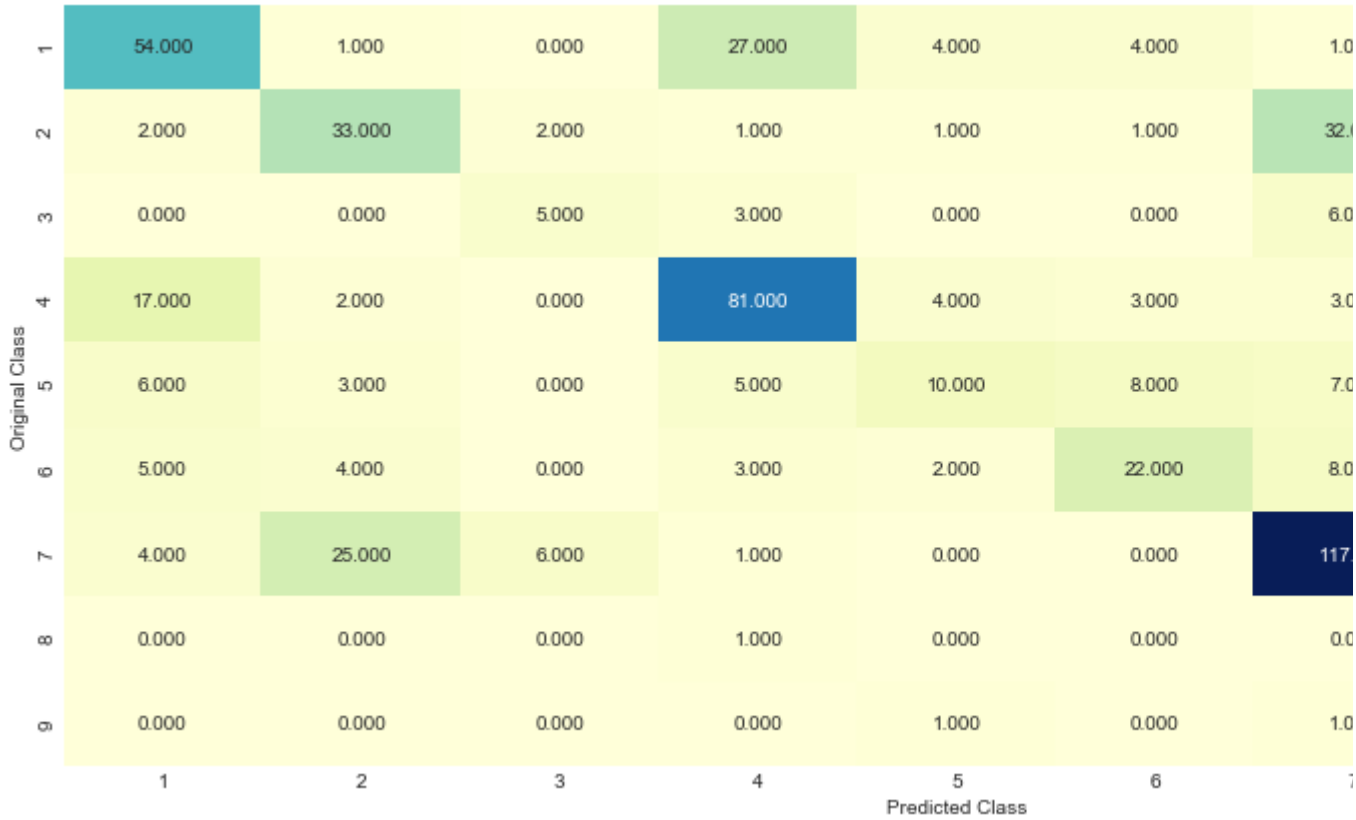

```
For values of best alpha =  11 The cross validation log loss is: 1.0433556213407689
For values of best alpha =  11 The test log loss is: 1.0925597047702071
```

4.2.2. Testing the model with best hyper paramters

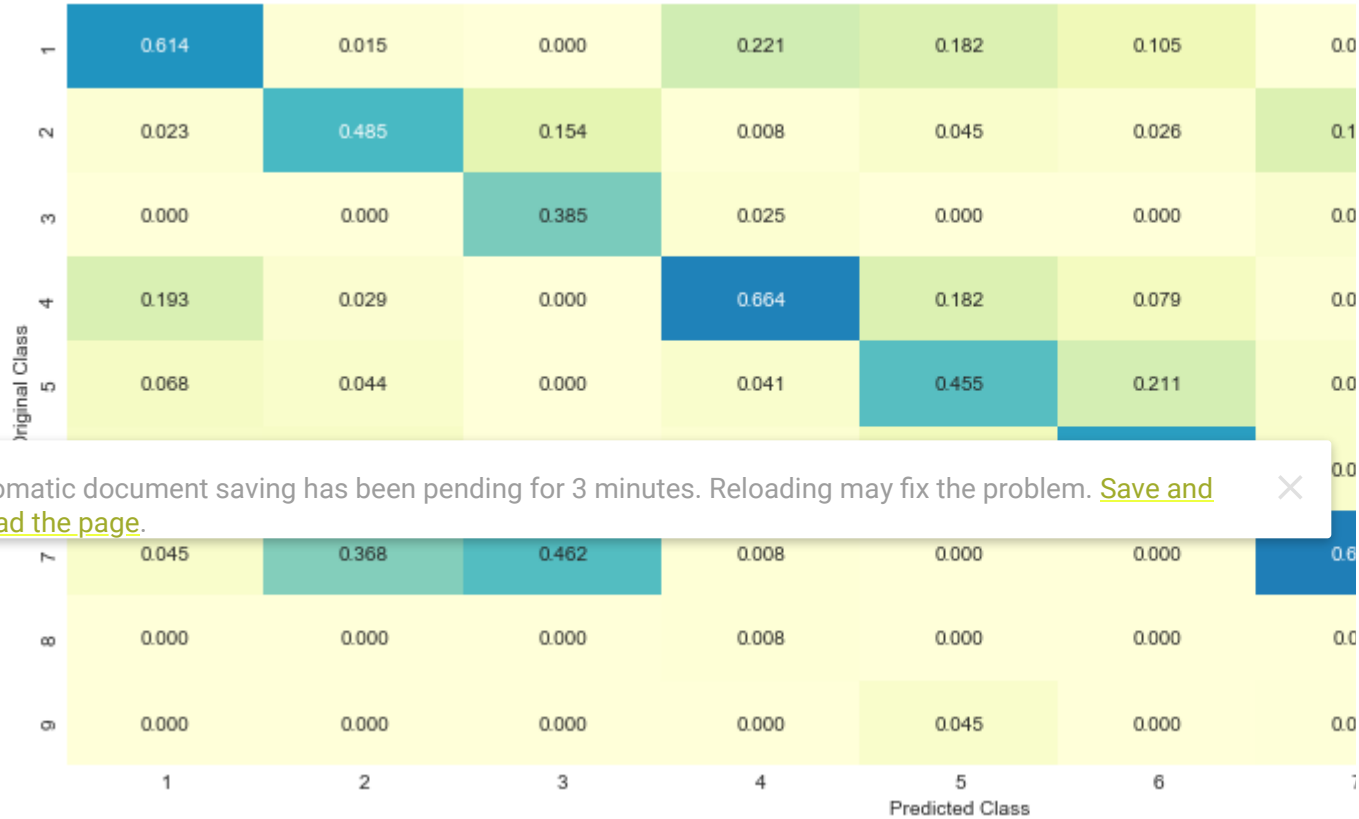
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklea
# ------------------------
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
```

```
    # 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/k-nearest-neig
    #-------------------------------------
    clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
    predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)
```

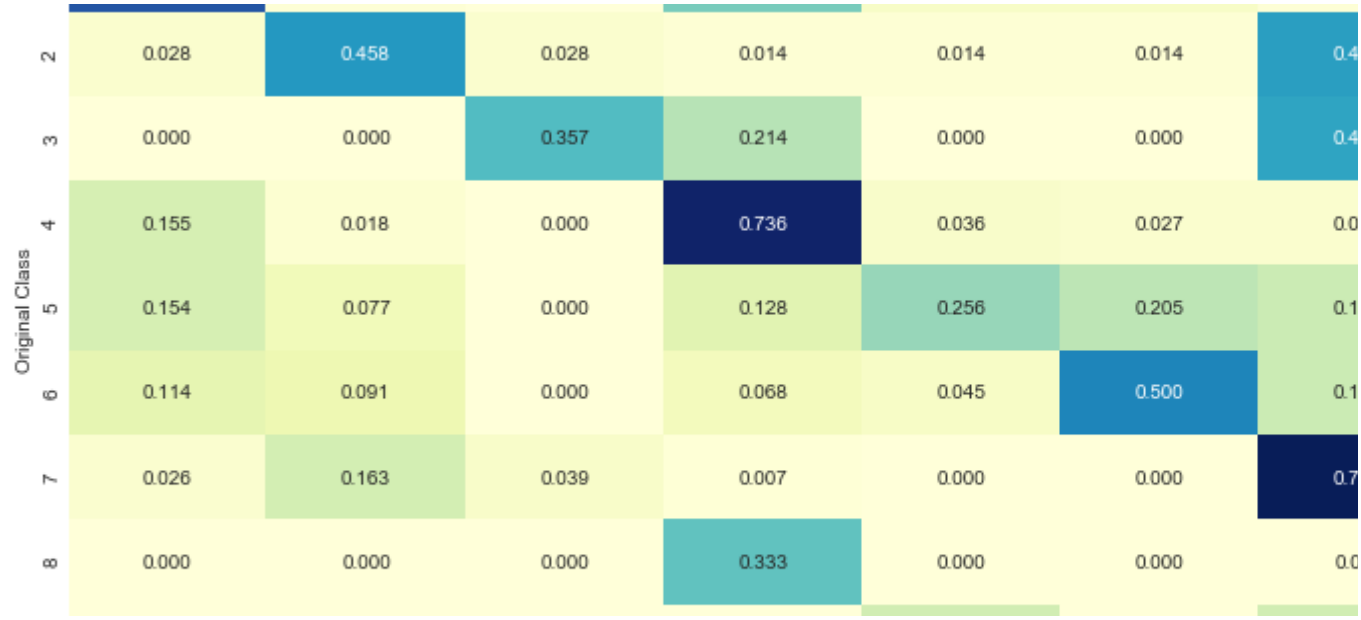
```
Log loss : 1.0433556213407689
Number of mis-classified points : 0.38533834586466165
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['KNN','tfidf',0.6612,1.0433,1.0925,0.3853]
```

```
                1          2          3          4          5          6
```

```
results.loc[1] = new
```

4.2.3.Sample Query point -1

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha]
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[n
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
The  11  nearest neighbours of the test points belongs to classes [7 7 7 2 7 7 7 7 7 7 7
Fequency of nearest points : Counter({7: 10, 2: 1})
```


4.2.4. Sample Query Point-2

```
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha]
```

```
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points belo
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 11 and the nearest neighbours of the test points belongs to class
Fequency of nearest points : Counter({7: 11})
```

4.3. Logistic Regression (TfidfVectorizer)

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

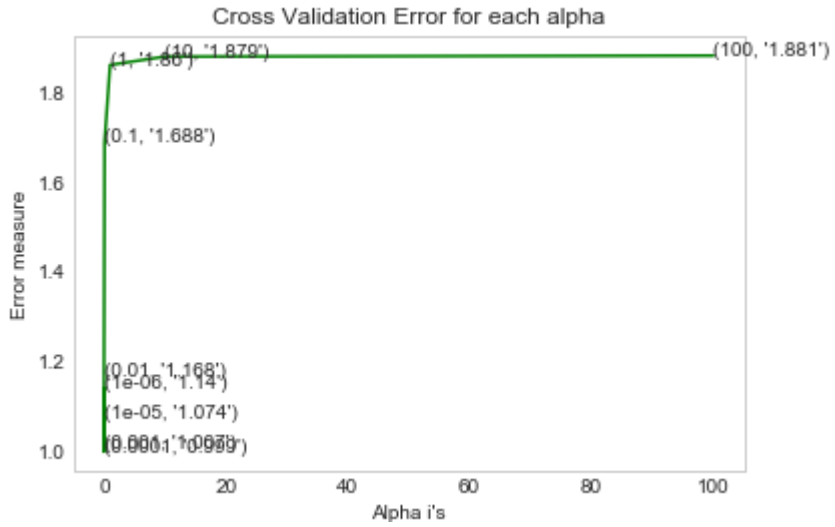
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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 = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))
```

```
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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.1402165160281854
for alpha = 1e-05
Log Loss : 1.0742855807864402
for alpha = 0.0001
Log Loss : 0.9991032994286291
for alpha = 0.001
Log Loss : 1.0065157855261144
for alpha = 0.01
Log Loss : 1.1678911993561587
for alpha = 0.1
Log Loss : 1.6878728695399141
for alpha = 1
Log Loss : 1.8604119352892565
for alpha = 10
Log Loss : 1.8789987564279809
for alpha = 100
Log Loss : 1.8812481030306558
```



4.3.1.2. Testing the model with best hyper paramters

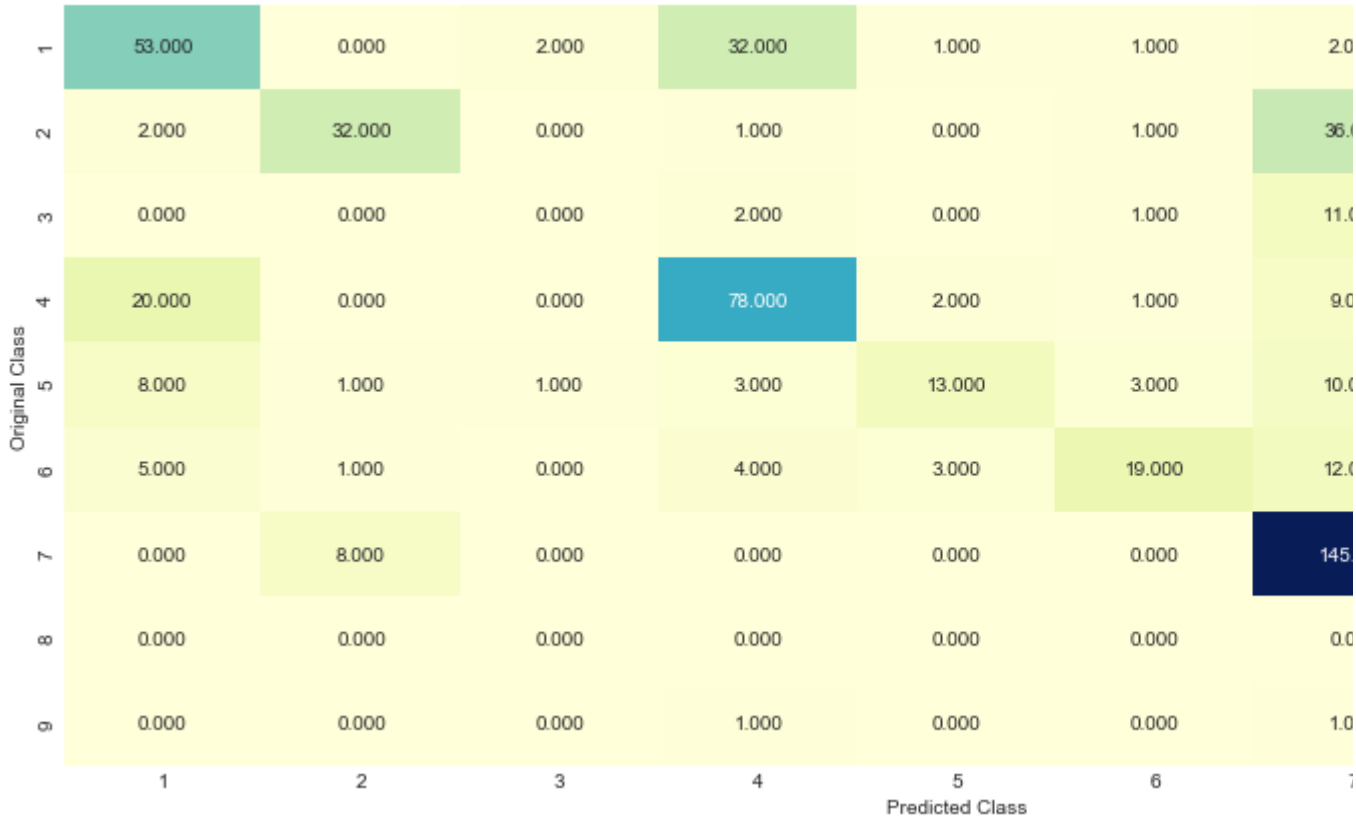
```
for values of best alpha =  0.0001 The test log loss is: 1.0363740556766239
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

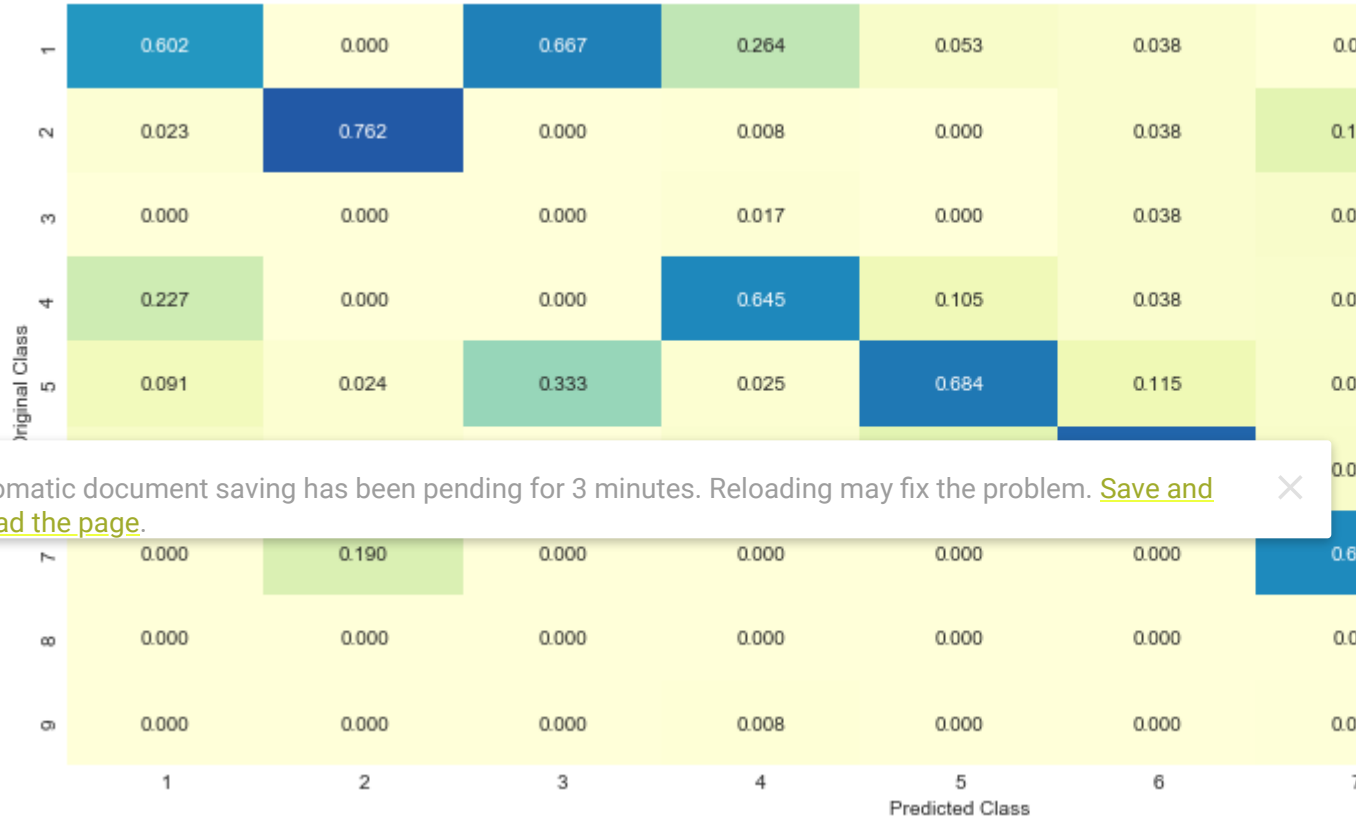


```
# predict(X)  Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intu
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

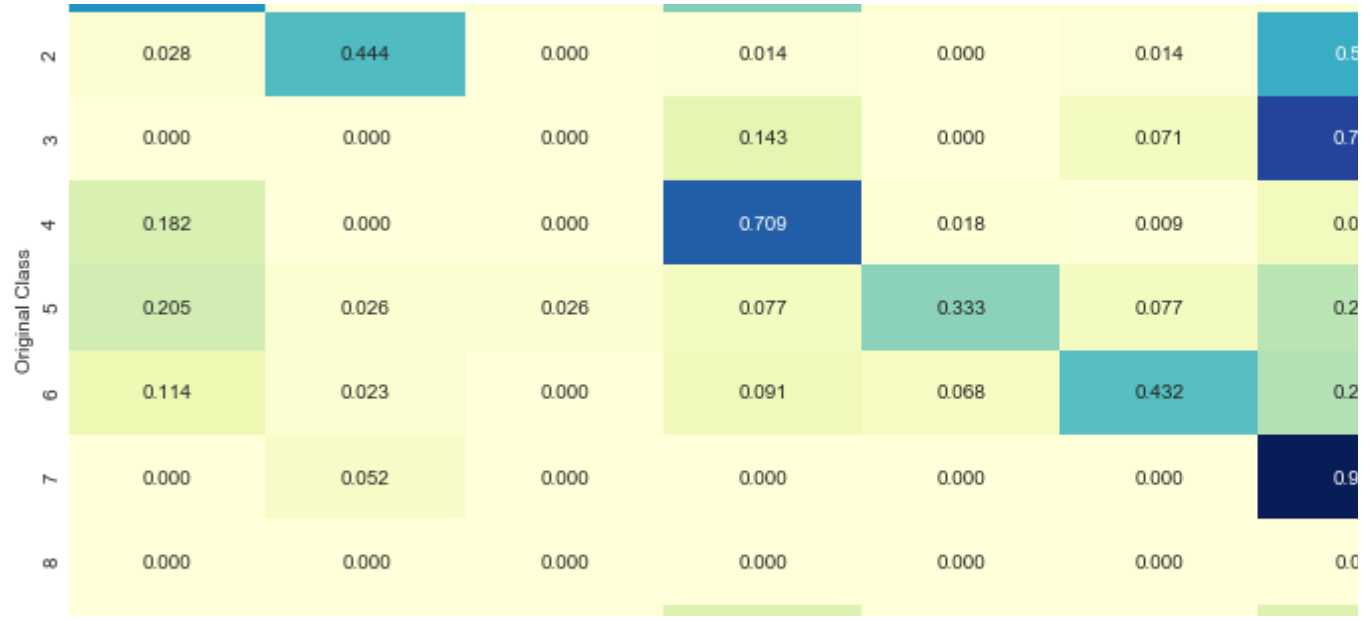
```
Log loss : 0.9991032994286291
Number of mis-classified points : 0.35150375939849626
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['Lr- Regression with class balance','tfidf',0.4517,0.9991,1.3637,0.3515]
results.loc[2] = new
```

4.3.1.3. Feature Importance

```
def get_imp_feature_names(text, indices, removed_ind = []):
    word_present = 0
    tabulte_list = []
    incresingorder_ind = 0
    for i in indices:
        if i < train_gene_feature_onehotCoding.shape[1]:
            tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
        elif i< 18:
            tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
        if ((i > 17) & (i not in removed_ind)) :
            word = train_text_features[i]
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']))
```


4.3.1.3.1. Correctly classified point

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.650e-02 3.240e-02 1.000e-03 1.910e-02 3.300e-03 3.500
  1.200e-03 5.000e-04]]
Actual Class : 7
--------------------------------------------------
98 Text feature [11f] present in test data point [True]
423 Text feature [113] present in test data point [True]
Out of the top  500  features  2 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0358 0.039  0.0025 0.1344 0.0137 0.0076 0.7631 0.0024
Actual Class : 7
--------------------------------------------------
17 Text feature [114] present in test data point [True]
284 Text feature [130] present in test data point [True]
309 Text feature [101] present in test data point [True]
423 Text feature [113] present in test data point [True]
437 Text feature [128] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

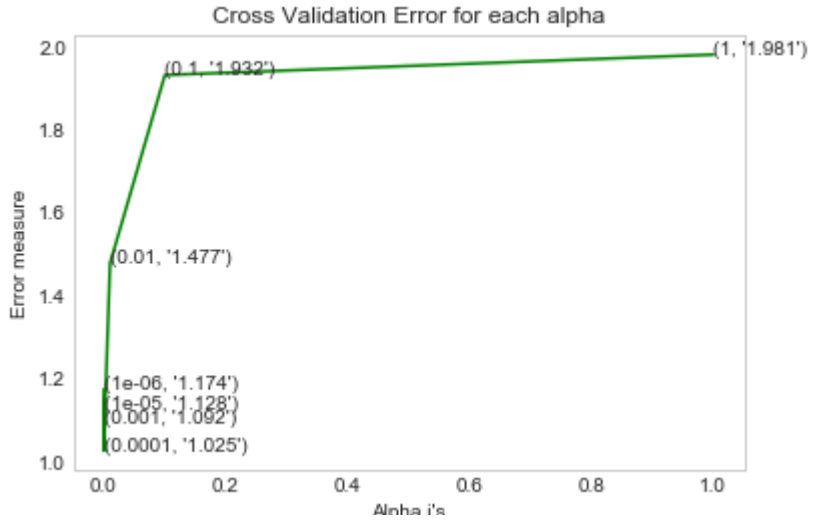
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning


```
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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

```
# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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 = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.1737823250705735
for alpha = 1e-05
Log Loss : 1.1277288561360281
for alpha = 0.0001
Log Loss : 1.0252303804269425
for alpha = 0.001
Log Loss : 1.0922130503823773
for alpha = 0.01
Log Loss : 1.4773837325481118
for alpha = 0.1
Log Loss : 1.9317900025141146
for alpha = 1
Log Loss : 1.9810927405461343
```



4.3.2.2. Testing model with best hyper parameters

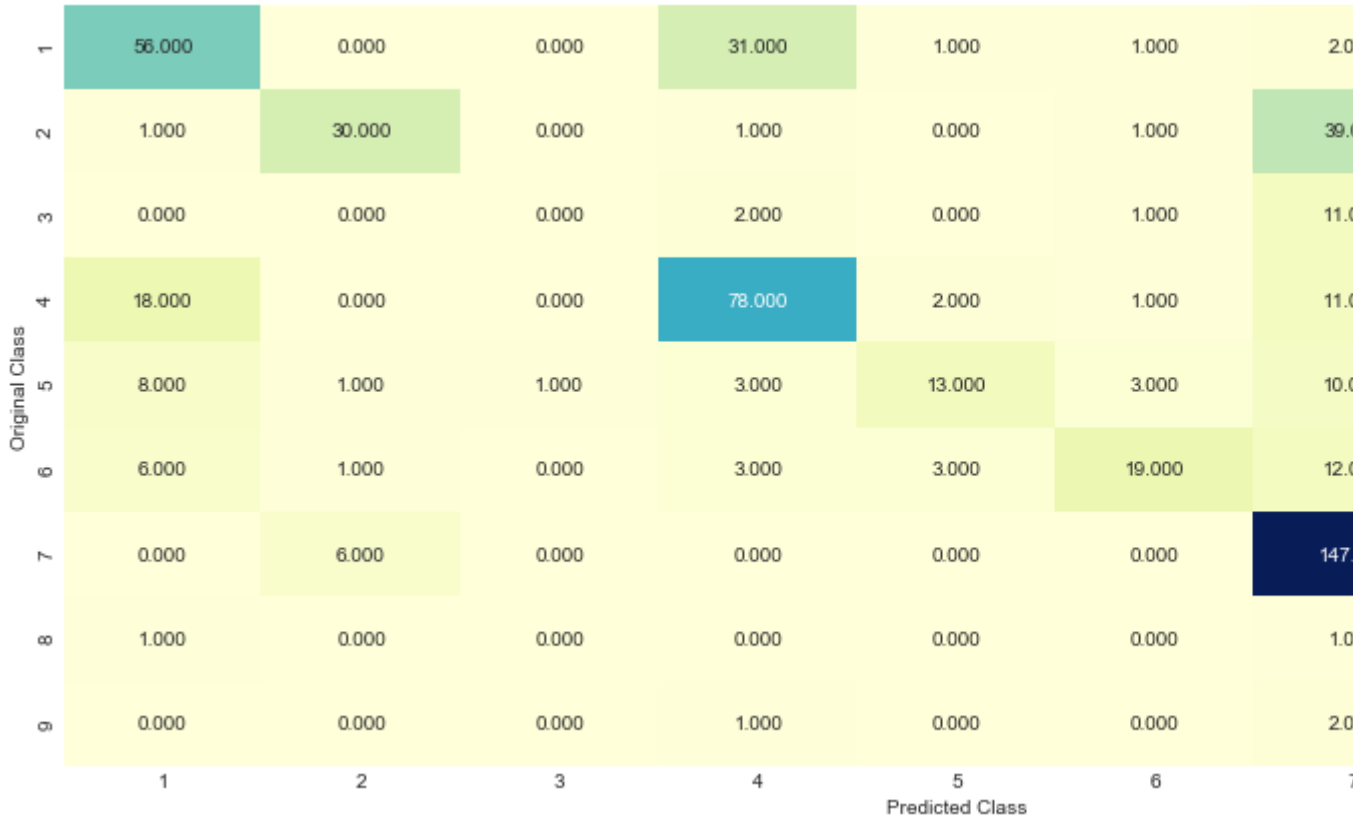
```
For values of best alpha =  0.0001 The test log loss is: 1.0614844377116432
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.
```

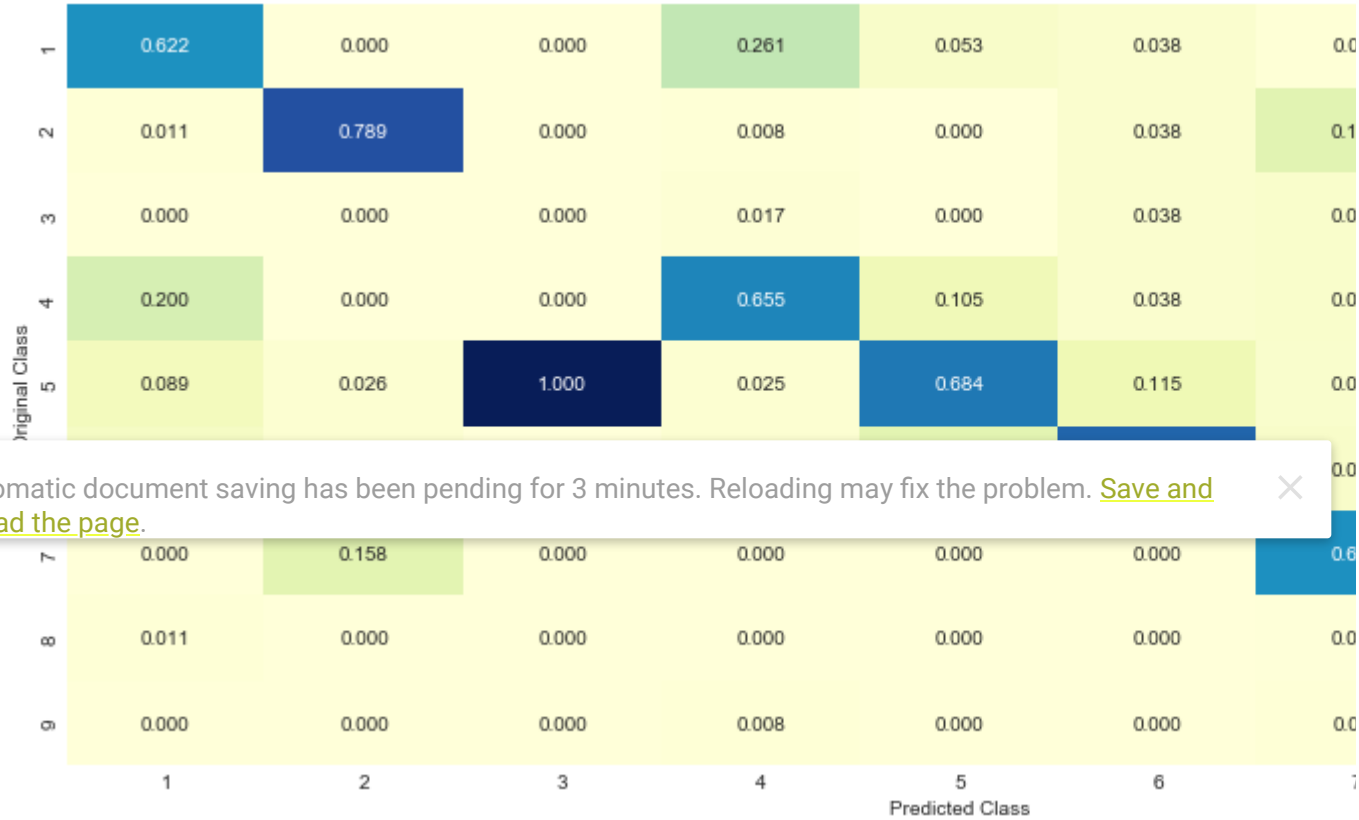


```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

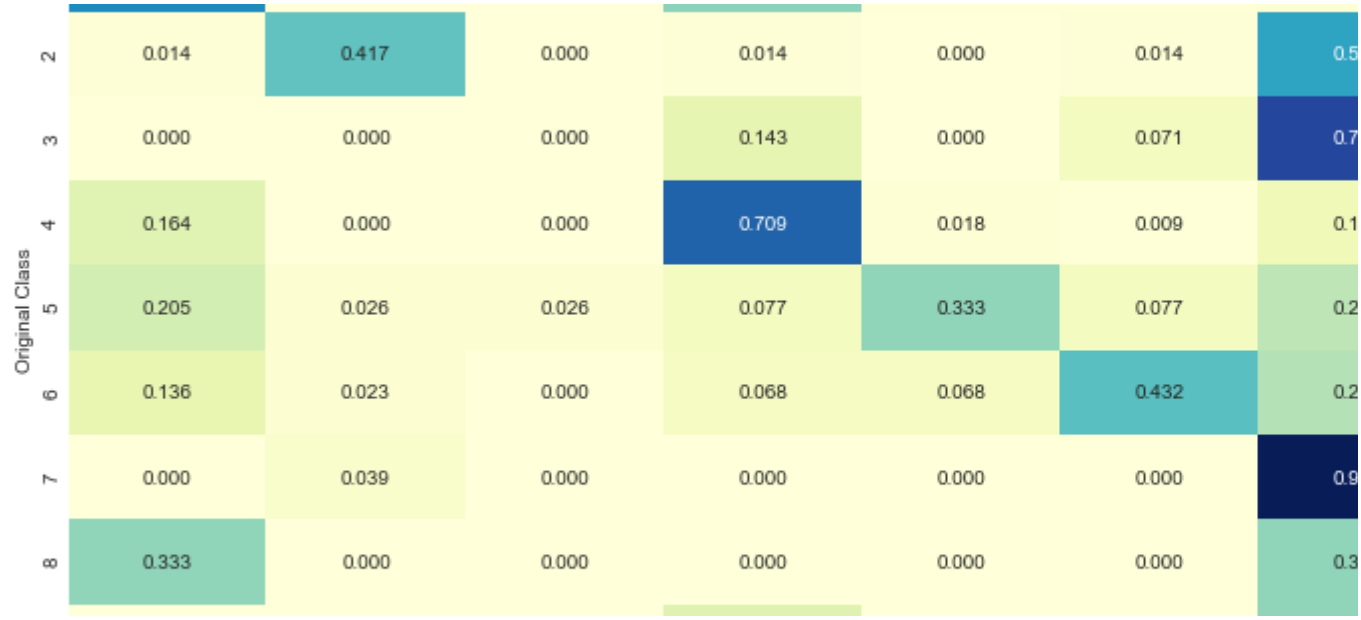
```
Log loss : 1.0252303804269425
Number of mis-classified points : 0.34962406015037595
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['Lr- Regression without class balance','tfidf',0.4438,1.0252,1.0614,0.3496]
results.loc[3] = new
```

4.3.2.3. Feature Importance, Correctly Classified point

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[3.810e-02 3.140e-02 7.000e-04 1.790e-02 2.900e-03 3.000
  5.000e-04 1.000e-04]]
Actual Class : 7
--------------------------------------------------
100 Text feature [11f] present in test data point [True]
```


4.3.2.4. Feature Importance, Inorrectly Classified point

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0327 0.0313 0.0025 0.1201 0.0128 0.0074 0.7891 0.003
Actual Class : 7
--------------------------------------------------
50 Text feature [114] present in test data point [True]
196 Text feature [101] present in test data point [True]
457 Text feature [130] present in test data point [True]
485 Text feature [127] present in test data point [True]
498 Text feature [128] present in test data point [True]
Out of the top  500  features  5 are present in query point
```

4.4. Logistic Regression (Unigrams & Bigrams)

4.4.1 Unigrams

4.4.1.1. With Class balancing

4.4.1.1.1. Hyper paramter tuning

```
# don't forget to normalize every feature
train_text_unigram_feature_onehotCoding = normalize(train_text_unigram_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_unigram_feature_onehotCoding = text_unigram_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_unigram_feature_onehotCoding = normalize(test_text_unigram_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_unigram_feature_onehotCoding = text_unigram_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_unigram_feature_onehotCoding = normalize(cv_text_unigram_feature_onehotCoding, axis=0)
```

```
#https://stackoverflow.com/a/2258273/4084039
sorted_text_unigram_fea_dict = dict(sorted(text_unigram_fea_dict.items(), key=lambda x: x[1] , rever
sorted_text_unigram_occur = np.array(list(sorted_text_unigram_fea_dict.values()))
```



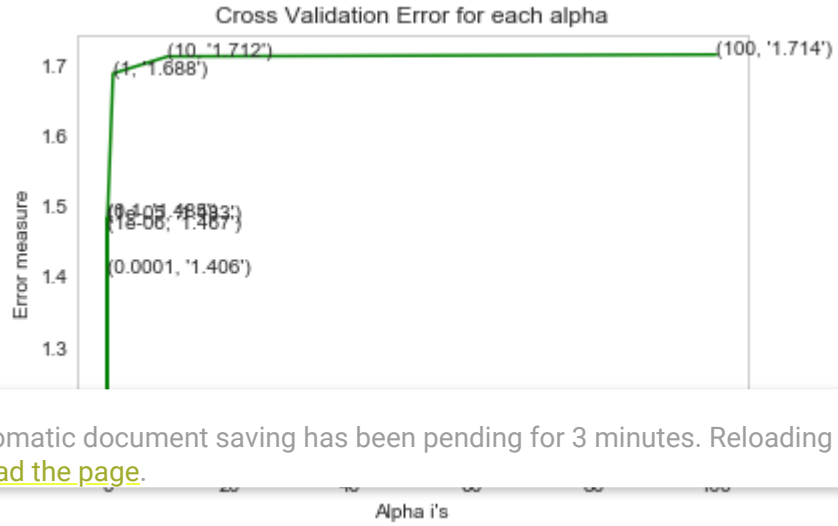
```
Counter({3: 4834, 4: 3856, 5: 2964, 6: 2572, 9: 2074, 8: 1828, 7: 1715, 12: 1421, 10: 13
```

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
```

```
#                [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehot
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCod
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

train_x_unigram_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_unigram_feature_oneho
train_y_unigram = np.array(list(train_df['Class']))

test_x_unigram_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_unigram_feature_onehotCo
test_y_unigram = np.array(list(test_df['Class']))

cv_x_unigram_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_unigram_feature_onehotCoding))
cv_y_unigram = np.array(list(cv_df['Class']))



# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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 = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
                                                                                              2)
    sig_clf_probs = sig_clf.predict_proba(cv_x_unigram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y_unigram, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y_unigram, sig_clf_probs))

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],str(txt)), (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()
```

```
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
clf.fit(train_x_unigram_onehotCoding, train_y_unigram)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_unigram_onehotCoding, train_y_unigram)

predict_y = sig_clf.predict_proba(train_x_unigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_unigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_unigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.4673206371110814
for alpha = 1e-05
Log Loss : 1.4825972267882255
for alpha = 0.0001
Log Loss : 1.405515739748252
for alpha = 0.001
Log Loss : 1.1531088456769596
for alpha = 0.01
Log Loss : 1.1924331407241215
for alpha = 0.1
Log Loss : 1.4847281667396244
for alpha = 1
Log Loss : 1.6876636251903747
for alpha = 10
Log Loss : 1.7117812058497202
for alpha = 100
Log Loss : 1.71435992944071
```



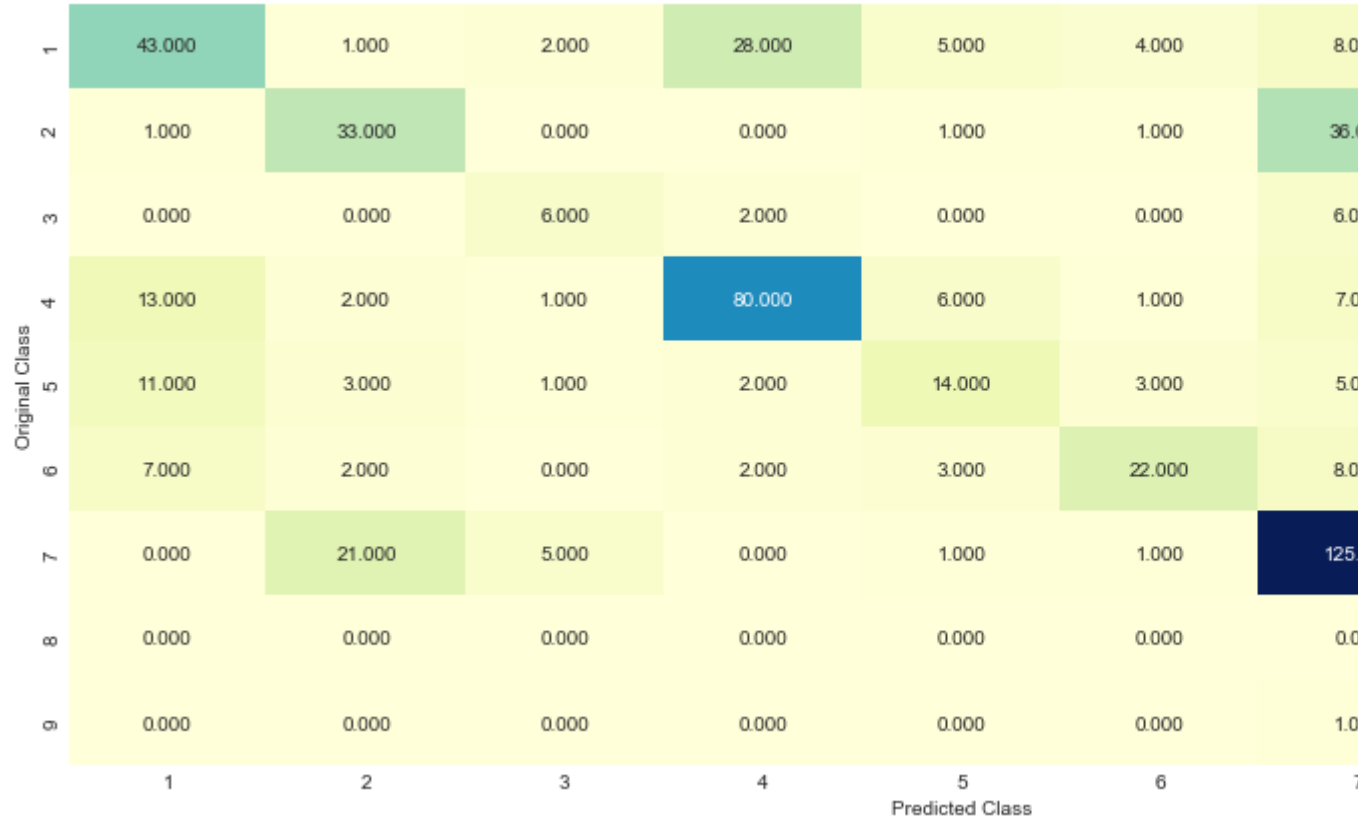

```
For values of best alpha =  0.001 The train log loss is: 0.6338315735754386
For values of best alpha =  0.001 The cross validation log loss is: 1.1531088456769596
For values of best alpha =  0.001 The test log loss is: 1.1218944287056698
```

4.4.1.1.2. Testing the model with best hyper paramters

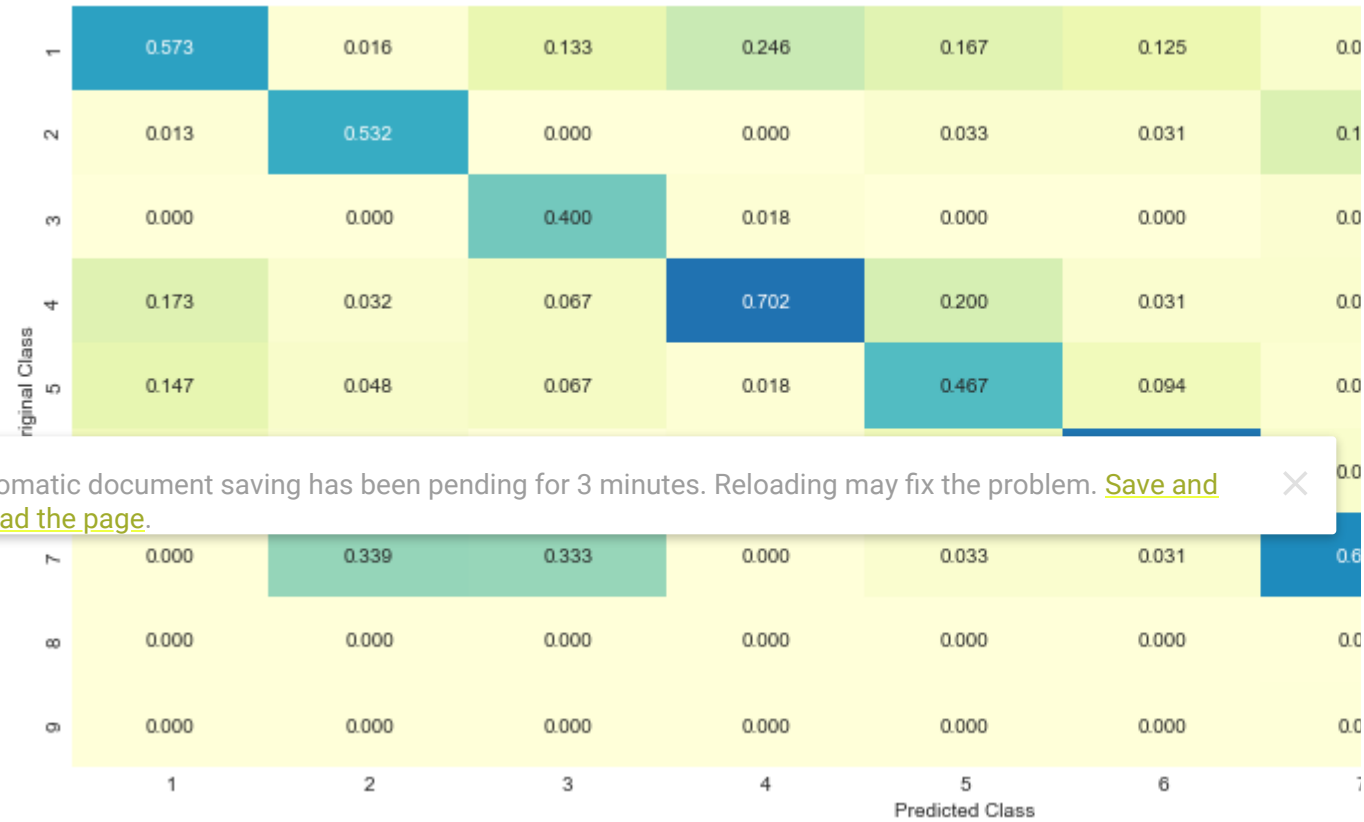
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
```

```
    # default parameters
    # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
    # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
    # 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 Descent.
    # predict(X)  Predict class labels for samples in X.

    #-------------------------------
    # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intu
    #-------------------------------
    clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
    predict_and_plot_confusion_matrix(train_x_unigram_onehotCoding, train_y_unigram, cv_x_unigram_onehot
```

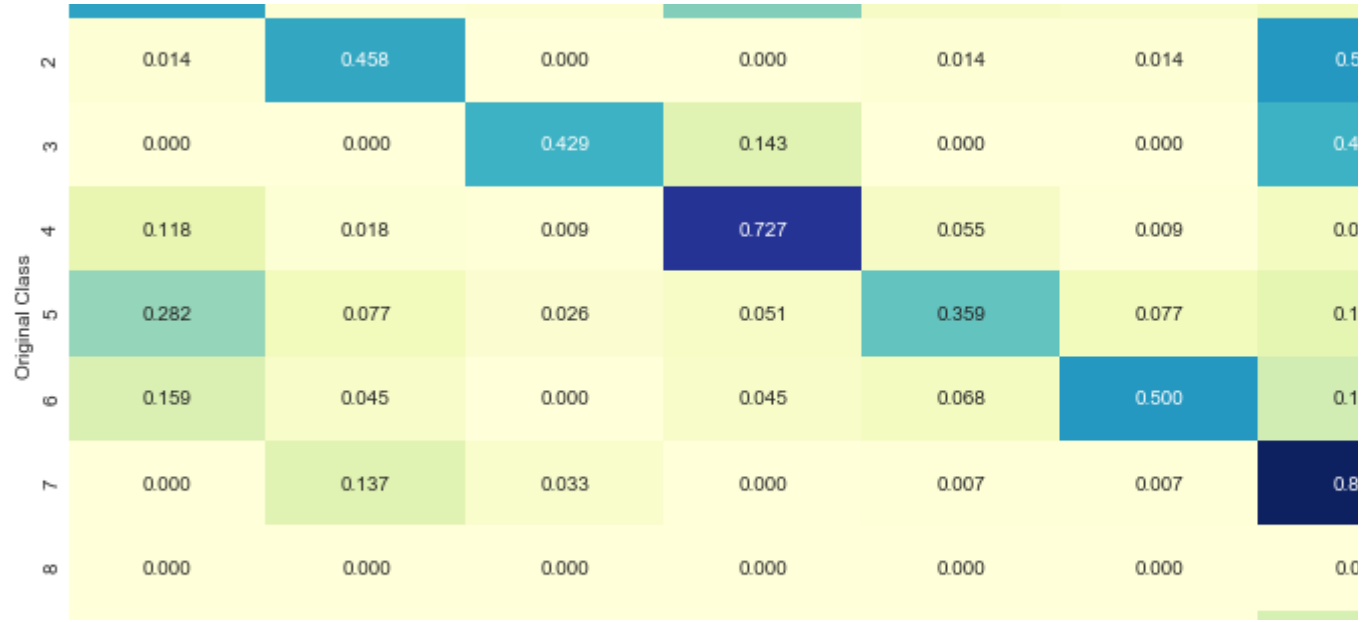
```
Log loss : 1.1531088456769596
Number of mis-classified points : 0.3815789473684211
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['Lr- Regression with class balance','Unigrams',0.6338,1.1531,1.1218,0.3815]
results.loc[4] = new
```

4.4.1.2. Without Class balancing

4.4.1.2.1. Hyper paramter tuning

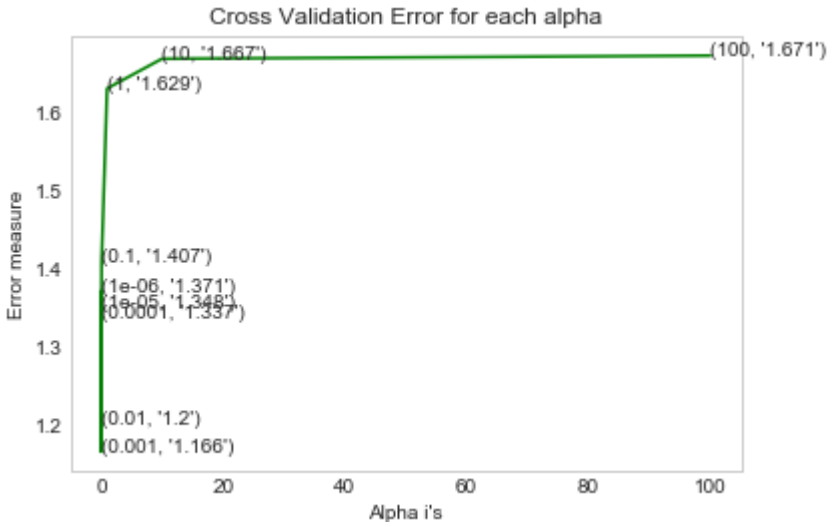
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
```



```
#
# 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 = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
```

```
        clf = SGDClassifier( alpha=i, penalty='l2', loss='log', random_state=42)
        clf.fit(train_x_unigram_onehotCoding, train_y_unigram)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_unigram_onehotCoding, train_y_unigram)
        sig_clf_probs = sig_clf.predict_proba(cv_x_unigram_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y_unigram, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-probability estimates
        print("Log Loss :",log_loss(cv_y_unigram, sig_clf_probs))

    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],str(txt)), (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()


    best_alpha = np.argmin(cv_log_error_array)
    clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_unigram_onehotCoding, train_y_unigram)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_unigram_onehotCoding, train_y_unigram)

    predict_y = sig_clf.predict_proba(train_x_unigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
    predict_y = sig_clf.predict_proba(cv_x_unigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
    predict_y = sig_clf.predict_proba(test_x_unigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.3706051959373866
for alpha = 1e-05
Log Loss : 1.3484603087633598
for alpha = 0.0001
Log Loss : 1.3365691324196889
for alpha = 0.001
Log Loss : 1.166097724858092
for alpha = 0.01
Log Loss : 1.2001288690098368
for alpha = 0.1
Log Loss : 1.4071725908954307
for alpha = 1
Log Loss : 1.6294017195612895
for alpha = 10
Log Loss : 1.6672942048822037
for alpha = 100
Log Loss : 1.6713860403575165
```



4.4.1.2.2. Testing model with best hyper parameters

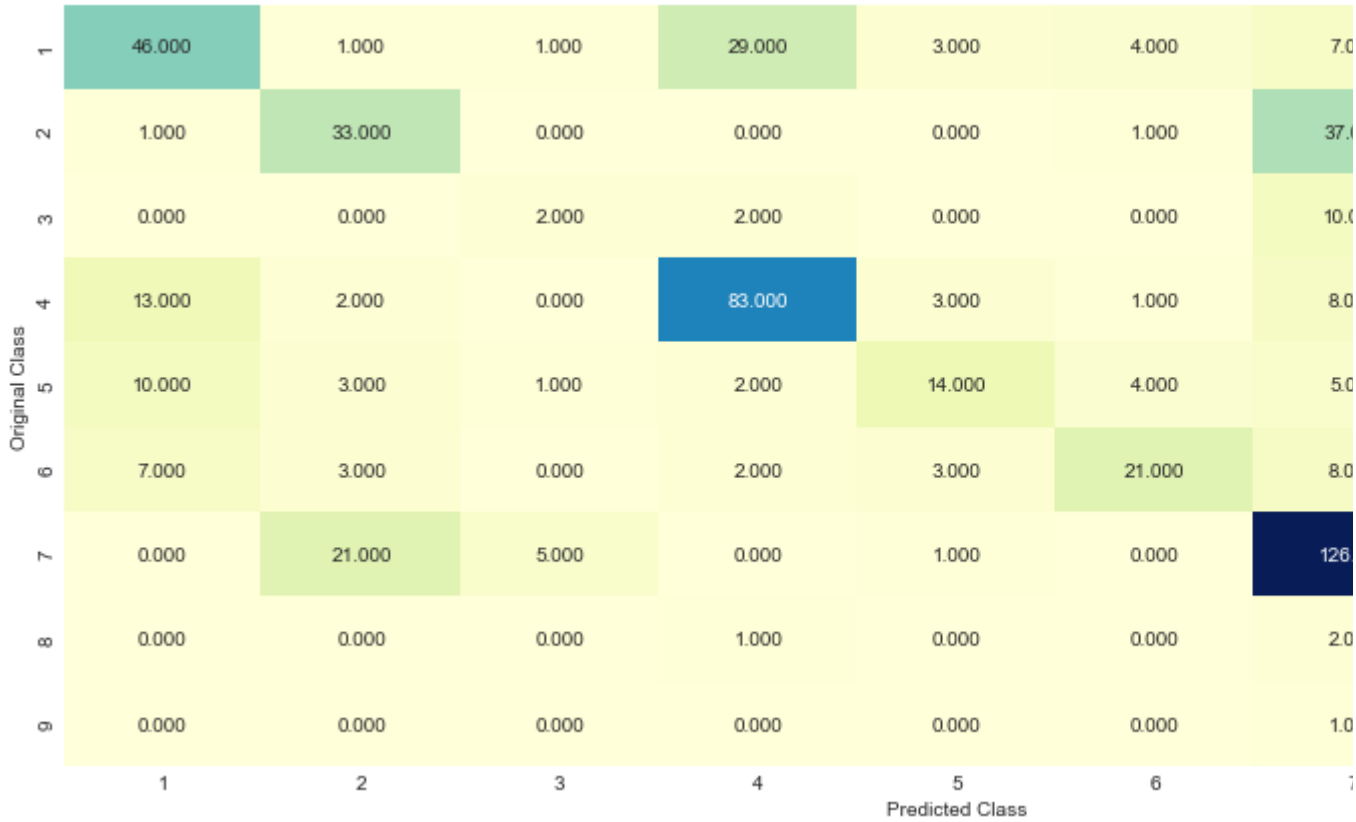
```
for values of best alpha =  0.001 The test log loss is: 1.1402442934188948
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

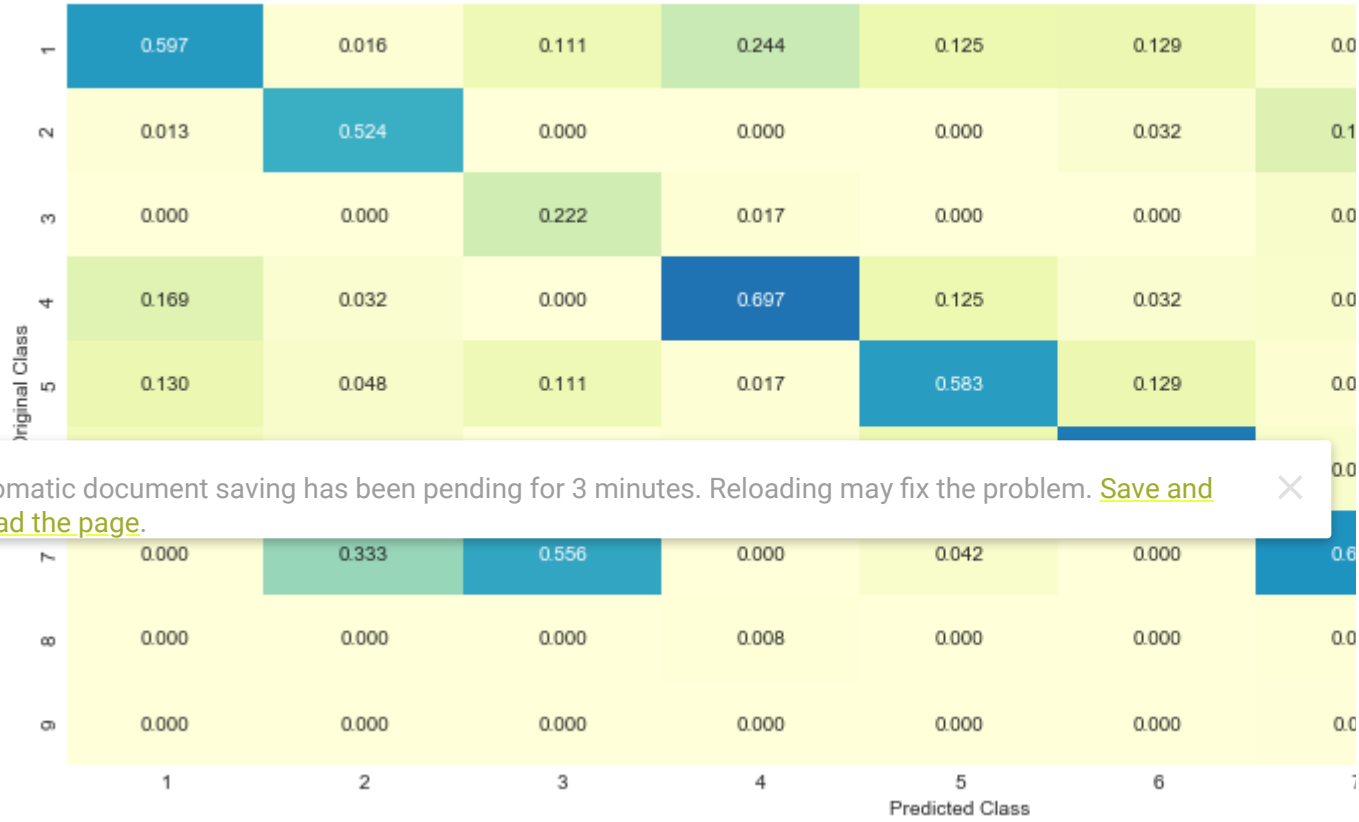


```
# predict(X)  Predict class labels for samples in X.

#------------------------------
# video link:
#------------------------------

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_unigram_onehotCoding, train_y_unigram, cv_x_unigram_onehot
```

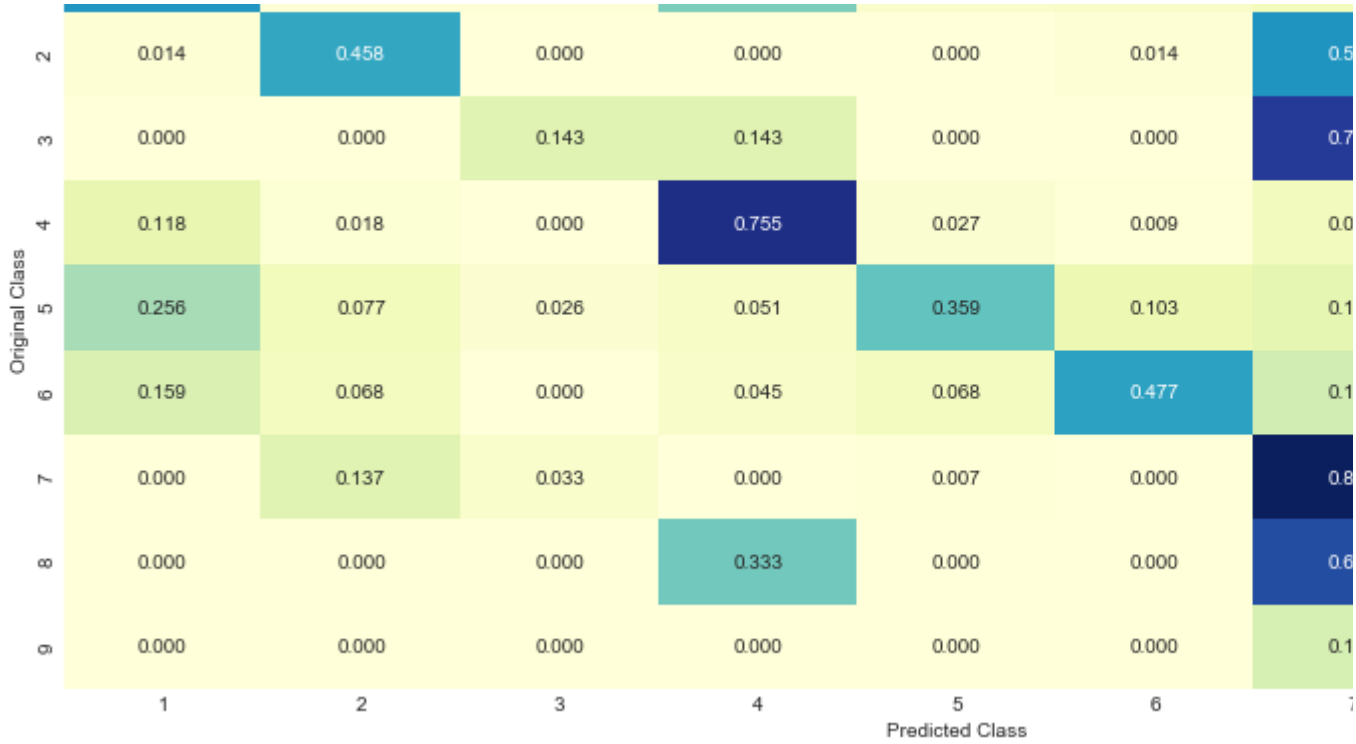
```
Log loss : 1.166097724858092
Number of mis-classified points : 0.37969924812030076
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



Automatic document saving has been pending for 3 minutes. Reloading may fix the problem. Save and reload the page.

```
new = ['Lr- Regression without class balnce','Unigram',0.6351,1.1660,1.1402,0.3796]
results.loc[5] = new
```

4.4.2. Bigrams

4.4.2.1. With Class balancing

```
# don't forget to normalize every feature
train_text_bigram_feature_onehotCoding = normalize(train_text_bigram_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
test_text_bigram_feature_onehotCoding = text_bigram_vectorizer.transform(test_df['TEXT'])
# don't forget to normalize every feature
test_text_bigram_feature_onehotCoding = normalize(test_text_bigram_feature_onehotCoding, axis=0)

# we use the same vectorizer that was trained on train data
cv_text_bigram_feature_onehotCoding = text_bigram_vectorizer.transform(cv_df['TEXT'])
# don't forget to normalize every feature
cv_text_bigram_feature_onehotCoding = normalize(cv_text_bigram_feature_onehotCoding, axis=0)
```

```
#https://stackoverflow.com/a/2258273/4084039
```

se

Automatic document saving has been pending for 3 minutes. Reloading may fix the problem. Save and reload the page.

```
# Number of words for a given frequency.
print(Counter(sorted_text_bigram_occur))
```

```
Counter({3: 144030, 4: 101748, 5: 70342, 6: 58318, 9: 41171, 7: 37842, 8: 33485, 10: 261
```

```
# merging gene, variance and text features

# building train, test and cross validation data sets
# a = [[1, 2],
#      [3, 4]]
```

```
# b = [[4, 5],
#      [6, 7]]
# hstack(a, b) = [[1, 2, 4, 5],
#                 [ 3, 4, 6, 7]]

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehot
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCod
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

train_x_bigram_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_bigram_feature_onehotC
train_y_bigram = np.array(list(train_df['Class']))

test_x_bigram_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_bigram_feature_onehotCodi
test_y_bigram = np.array(list(test_df['Class']))

cv_x_bigram_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_bigram_feature_onehotCoding)).t
cv_y_bigram = np.array(list(cv_df['Class']))
```

4.4.2.1.1. Hyper paramter tuning

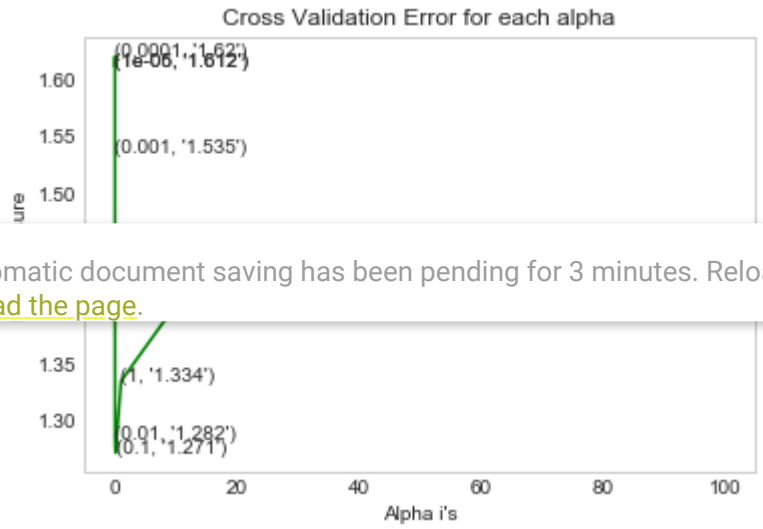
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# 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:
#-----------------------------------
```



```
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
    sig_clf_probs = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y_bigram, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y_bigram, sig_clf_probs))

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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_bigram_onehotCoding, train_y_bigram)

predict_y = sig_clf.predict_proba(train_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_bigram_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.6122551372792584
for alpha = 1e-05
Log Loss : 1.6119429191388706
for alpha = 0.0001
Log Loss : 1.620320371995538
for alpha = 0.001
Log Loss : 1.5348191362038057
for alpha = 0.01
Log Loss : 1.282424369470158
for alpha = 0.1
Log Loss : 1.2708401168260648
for alpha = 1
Log Loss : 1.3340173579219667
for alpha = 10
Log Loss : 1.4013414606301826
for alpha = 100
Log Loss : 1.4162411944273434
```



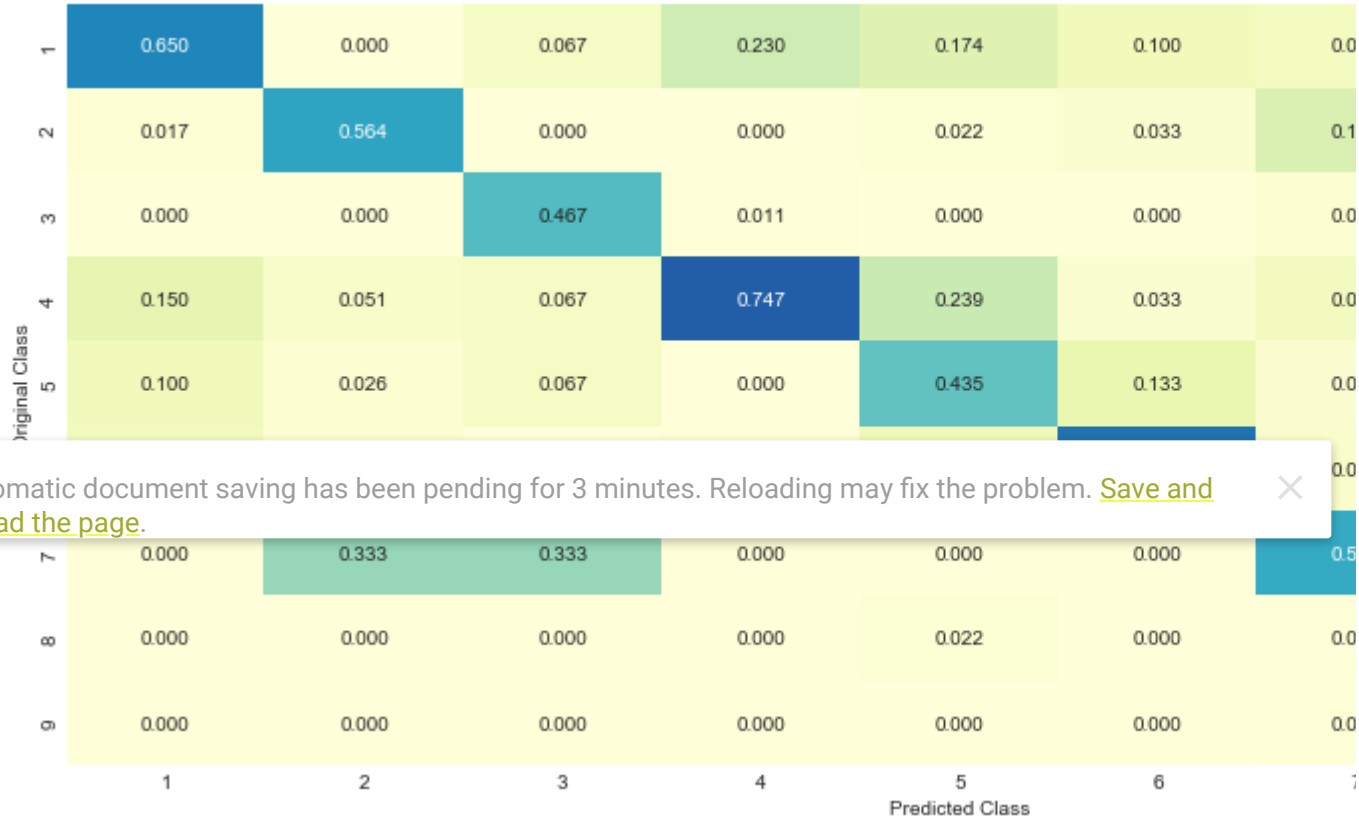

```
For values of best alpha =  0.1 The train log loss is: 0.8720574424921468
For values of best alpha =  0.1 The cross validation log loss is: 1.2708401168260648
For values of best alpha =  0.1 The test log loss is: 1.2329510001500568
```

4.4.2.1.2. Testing the model with best hyper paramters

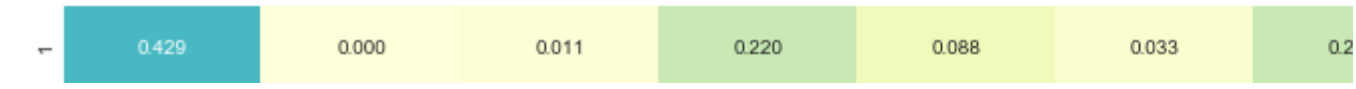
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)
# predict(X)  Predict class labels for samples in X.

#-------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intu
#-------------------------------
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', rand
predict_and_plot_confusion_matrix(train_x_bigram_onehotCoding, train_y_bigram, cv_x_bigram_onehotCod
```

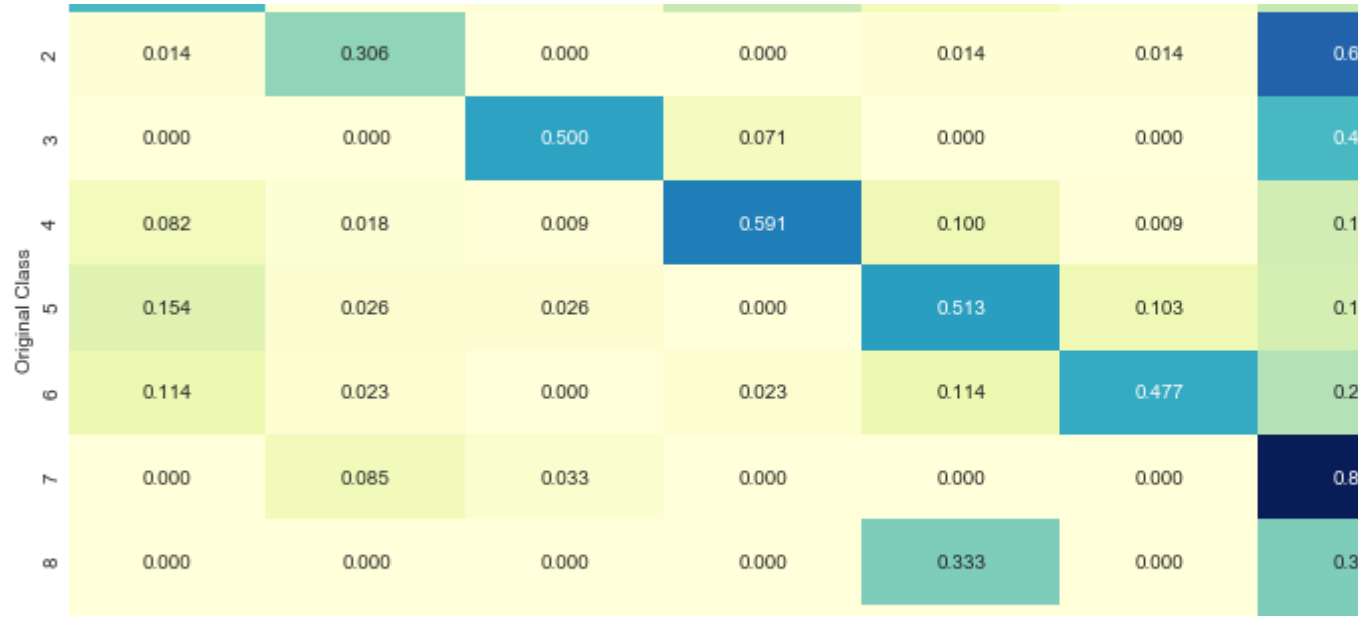
```
Log loss : 1.2708401168260648
Number of mis-classified points : 0.40977443609022557
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```






```
new = ['Lr- Regression with class balance','Bigram',0.8720,1.2708,1.2329,0.4097]
results.loc[6] = new
```

4.4.2.2. Without Class balancing

4.4.2.2.1. Hyper paramter tuning

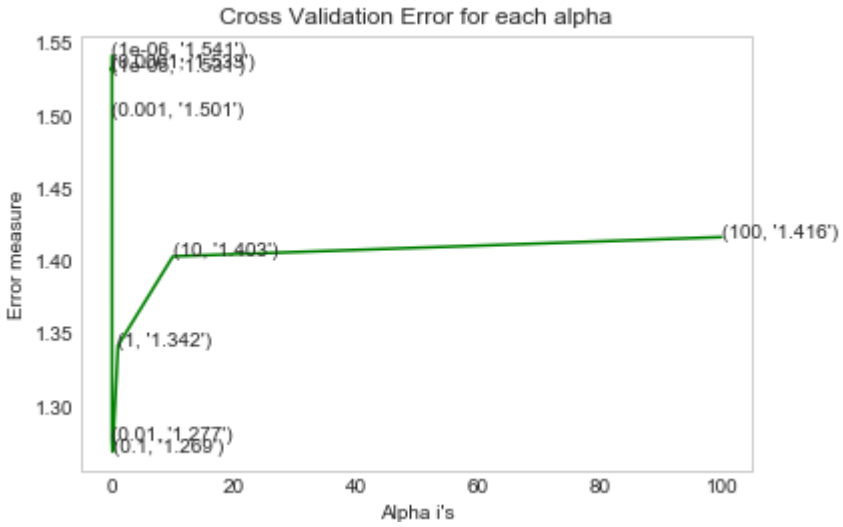
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# 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 Descent.
# predict(X)  Predict class labels for samples in X.

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
```



```
#
# 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 = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
```

```
        clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
        clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
        sig_clf_probs = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y_bigram, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        # to avoid rounding error while multiplying probabilites we use log-probability estimates
        print("Log Loss :",log_loss(cv_y_bigram, sig_clf_probs))

    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],str(txt)), (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()


    best_alpha = np.argmin(cv_log_error_array)
    clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_bigram_onehotCoding, train_y_bigram)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_bigram_onehotCoding, train_y_bigram)

    predict_y = sig_clf.predict_proba(train_x_bigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
    predict_y = sig_clf.predict_proba(cv_x_bigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
    predict_y = sig_clf.predict_proba(test_x_bigram_onehotCoding)
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for alpha = 1e-06
Log Loss : 1.5412672460828307
for alpha = 1e-05
Log Loss : 1.5309826483732532
for alpha = 0.0001
Log Loss : 1.5331334008745652
for alpha = 0.001
Log Loss : 1.500539564042237
for alpha = 0.01
Log Loss : 1.2771512839113088
for alpha = 0.1
Log Loss : 1.2687869269163154
for alpha = 1
Log Loss : 1.3418613843970917
for alpha = 10
Log Loss : 1.4031454597545883
for alpha = 100
Log Loss : 1.4164125384744848
```



4.4.2.2.2. Testing model with best hyper parameters

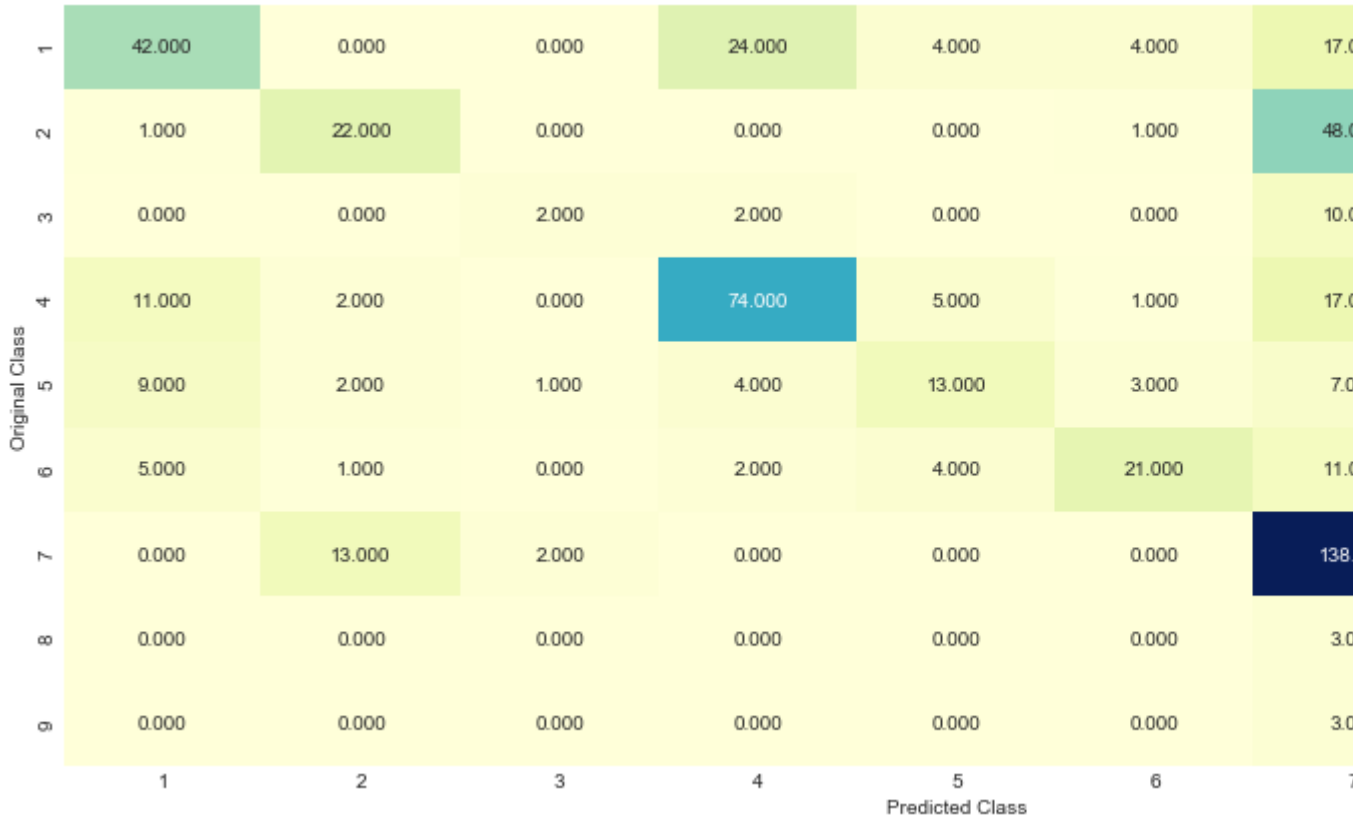
```
for values of best alpha =  0.1 the test log loss is: 1.2377498027113152
```

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)
```

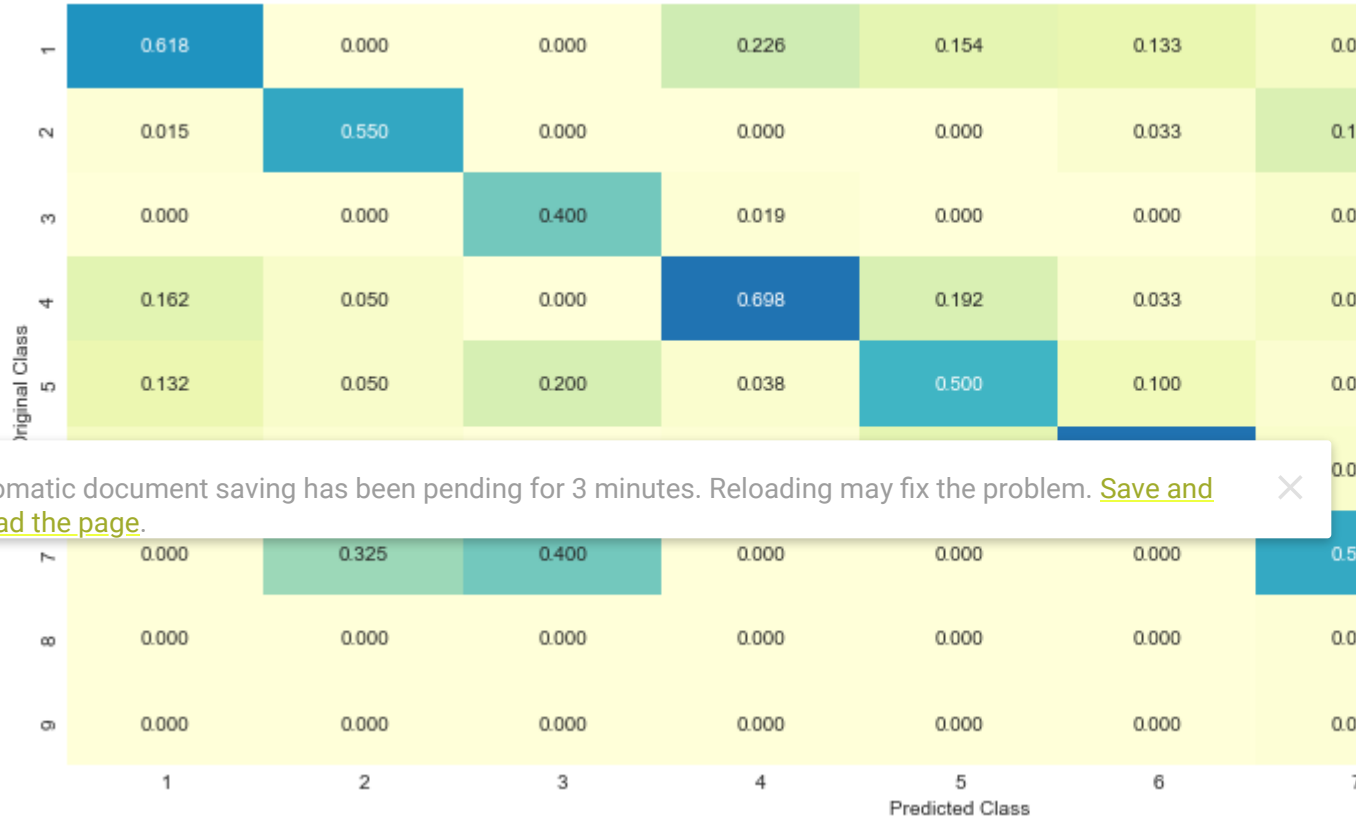


```
# predict(X)  Predict class labels for samples in X.

#------------------------------
# video link:
#------------------------------

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_bigram_onehotCoding, train_y_bigram, cv_x_bigram_onehotCod
```

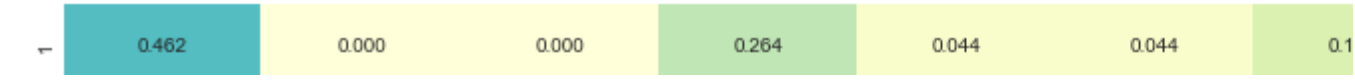
```
Log loss : 1.2687869269163154
Number of mis-classified points : 0.40789473684210525
-------------------- Confusion matrix --------------------
```

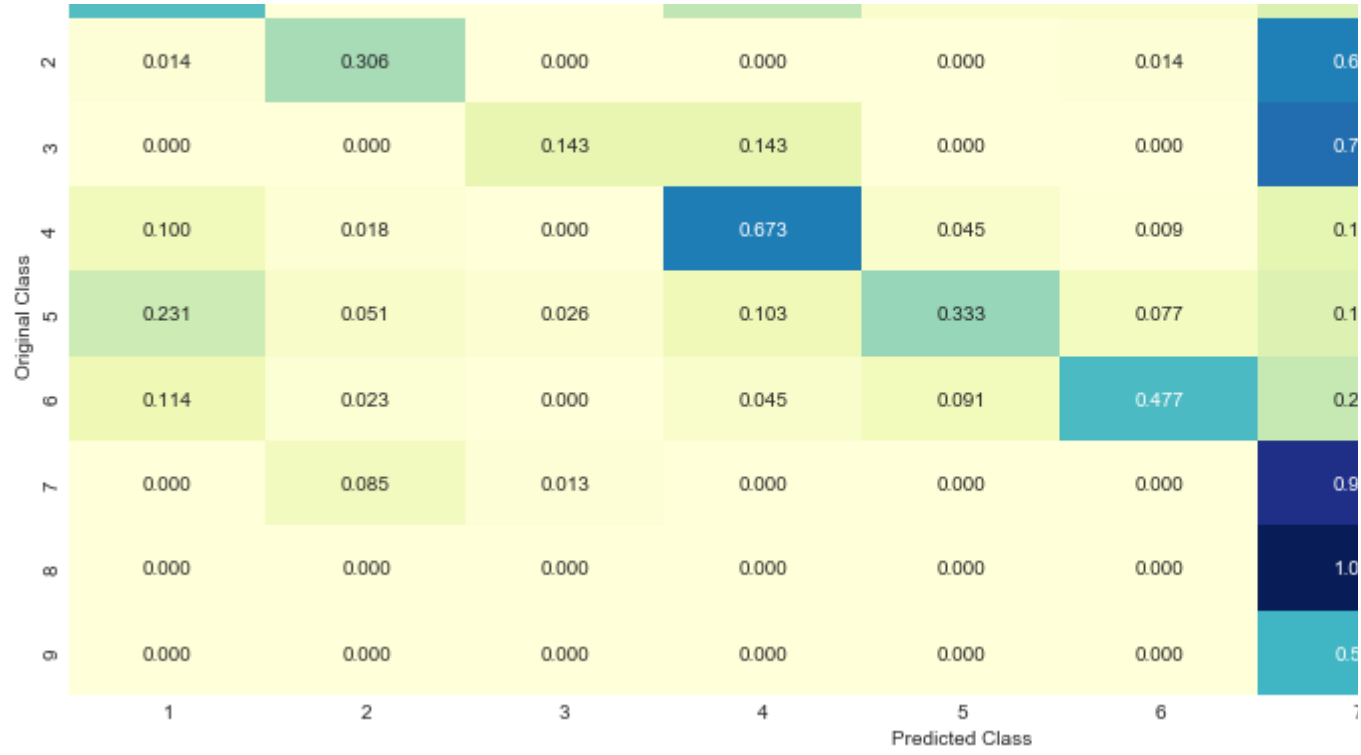



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```





```
new = ['Lr-Regression without class balance','Bigrams',0.8477,1.2687,1.2377,0.4078]
results.loc[7] = new
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

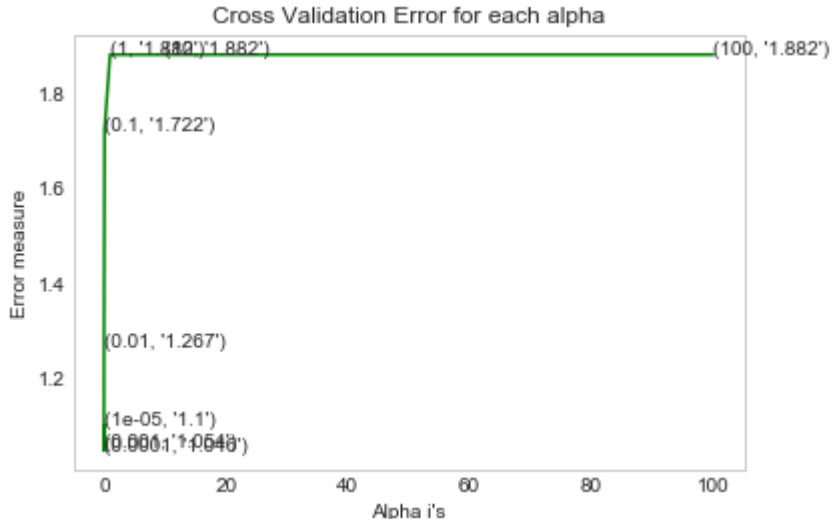
```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/mo

# ---------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', rand

# Some of methods of SVM()
# fit(X, y, [sample_weight])  Fit the SVM model according to the given training data.
# predict(X)  Perform classification on samples in X.
# ---------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-d
# ---------------------------------
```



```
# 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 = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
```

```
for i in alpha:
    print("for C =", i)
#     clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', ra
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train, p
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:",log_loss
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test, pre
```

```
for C = 1e-05
Log Loss : 1.1003092884982582
for C = 0.0001
Log Loss : 1.0464418333491776
for C = 0.001
Log Loss : 1.0540680333484442
for C = 0.01
Log Loss : 1.267242073470559
for C = 0.1
Log Loss : 1.7224918813984675
for C = 1
Log Loss : 1.8819846043778437
for C = 10
Log Loss : 1.8819845693991568
for C = 100
Log Loss : 1.8819846594303946
```



4.4.2. Testing model with best hyper parameters

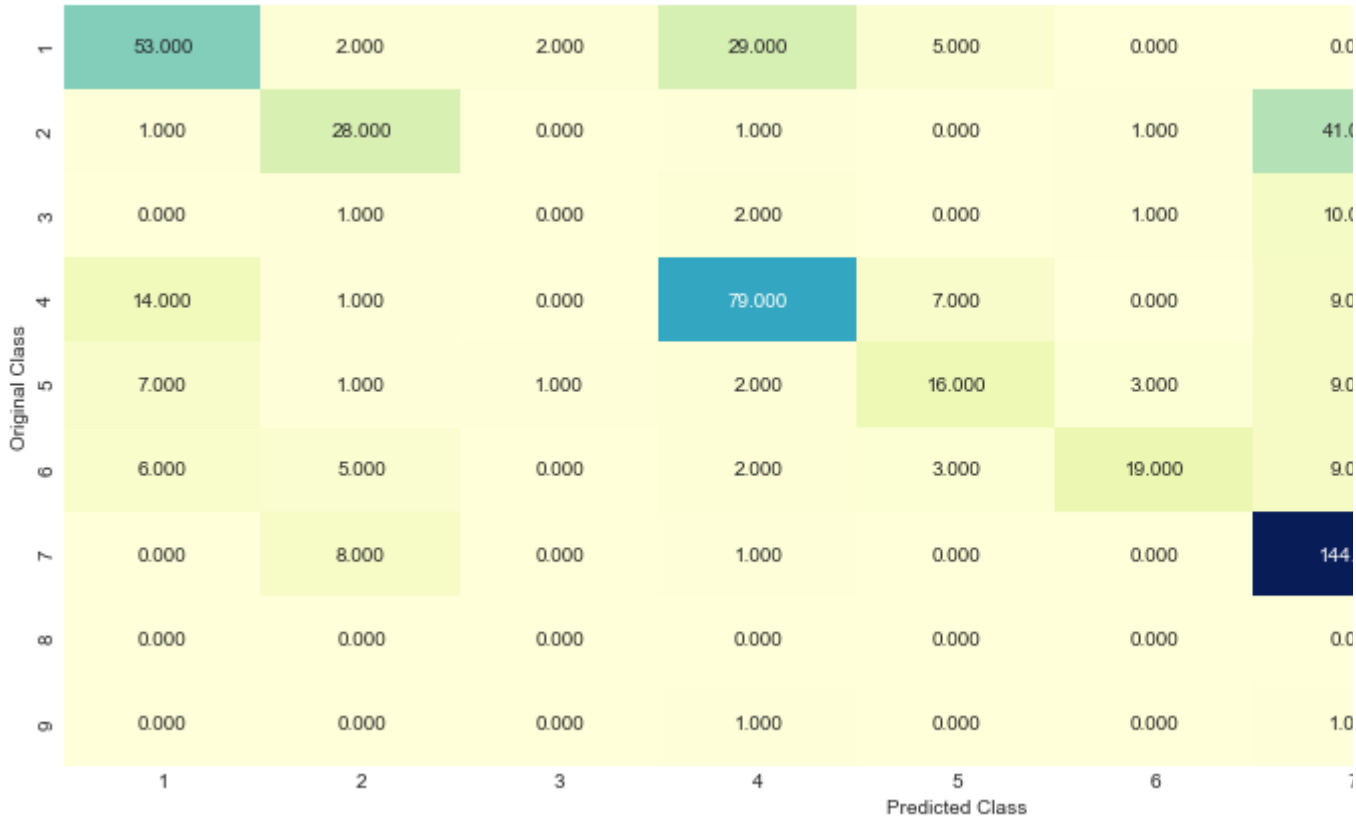
```
for values of best alpha =  0.0001 the test log loss is: 1.0879253544392445
```

```
# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/mo

# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', rand

# Some of methods of SVM()
```

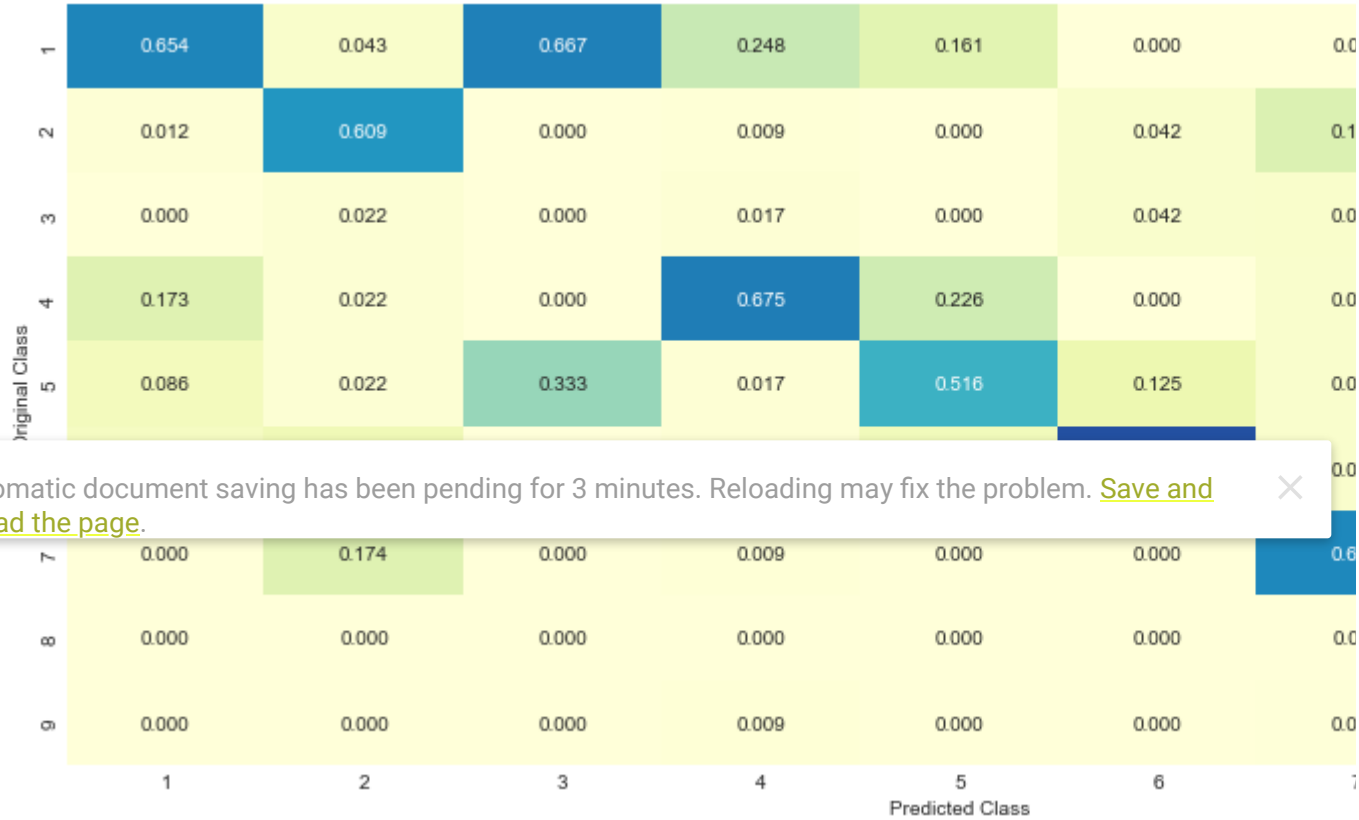


```
# ------------------------------


# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42,class_weigh
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

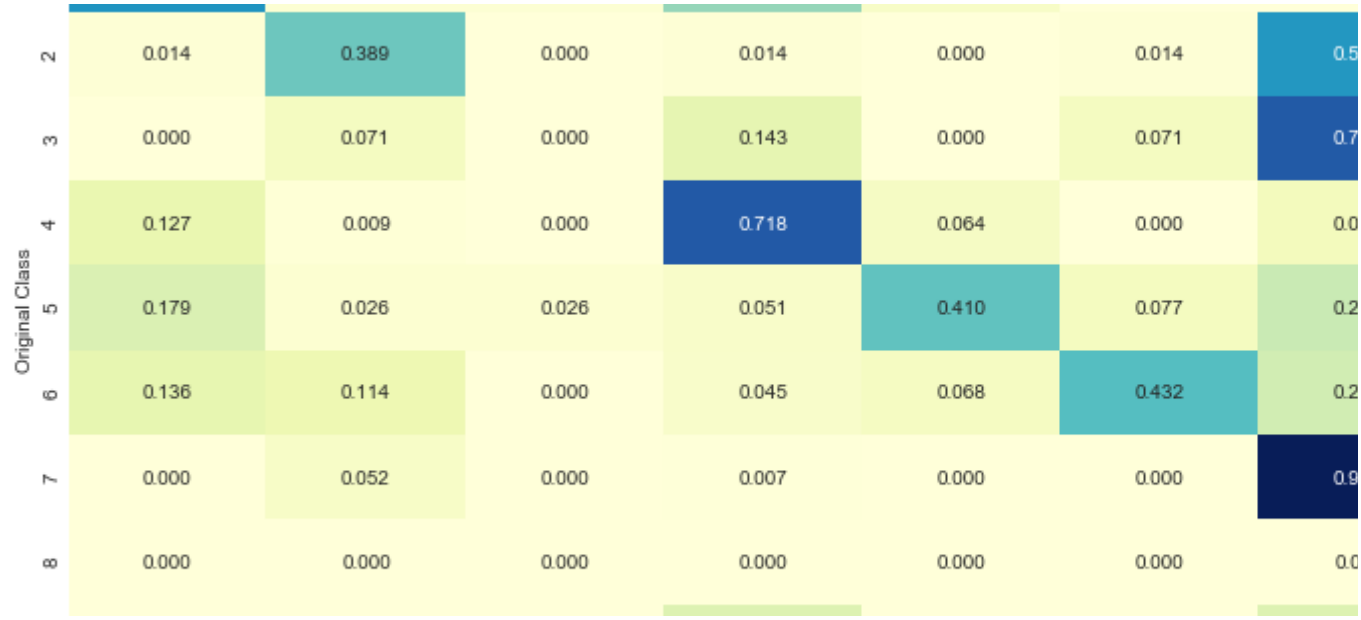
```
Log loss : 1.0464418333491776
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['Linear SVM','tfidf',0.5088,1.0464,1.0879,0.3533]
results.loc[8] = new
```

4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0584 0.0513 0.0041 0.0278 0.0177 0.0082 0.8297 0.0014
```



```
298 Text feature [1169] present in test data point [True]
324 Text feature [108] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
```

```
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_po
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0419 0.0445 0.0039 0.1227 0.0222 0.0027 0.758  0.0023
Actual Class : 7
--------------------------------------------------
223 Text feature [101] present in test data point [True]
235 Text feature [114] present in test data point [True]
324 Text feature [108] present in test data point [True]
333 Text feature [128] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sam
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
# 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/random-forest-
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
```



```
#
# 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:
#-------------------------------------
from sklearn.ensemble import RandomForestClassifier
alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
```

```
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42,
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[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()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation log loss is
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",log_loss(
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2355616485579877
for n_estimators = 100 and max depth =  10
Log Loss : 1.2628410862243704
for n_estimators = 200 and max depth =  5
Log Loss : 1.2195311732814633
for n_estimators = 200 and max depth =  10
Log Loss : 1.2445476312631405
for n_estimators = 500 and max depth =  5
Log Loss : 1.2074168254795592
for n_estimators = 500 and max depth =  10
Log Loss : 1.238624481164977
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2028184786159801
```



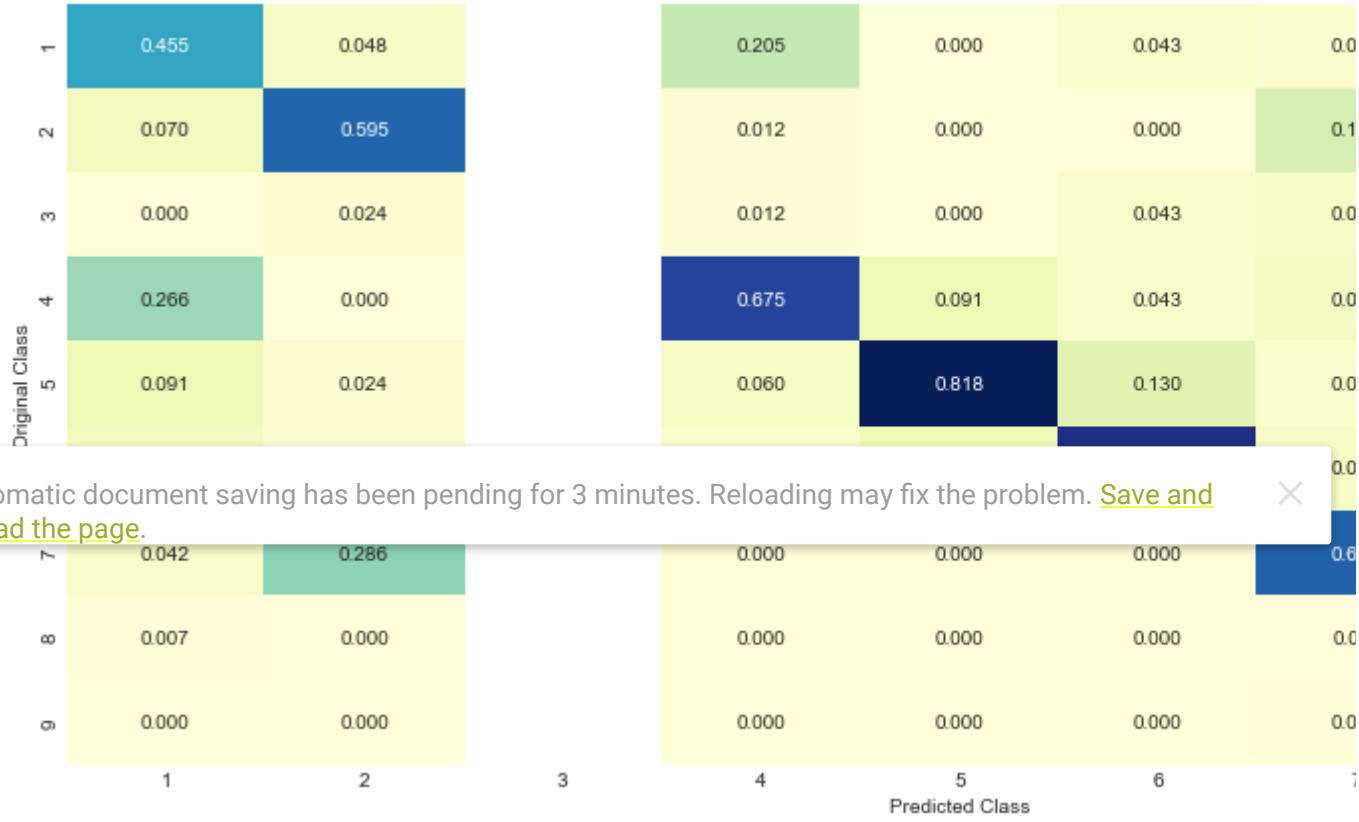
```
Log Loss : 1.2004522570754632
for n_estimators = 2000 and max depth =  10
Log Loss : 1.237167733701181
For values of best estimator =  2000 The train log loss is: 0.859766604710589
For values of best estimator =  2000 The cross validation log loss is: 1.200452257075463
For values of best estimator =  2000 The test log loss is: 1.230219509866679
```

4.5.2. Testing model with best hyper parameters (One Hot Encoding)

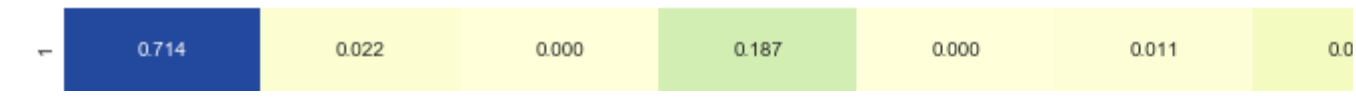
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sam
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
# 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/random-forest-
# --------------------------------

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)
```

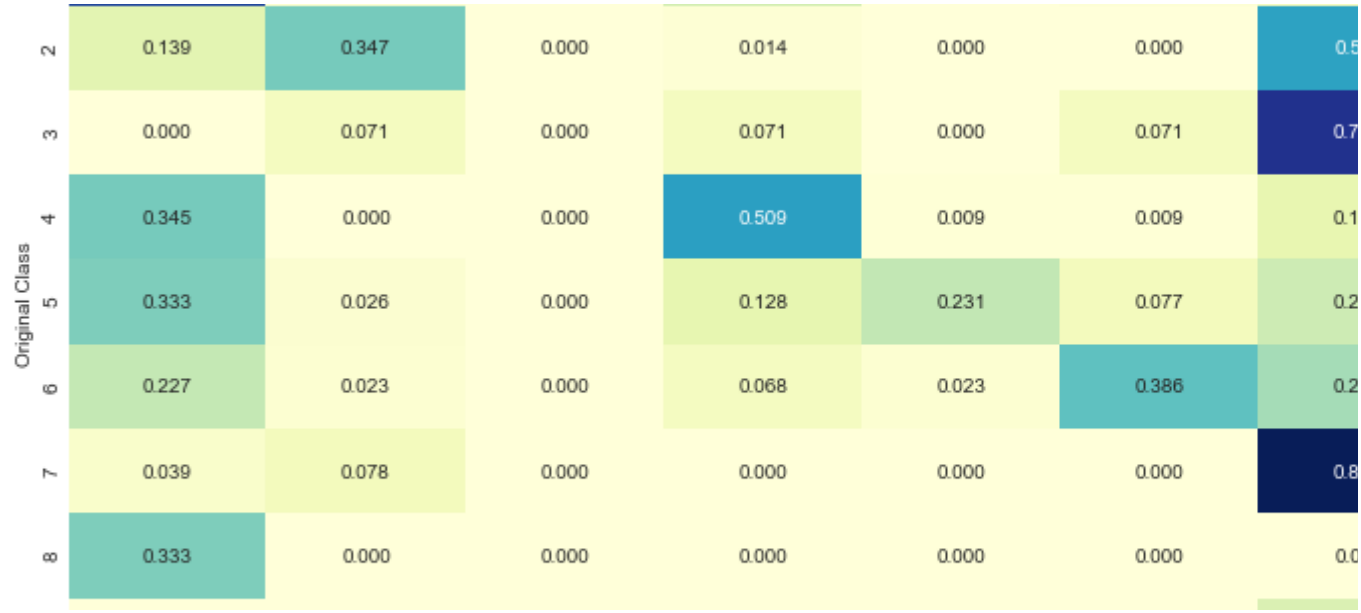
```
Log loss : 1.2004522570754632
Number of mis-classified points : 0.41353383458646614
-------------------- Confusion matrix --------------------
```


```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```






```
new = ['Random Forest with One hot encoding','tfidf',0.8597,1.2004,1.2302,0.4135]
results.loc[9] = new
```

Predicted Class

4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
# test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```



```
51
--------------------------------------------------
4 Text feature [100] present in test data point [True]
73 Text feature [1020] present in test data point [True]
99 Text feature [113] present in test data point [True]
Out of the top  100  features  3 are present in query point
```

4.5.3.2. Inorrectly Classified point

```
test_point_index = 100
no_feature = 100
```

```
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_poin
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.1789 0.1351 0.0159 0.1488 0.0519 0.0406 0.4106 0.0104
Actuall Class : 7
--------------------------------------------------
4 Text feature [100] present in test data point [True]
37 Text feature [1000] present in test data point [True]
69 Text feature [114] present in test data point [True]
76 Text feature [118k] present in test data point [True]
99 Text feature [113] present in test data point [True]
Out of the top  100  features  5 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

```
# -------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sam
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
# 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/random-forest-
# -------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sk
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
```



```
# 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 = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42,
```

```
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
'''
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[:,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[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()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y_t
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:",l
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:",log_loss(y_te
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.0331173397244684
for n_estimators = 10 and max depth =  3
Log Loss : 1.4629978403358959
for n_estimators = 10 and max depth =  5
Log Loss : 1.343328189939809
for n_estimators = 10 and max depth =  10
Log Loss : 1.8321577759870222
for n_estimators = 50 and max depth =  2
Log Loss : 1.5926260051602965
for n_estimators = 50 and max depth =  3
Log Loss : 1.3974460377329616
for n_estimators = 50 and max depth =  5
Log Loss : 1.2759609177882594
for n_estimators = 50 and max depth =  10
Log Loss : 1.6008817237601525
for n_estimators = 100 and max depth =  2
Log Loss : 1.5219818300791967
for n_estimators = 100 and max depth =  3
Log Loss : 1.4741425588978514
for n_estimators = 100 and max depth =  5
Log Loss : 1.2810235202720233
for n_estimators = 100 and max depth =  10
Log Loss : 1.6979074937036185
for n_estimators = 200 and max depth =  2
Log Loss : 1.5686981091364645
for n_estimators = 200 and max depth =  3
Log Loss : 1.4660015583366695
for n_estimators = 200 and max depth =  5
Log Loss : 1.303276861102754
for n_estimators = 200 and max depth =  10
Log Loss : 1.744593542747558
for n_estimators = 500 and max depth =  2
Log Loss : 1.644106268519709
for n_estimators = 500 and max depth =  3
Log Loss : 1.4972210523171363
for n_estimators = 500 and max depth =  5
Log Loss : 1.2992591755363099
for n_estimators = 500 and max depth =  10
Log Loss : 1.7550077472666705
for n_estimators = 1000 and max depth =  2
Log Loss : 1.6113584639620586
```



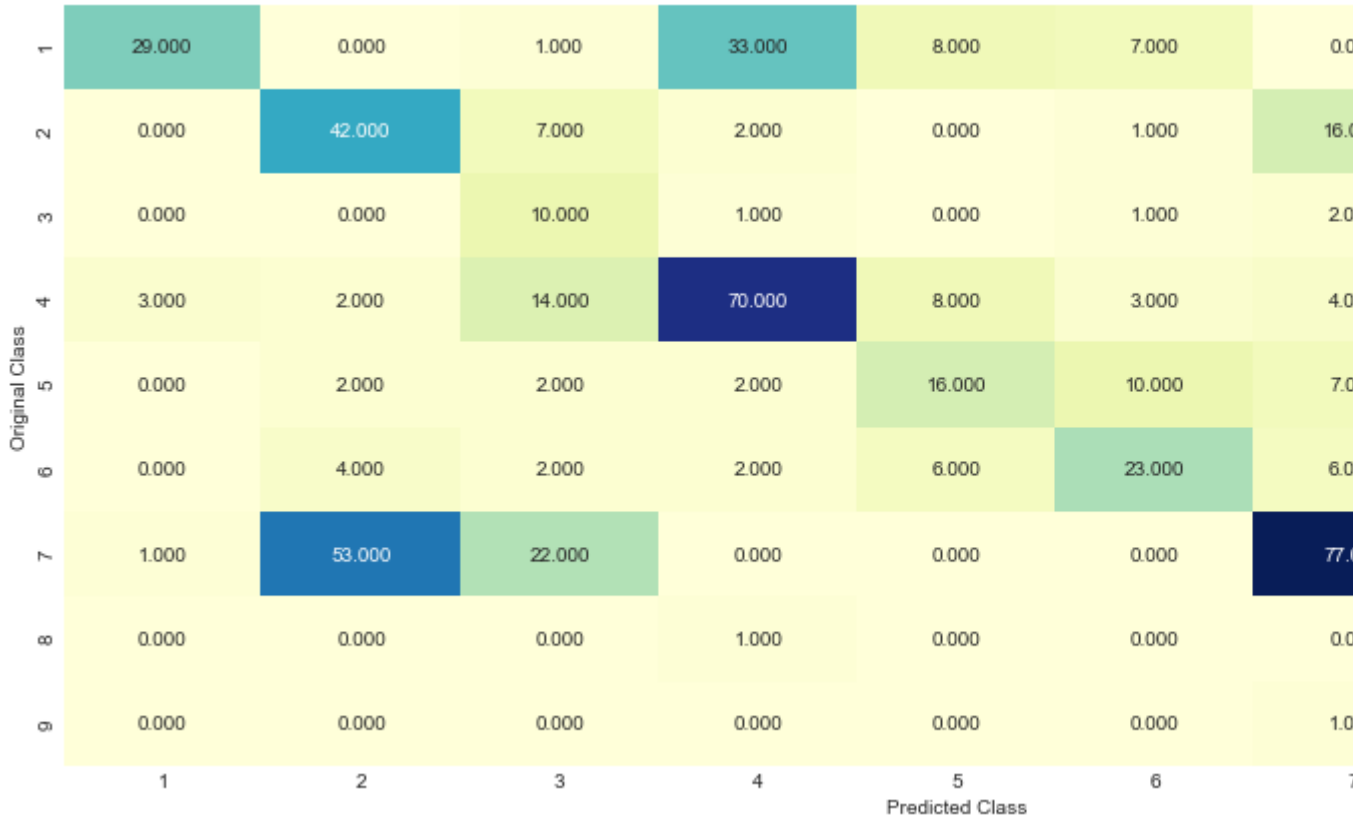
```
Log Loss : 1.3041078264930388
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7198001296919392
For values of best alpha =  50 The train log loss is: 0.05558936116994926
```

4.5.4. Testing model with best hyper parameters (Response Coding)

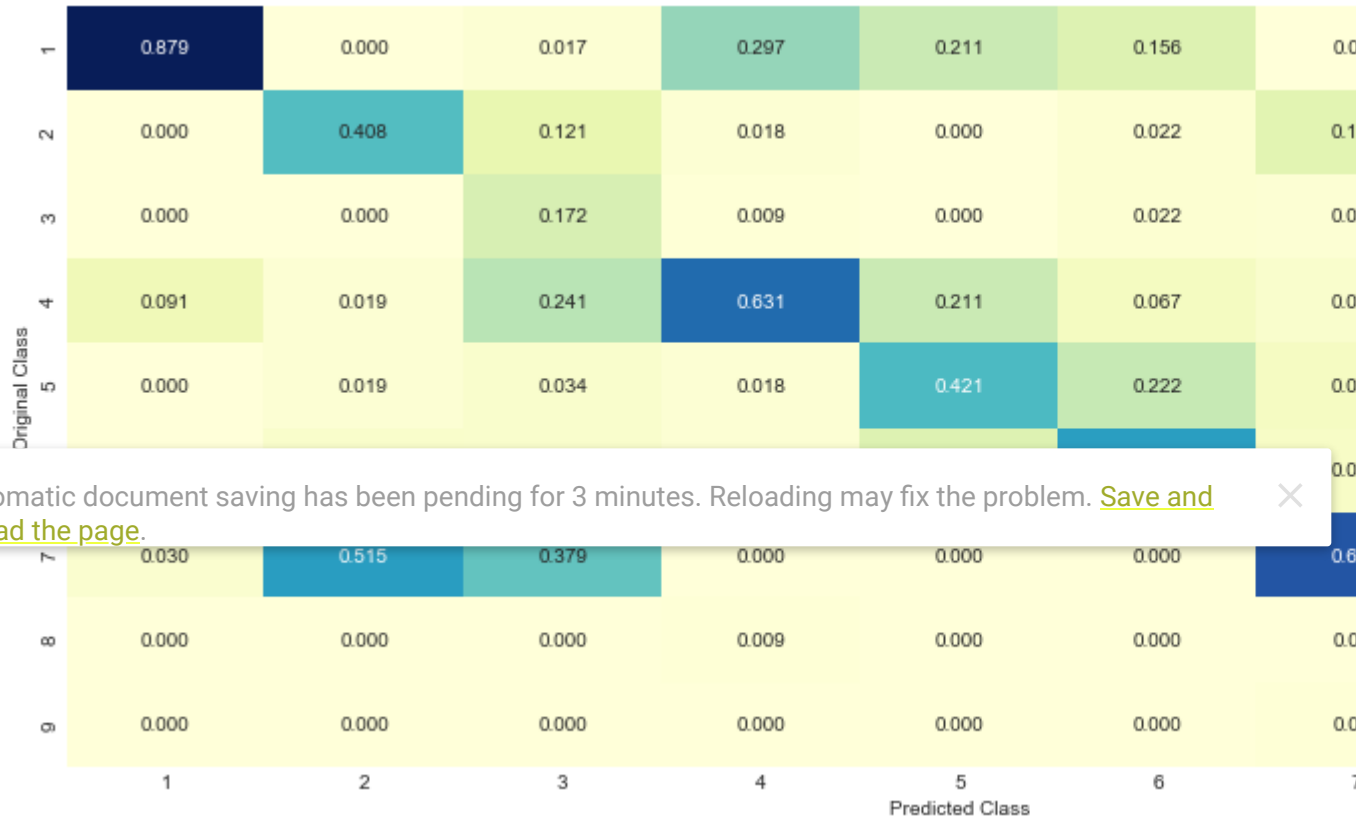
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sam
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
```

```
    # 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/random-forest-
    # --------------------------------

    clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alp
    predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)
```

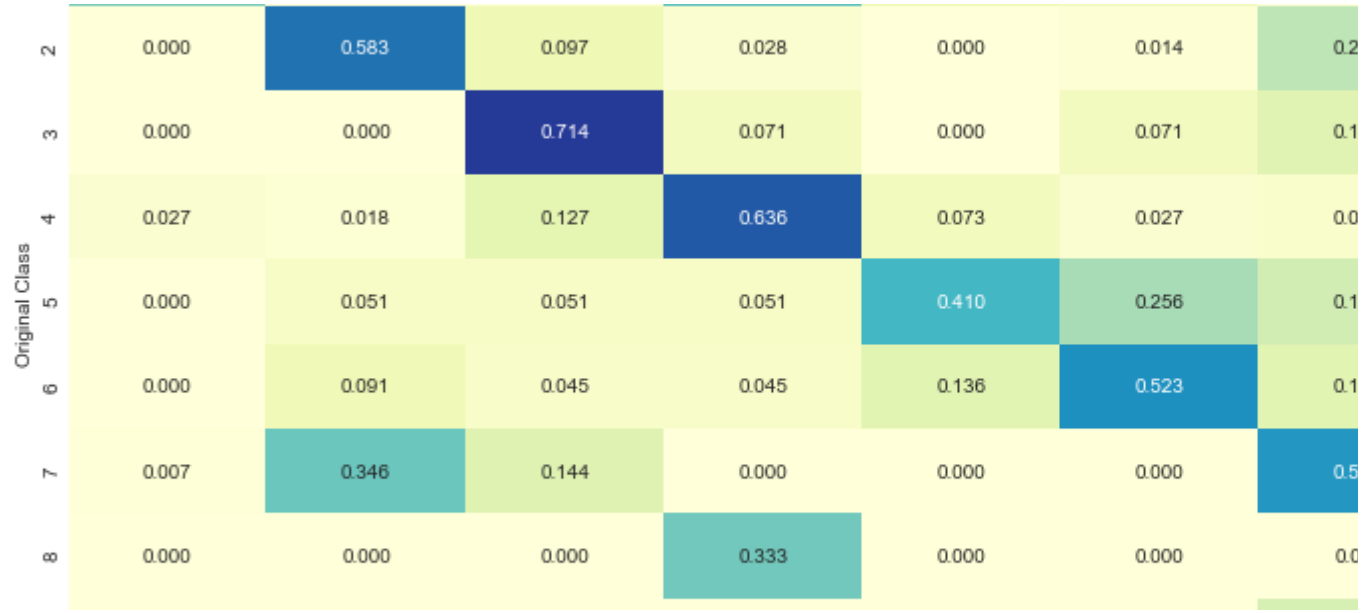
```
Log loss : 1.2759609177882596
Number of mis-classified points : 0.4868421052631579
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['Random Forest with Response code','tfidf',0.0555,1.2759,1.3283,0.4868]
results.loc[10] = new
```

4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)


test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_po
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0271 0.1865 0.1814 0.0231 0.0261 0.0449 0.4268 0.0737
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
```

4.5.5.2. Incorrectly Classified point

```
test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_po
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0021 0.0038 0.002  0.0029 0.001  0.0024 0.9826 0.0021
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Text is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Gene is important feature
Text is important feature
Text is important feature
Variation is important feature
Text is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_ite
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0
# class_weight=None, warm_start=False, average=False, n_iter=None)
```



```
# predict(X)  Predict class labels for samples in X.

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


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/mo
# ------------------------------
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', rand

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

```
# predict(X)  Perform classification on samples in X.
# --------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-d
# --------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn.org/stable/mo
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sam
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_im
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0,
# 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/random-forest-
# --------------------------------

clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
clf1.fit(train_x_onehotCoding, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
clf2.fit(train_x_onehotCoding, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")


clf3 = MultinomialNB(alpha=0.001)
clf3.fit(train_x_onehotCoding, train_y)
sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotC
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_oneh
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_pr
    sclf.fit(train_x_onehotCoding, train_y)
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    print("Log Loss : %0.2f" % (log_error))
    if best_alpha > log_error:
        best_alpha = log_error
```

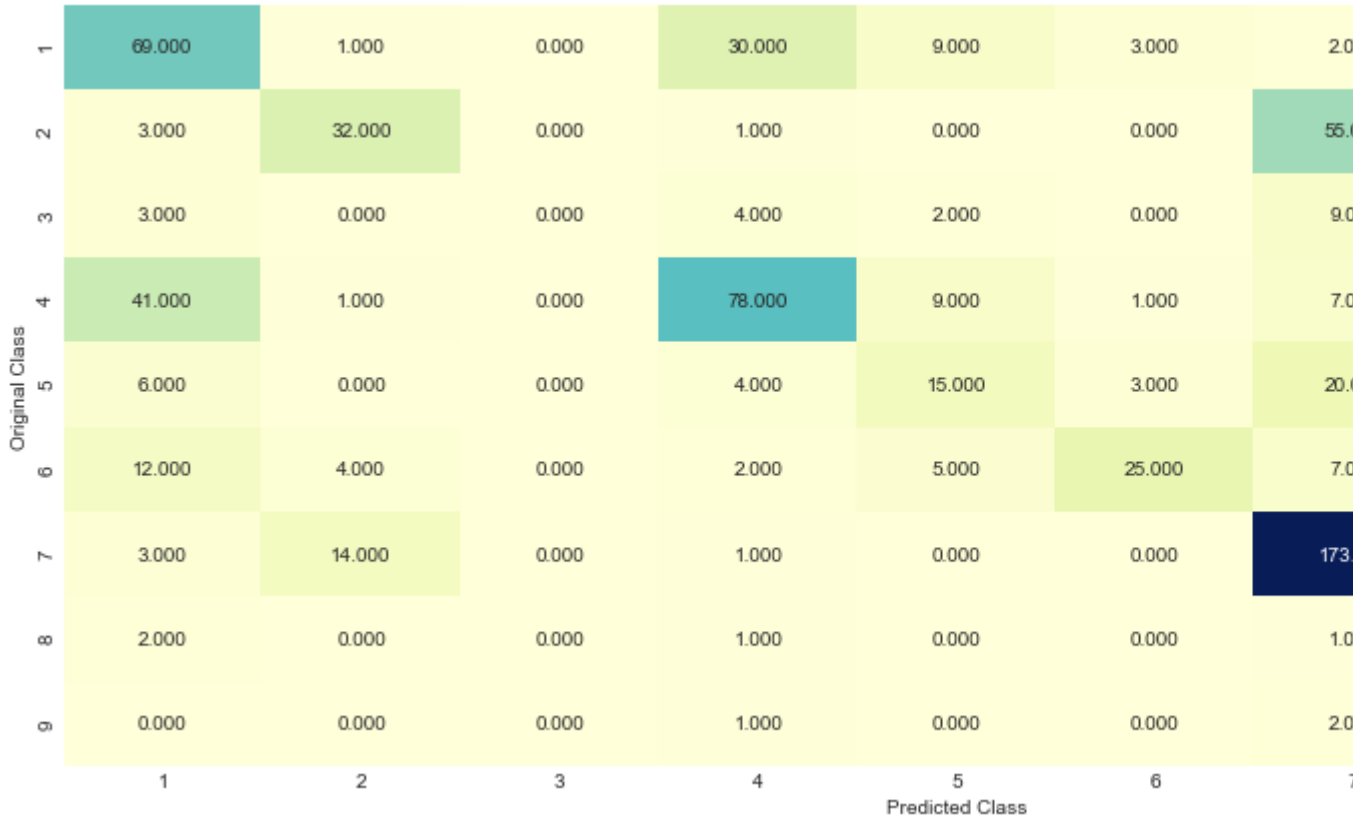
```
Logistic Regression :  Log Loss: 1.00
Support vector machines : Log Loss: 1.88
Naive Bayes : Log Loss: 1.20
-------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.031
```

4.7.2 testing the model with the best hyper parameters

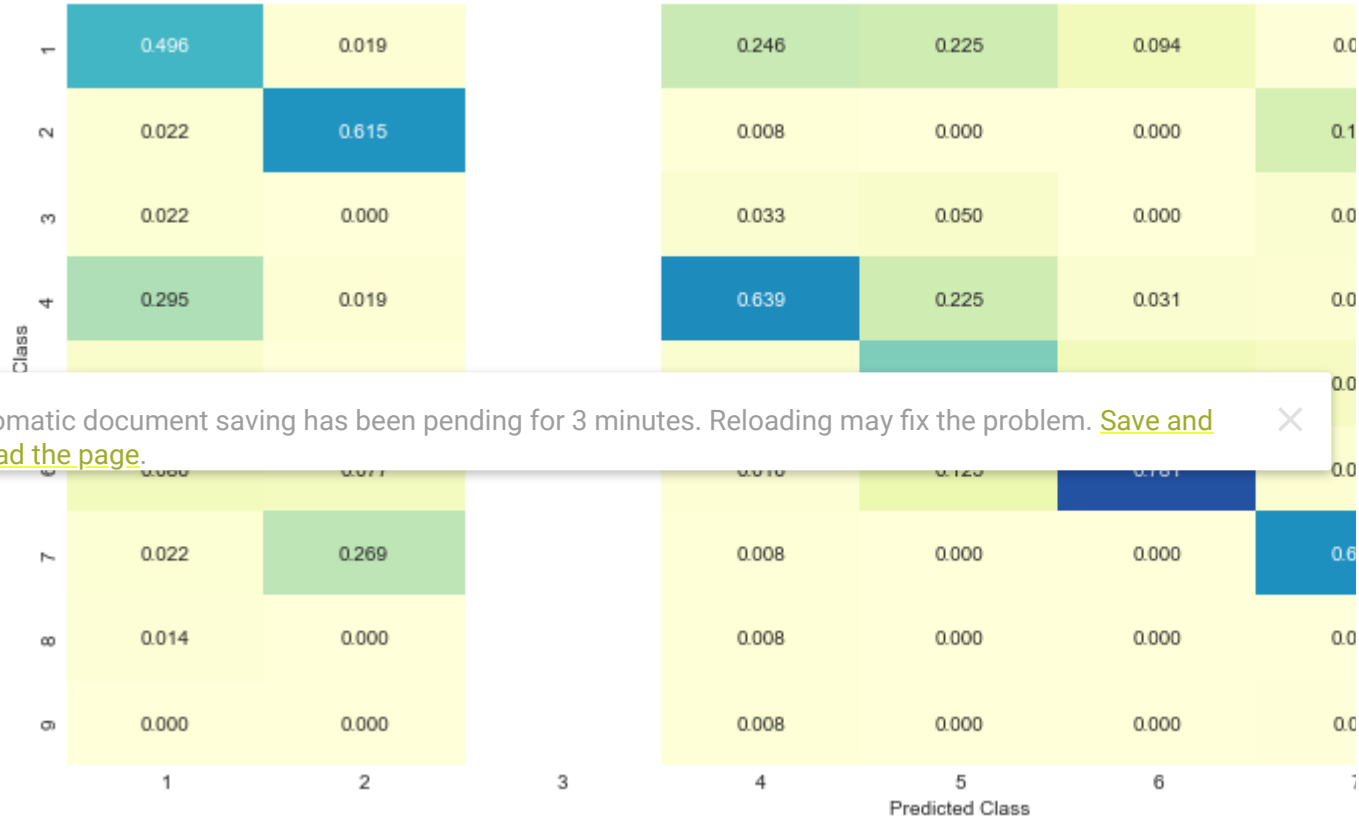
```
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.897
```

```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probas
sclf.fit(train_x_onehotCoding, train_y)

log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
print("Log loss (train) on the stacking classifier :",log_error)

log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
print("Log loss (CV) on the stacking classifier :",log_error)

log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
print("Log loss (test) on the stacking classifier :",log_error)

print("Number of missclassified point :", np.count_nonzero((sclf.predict(test_x_onehotCoding)- test_
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

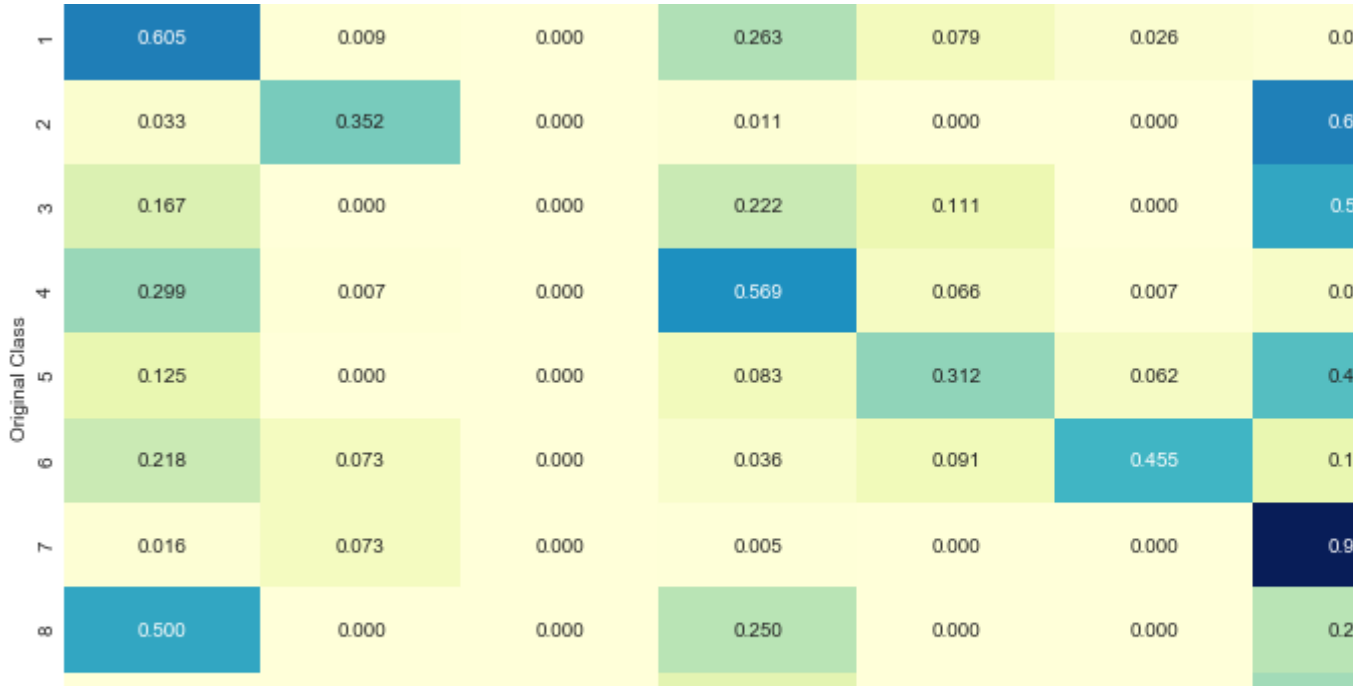
```
Log loss (train) on the stacking classifier : 0.5362107005580392
Log loss (CV) on the stacking classifier : 1.1722226690977273
Log loss (test) on the stacking classifier : 1.1956332648243035
Number of missclassified point : 0.4045112781954887
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```



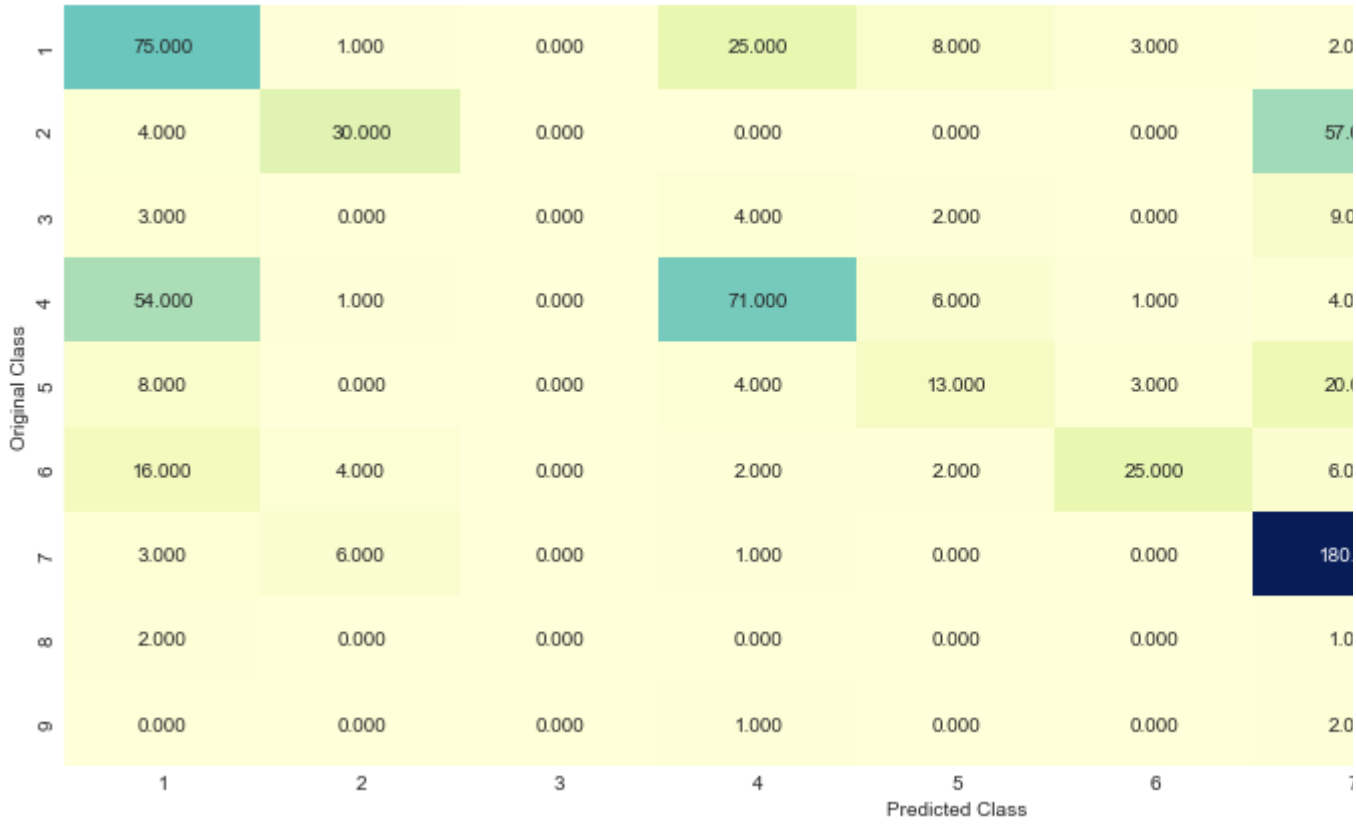

```
new = ['staking','tfidf',0.5362,1.1722,1.1956,0.4045]
results.loc[11] = new
```

Predicted Class

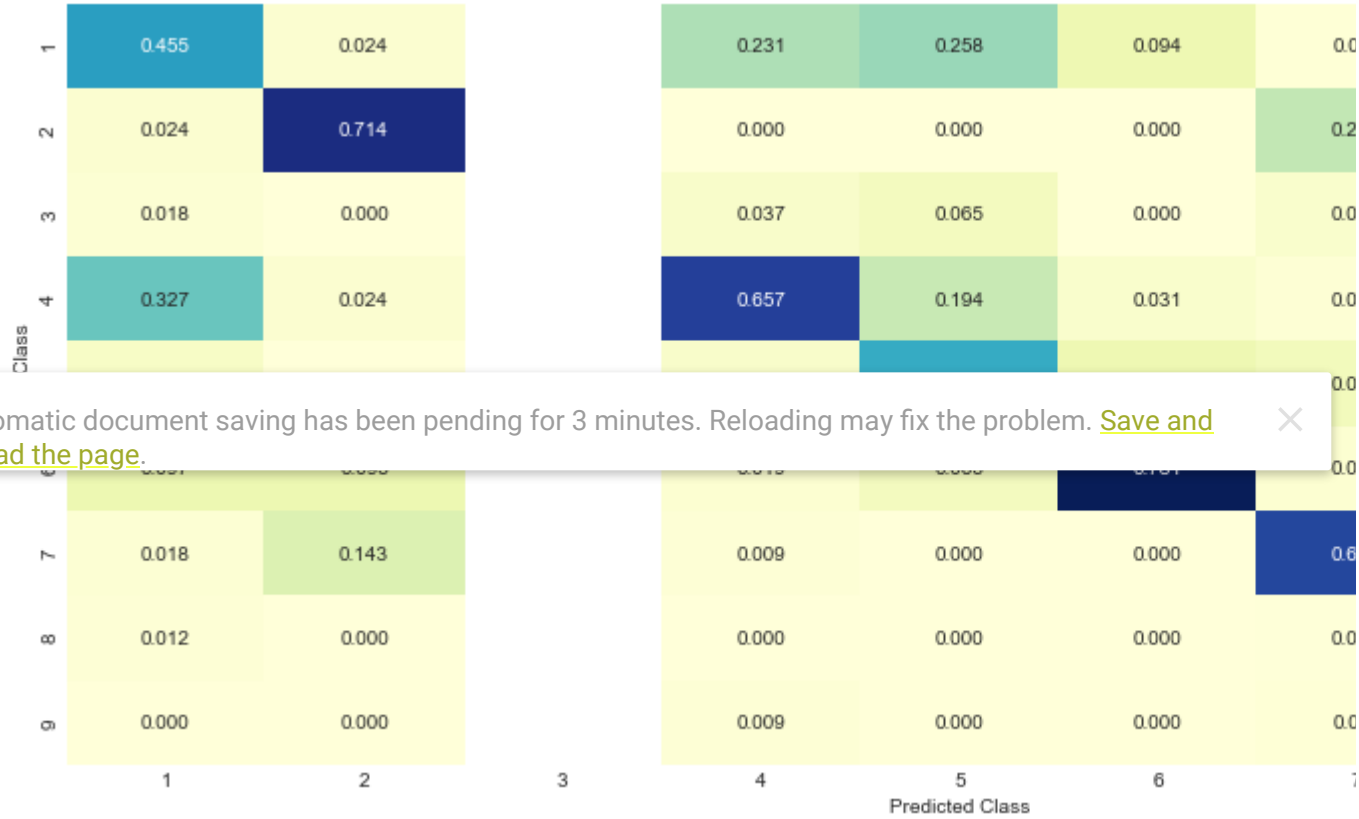
4.7.3 Maximum Voting classifier

```
#Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
from sklearn.ensemble import VotingClassifier
vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_one
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehot
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding)- test_
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

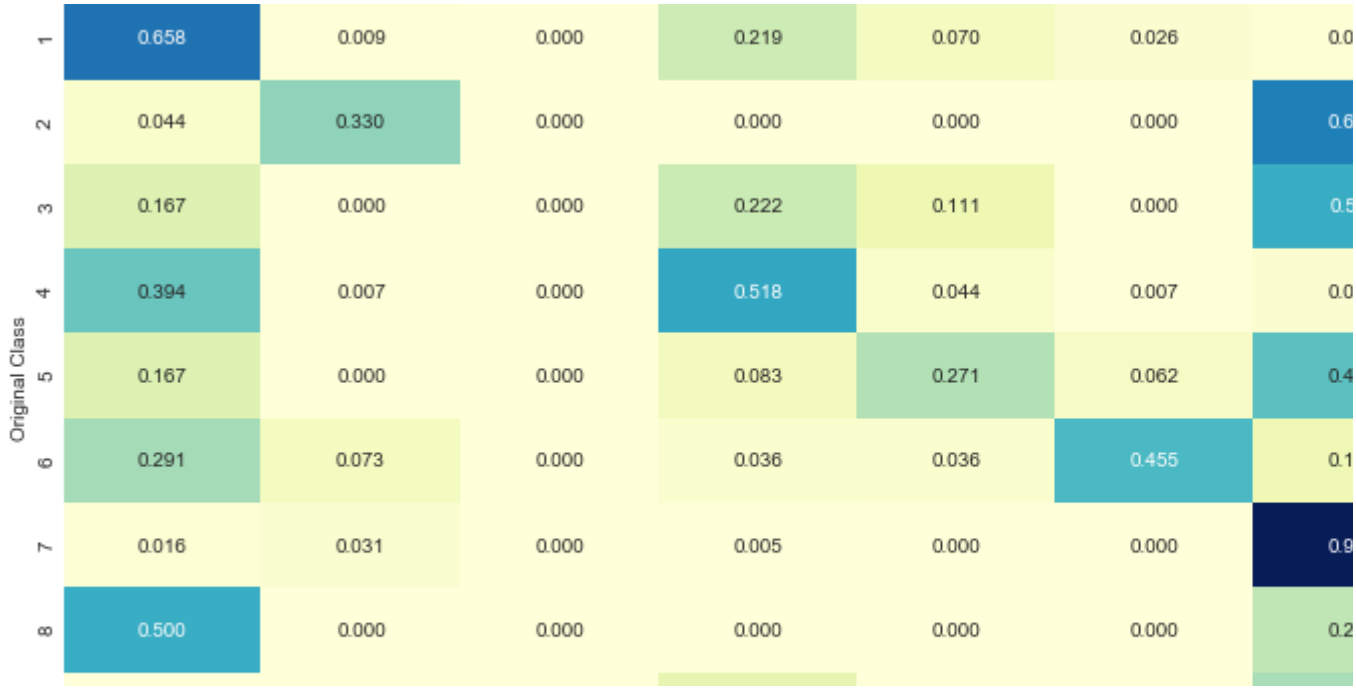
```
Log loss (train) on the VotingClassifier : 0.8327739083507836
Log loss (CV) on the VotingClassifier : 1.2054821308961232
Log loss (test) on the VotingClassifier : 1.224545673345812
Number of missclassified point : 0.40150375939849625
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['maximum voting','tfidf',0.8327,1.2054,1.2245,0.4015]
results.loc[12] = new
```

Feature Engineering Techniques

Gene Feature

```
result = pd.merge(data, data_text,on='ID', how='left')
result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

x_train, x_test, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)

x_train, x_cv, y_train, y_cv = train_test_split(x_train, y_train, stratify=y_train, test_size=0.2)


def get_gv_fea_dict(alpha, feature, df):
    value_count = x_train[feature].value_counts()
```



```
        for k in range(1,10):
            cls_cnt = x_train.loc[(x_train['Class']==k) & (x_train[feature]==i)]
            vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
        gv_dict[i]=vec
    return gv_dict

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    value_count = x_train[feature].value_counts()
    gv_fea = []
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
```

```
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    return gv_fea


alpha = 1

# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_train))

# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_test))

# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", x_cv))


gene_vectorizer = TfidfVectorizer()
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(x_train['Gene'])
test_gene_feature_onehotCoding = gene_vectorizer.transform(x_test['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(x_cv['Gene'])
```

Variation Feature

```
alpha = 1

# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_train))

# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_test))

# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", x_cv))


variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(x_train['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(x_test['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(x_cv['Variation'])
```

Text Feature

```
def extract_dictionary_paddle(cls_text):
            dictionary[word] +=1
    return dictionary


import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
                sum_prob += math.log(((dict_list[i].get(word,0)+10 )/(total_dict.get(word,0)+90)))
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
```

```
            row_index += 1
    return text_feature_responseCoding
```

```
text_vectorizer = TfidfVectorizer()
train_text_feature_onehotCoding = text_vectorizer.fit_transform(x_train['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

# train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of feature
train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1

# zip(list(text_features),text_fea_counts) will zip a word with its number of times it occured
text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))


print("Total number of unique words in train data :", len(train_text_features))
```

```
Total number of unique words in train data : 128536
```

```
dict_list = []
# dict_list =[] contains 9 dictoinaries each corresponds to a class
for i in range(1,10):
    cls_text = x_train[x_train['Class']==i]
    # build a word dict based on the words in that class
    dict_list.append(extract_dictionary_paddle(cls_text))
    # append it to dict_list

# dict_list[i] is build on i'th  class text data
# total_dict is buid on whole training text data
total_dict = extract_dictionary_paddle(x_train)


confuse_array = []
for i in train_text_features:
    ratios = []
    max_val = -1
    for j in range(0,9):
        ratios.append((dict_list[j][i]+10 )/(total_dict[i]+90))
    confuse_array.append(ratios)
confuse_array = np.array(confuse_array)
```

```
train_text_feature_responseCoding  = get_text_responsecoding(x_train)
test_text_feature_responseCoding  = get_text_responsecoding(x_test)
cv_text_feature_responseCoding  = get_text_responsecoding(x_cv)

# https://stackoverflow.com/a/16202486
# we convert each row values such that they sum to 1
```

Automatic document saving has been pending for 3 minutes. Reloading may fix the problem. Save and reload the page.

```
test_text_feature_onehotCoding = text_vectorizer.transform(x_test['TEXT'])
cv_text_feature_onehotCoding = text_vectorizer.transform(x_cv['TEXT'])
```

Features after feature engineering

```
gene_variation = []
```

```
for gene in data['Gene'].values:
    gene_variation.append(gene)

for variation in data['Variation'].values:
    gene_variation.append(variation)


tfidfVectorizer = TfidfVectorizer(max_features=1000)
text2 = tfidfVectorizer.fit_transform(gene_variation)
gene_variation_features = tfidfVectorizer.get_feature_names()

train_text = tfidfVectorizer.transform(x_train['TEXT'])
test_text = tfidfVectorizer.transform(x_test['TEXT'])
cv_text = tfidfVectorizer.transform(x_cv['TEXT'])
```

Stack above three features

```
train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,train_variation_feature_onehot
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCod
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))

# Adding the train_text feature
train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text))
train_x_onehotCoding = hstack((train_x_onehotCoding, train_text_feature_onehotCoding)).tocsr()
train_y = np.array(list(x_train['Class']))

# Adding the test_text feature
test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text))
test_x_onehotCoding = hstack((test_x_onehotCoding, test_text_feature_onehotCoding)).tocsr()
test_y = np.array(list(x_test['Class']))

# Adding the cv_text feature
cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text))
cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
cv_y = np.array(list(x_cv['Class']))


train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_variation_feature
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_feature_re
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_feature_response

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_responseCoding
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding))
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))


print("One hot encoding features :")
```



```
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.s
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 131724)
(number of data points * number of features) in test data =  (665, 131724)
(number of data points * number of features) in cross validation data = (532, 131724)
```

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_responseCoding
```

```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

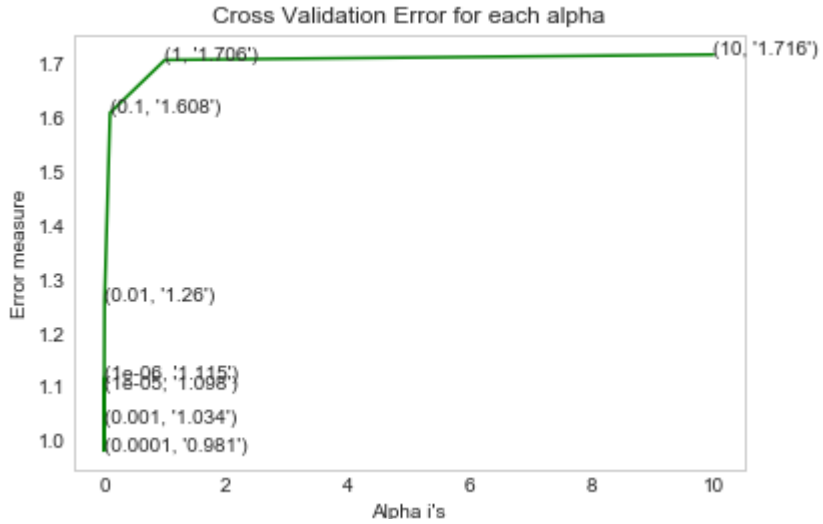
Logistic Regression

```
alpha = [10 ** x for x in range(-6, 2)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=41)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimates
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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],str(txt)), (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()


best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log',)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha],
      "The train log loss is:",
      log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))

predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha]
```



```
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ',
      alpha[best_alpha], "The test log loss is:",
      log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

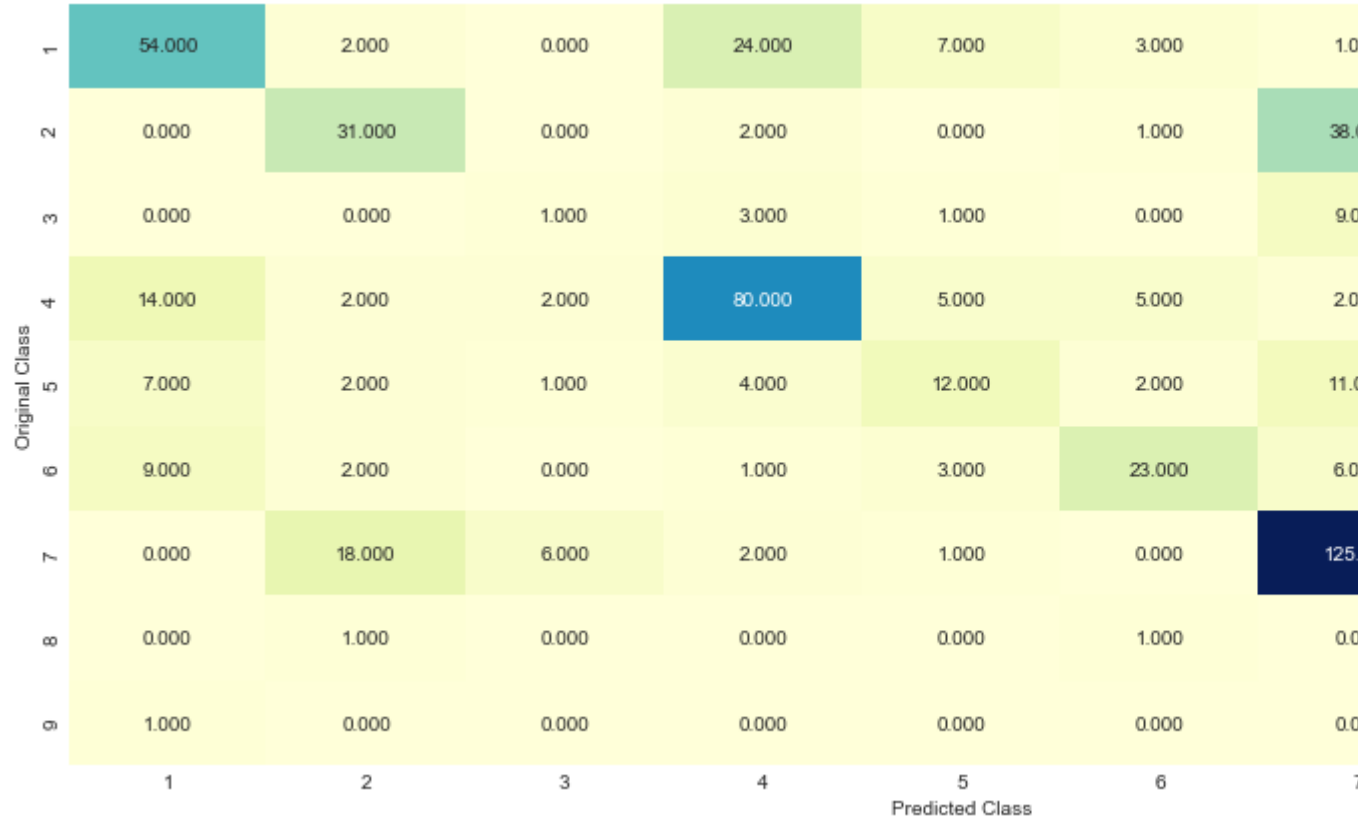
```
for alpha = 1e-06
Log Loss : 1.1153160770163346
for alpha = 1e-05
Log Loss : 1.0976673949227687
for alpha = 0.0001
Log Loss : 0.9814826194349467
for alpha = 0.001
Log Loss : 1.034154079464465
for alpha = 0.01
Log Loss : 1.2595523814367904
for alpha = 0.1
Log Loss : 1.6083947353339116
for alpha = 1
Log Loss : 1.7064330653847322
for alpha = 10
Log Loss : 1.7162952387646266
```



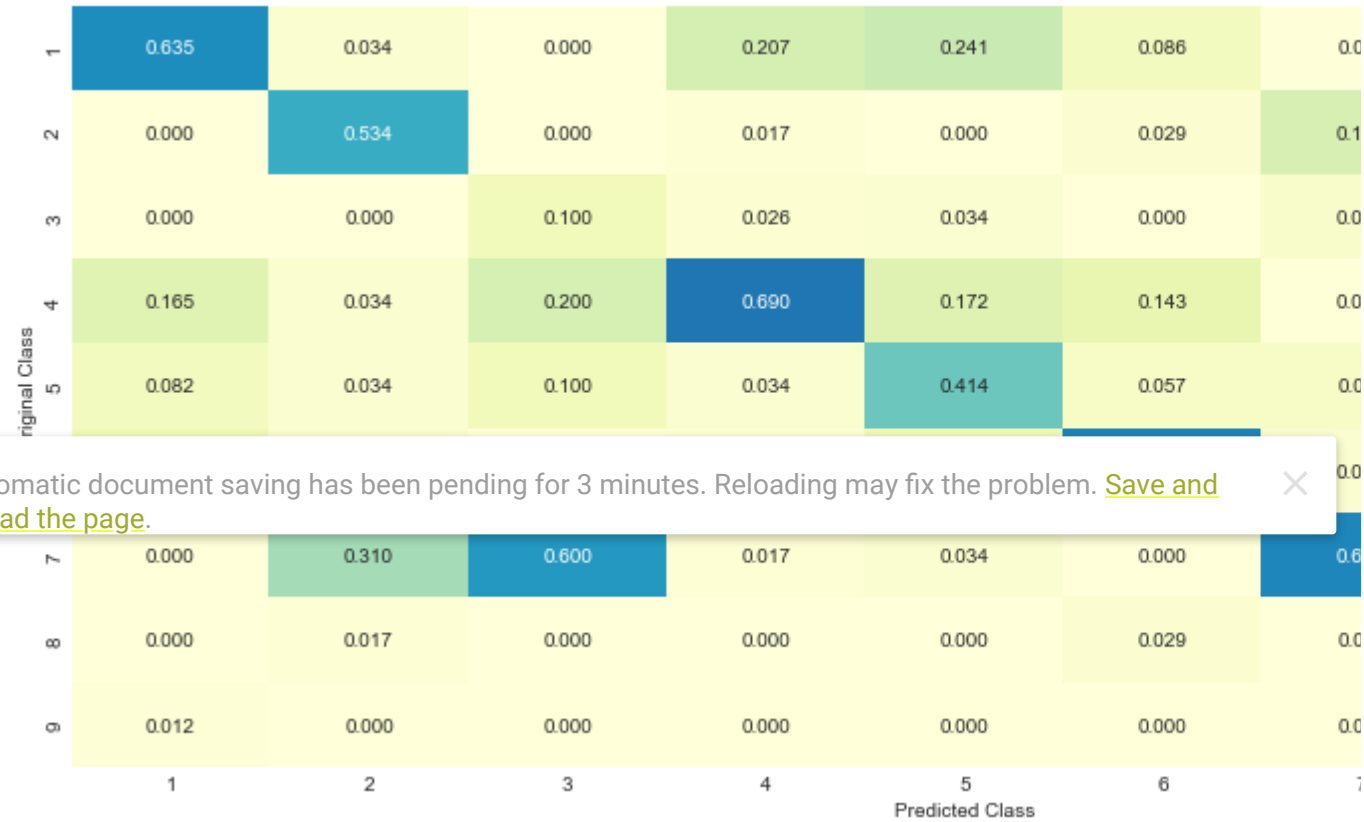

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log',)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)
```

Automatic document saving has been pending for 3 minutes. Reloading may fix the problem. Save and reload the page.

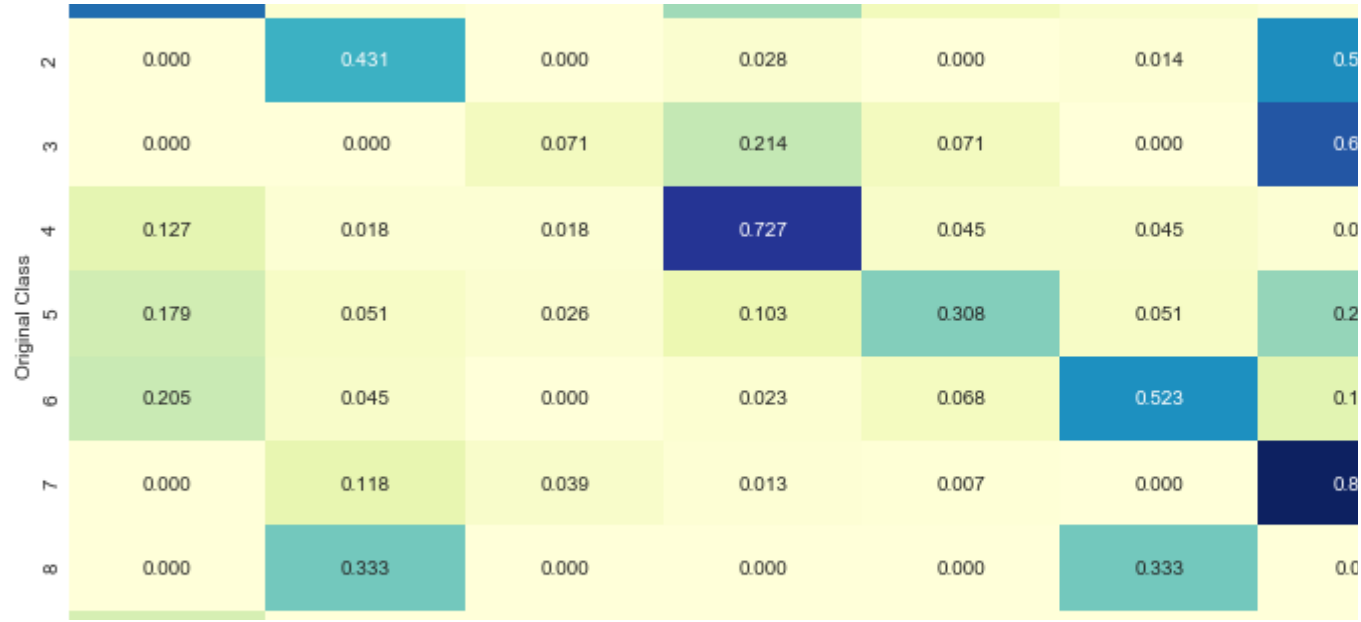
```
Log loss : 0.977353478401608
Number of mis-classified points : 0.37593984962406013
-------------------- Confusion matrix --------------------
```




```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




```
-------------------- Recall matrix (Row sum=1) --------------------
```




```
new = ['After Feature Engineering Logistic Regression','tfidf',0.4432,0.9773,0.9868,0.3759]
results.loc[13] = new
```

Performance Table

```
results
```

	Model	Featuraization	Train loss	CV loss	Test los
0	Naive Bayes	tfidf	0.5215	1.1689	1.200
1	KNN	tfidf	0.6612	1.0433	1.092
2	Lr- Regression with class balance	tfidf	0.4517	0.9991	1.363
3	Lr- Regression without class balance	tfidf	0.4438	1.0252	1.061
4	Lr- Regression with class balance	Unigrams	0.6338	1.1531	1.121
5	Lr- Regression without class balnce	Unigram	0.6351	1.1660	1.140
					32
					37
8	Linear SVM	tfidf	0.5088	1.0464	1.087
9	Random Forest with One hot encoding	tfidf	0.8597	1.2004	1.230
10	Random Forest with Response code	tfidf	0.0555	1.2759	1.328
11	staking	tfidf	0.5362	1.1722	1.195
12	maximum voting	tfidf	0.8327	1.2054	1.224
13	After Feature Engineering Logistic Regression	tfidf	0.4432	0.9773	0.986

Conclusion:

1. We build a Random model ofter we done Univariate on each feature
2. Stack the three types of features(gini,variation,text)
3. We build simple baseline model Naive Bayes with hyper parameter tuning and tfidf featuraization with top 10(
4. We build KNN on responsecoding with hyper parameter tuning. It gives log loss is :1.0433 and missclassified
5. We build Logistic Regression with calss balance and without class balance.With class balance and tfidf featu points = 35.15 % . With out class balance and tfidf featuraization log loss = 1.0252 and missclassified points =
6. Logistic Regression with class balnce and unigram featuraization log loss = 1.1531 and missclassified points featuraization log loss = 1.1660 and missclassified points = 37.96%
7. Logistic Regression with class balnce and bigram featuraization log loss = 1.2708 and missclassified points = featuraization log loss = 1.2687 and missclassified points = 40.78 %
8. We buid Linear SVM with tfidf featuraization and hyper perameter tuning log loss = 1.0464 and missclassified

10. We do staking with tfidf featuraization log loss = 1.1722 and missclassified points = 40.55 %.Stacking Model
11. Maximum voting with tfidf featuraization log loss = 1.2054 and missclassified points = 40.15 %
12. After applying some feature engineering techniques on logistic regression log loss and test loss reduce belov