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 literatu

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-v 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. Machine Learning Problem Formulation

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

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
```

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
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
```

```
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.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
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
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarnin
  warnings.warn(message, FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/externals/six.py:31: FutureWarning: Th
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:144: FutureWarnin
  warnings.warn(message, FutureWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
data = pd.read_csv('/content/drive/My Drive/Colab Notebooks/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 muta
Fields are

- **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

```
# note the seprator in this file
data_text =pd.read_csv("/content/drive/My Drive/Colab Notebooks/training_text",sep="\|\|",
print('Number of data points : ', data_text.shape[0])
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...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
# 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
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text : 26.294294999999998 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...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

```
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+', '_')

# split the data into test and train by maintaining same distribution of output varaible '
X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test
# split the train data into train and cross validation by maintaining same distribution of
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distrib

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

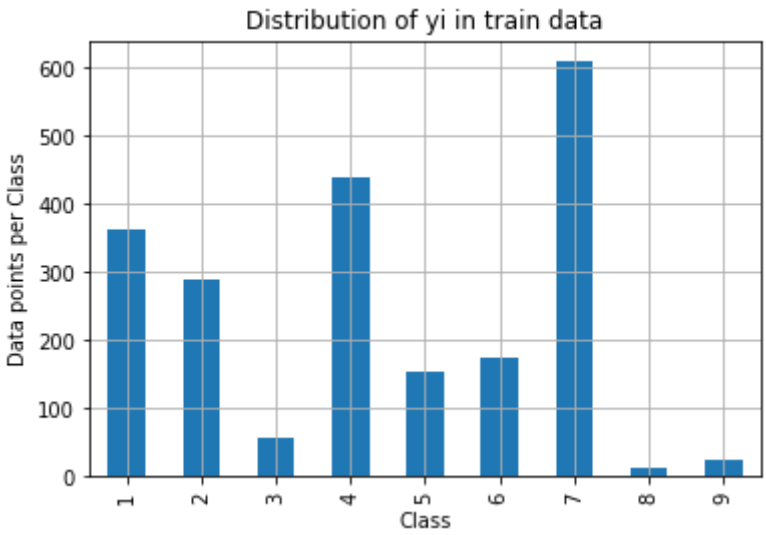
```
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

```
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
```

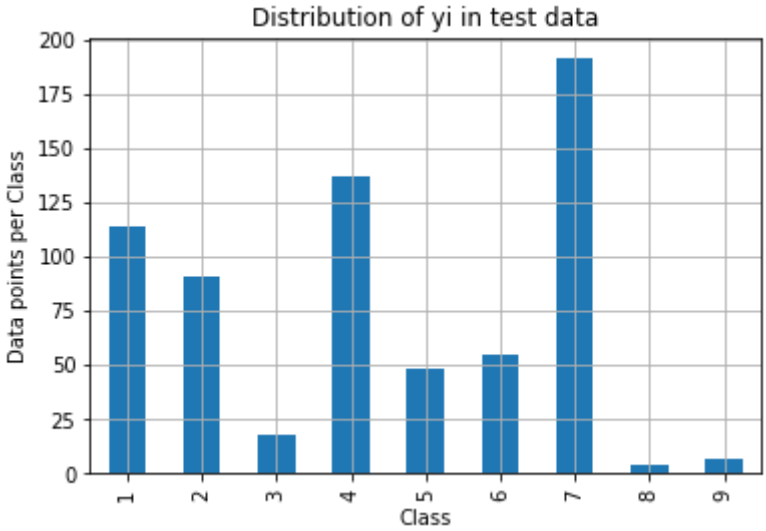
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
# it returns a dict, keys as class labels and values as the number of data points in that
train_class_distribution = train_df['Class'].value_counts().sort_index()
test_class_distribution = test_df['Class'].value_counts().sort_index()
cv_class_distribution = cv_df['Class'].value_counts().sort_index()

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], '


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], '(

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
```

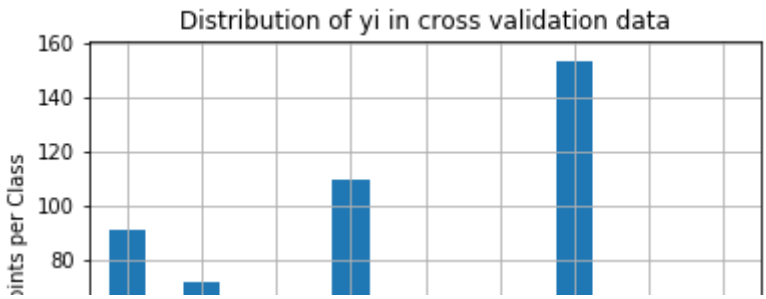
```
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':',cv_class_distribution.values[i], '(',
```

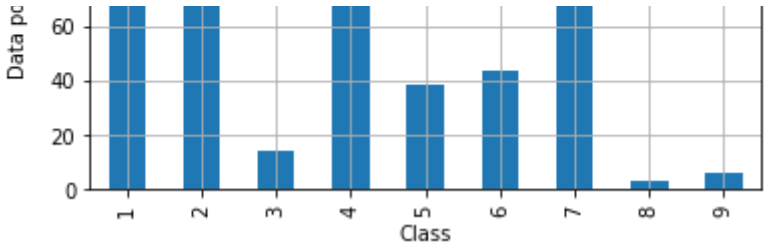



```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298 %)
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 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
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 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

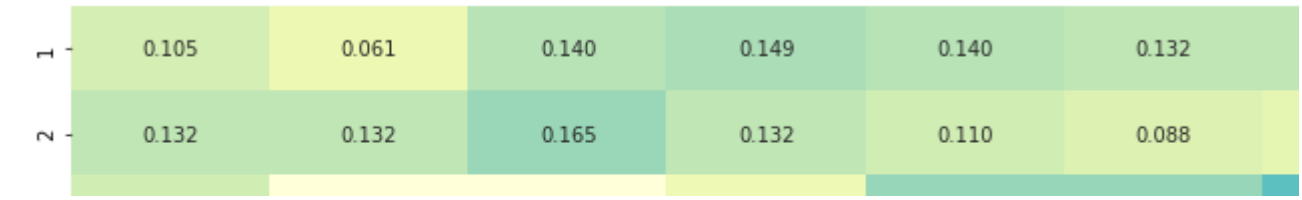
```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

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

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

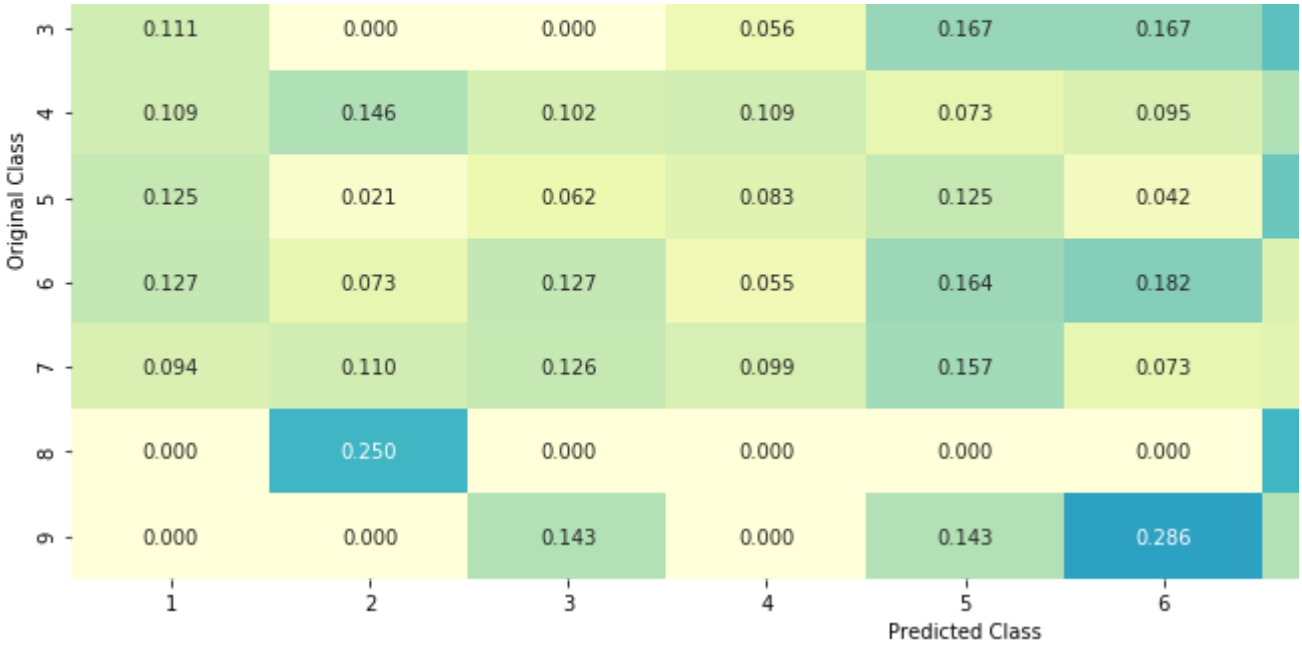
```
    # representing A in heatmap format
    print("-"*20, "Confusion matrix", "-"*20)
    plt.figure(figsize=(20,7))
    sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=l
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.show()

    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=l
    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=l
    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,


# 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-

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

```
Log loss on Cross Validation Data using Random Model 2.4266785057117146
Log loss on Test Data using Random Model 2.455515413264332
-------------------- 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 d
# build a vector (1*9) , the first element = (number of times it occured in class1 + 10*al
# 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
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         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/va
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occured in whole
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to perticula
        # 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
            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, 0.136
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.270
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181817
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.07
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.46
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.0728
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.073
```

```
    #      ...
    #     }
    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 i
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to 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])
#             gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea
```

when we caculate the probability of a feature belongs to any particular class, we apply laplace sm

- (numerator + 10*alpha) / (denominator + 90*alpha)

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))
```

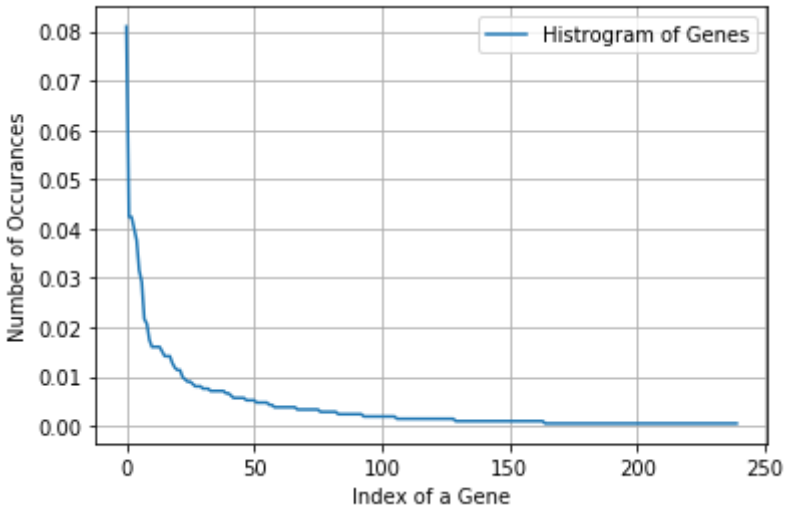
```
Number of Unique Genes : 240
BRCA1     172
TP53       90
EGFR       90
PTEN       85
BRCA2      80
BRAF       67
KIT        62
ALK        46
ERBB2      44
PDGFRA     37
Name: Gene, dtype: int64
```

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

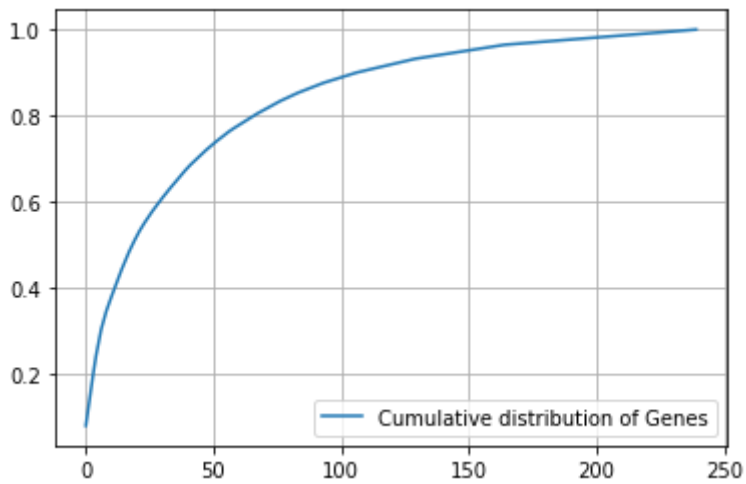
```
Ans: There are 240 different categories of genes in the train data, and they are dist
```

```
s = sum(unique_genes.values);
h = unique_genes.values/s;
```

```
h = unique_genes.values/5;
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.appliedai
online/lessons/handling-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 r
features, one-hot encoding is better for Logistic regression while response coding is better for Ra

```
#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.
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. T
```

```
# tfidf encoding of Gene feature.
from sklearn.feature_extraction.text import TfidfVectorizer
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()
```

```
2823     BRCA2
3284       RET
1493     FGFR2
838       ABL1
3094    NOTCH1
Name: Gene, dtype: object
```

```
gene_vectorizer.get_feature_names()
```

```
['abl1',
 'acvr1',
 'ago2',
 'akt1',
 'akt2',
 'akt3',
 'alk',
 'apc',
 'ar',
 'araf',
 'arid1b',
 'arid2',
 'arid5b',
 'asxl1',
 'asxl2',
 'atm',
 'atr',
 'atrx',
 'aurka',
 'aurkb',
 'axin1',
 'axl',
 'b2m',
 'bap1',
 'bcl10',
 'bcl2l11',
 'bcor',
 'braf',
 'brca1',
 'brca2',
 'brd4',
 'brip1',
 'btk',
 'card11',
 'carm1',
 'casp8',
 'cbl',
 'ccnd1',
 'ccnd3',
 'ccne1',
 'cdh1',
 'cdk12',
 'cdk4',
 'cdk6',
 'cdk8',
 'cdkn1a',
 'cdkn1b',
 'cdkn2a',
 'cdkn2b',
 'cdkn2c',
 'cebpa',
 'chek2',
 'cic',
 'crebbp',
 'ctcf',
 'ctla4',
 'ctnnb1',
 'ddr2',
 'dicer1',
 'dnmt3b',
 'dusp4',
```

```
         'egfr',
         'eif1ax',
         'elf3',
         'ep300',
         'epas1',
         'epcam',
         'erbb2',
         'erbb3',
         'erbb4',
         'ercc2',
         'ercc3',
         'ercc4',
         'erg',
         'errfi1',
         'esr1',
         'etv1',
         'etv6',
         'ewsr1',
         'ezh2',
         'fam58a',
         'fanca',
         'fancc',
         'fat1',
         'fbxw7',
         'fgf3',
         'fgfr1',
         'fgfr2',
         'fgfr3',
         'fgfr4',
         'flt1',
         'flt3',
         'foxa1',
         'foxl2',
         'foxo1',
         'foxp1',
         'fubp1',
         'gata3',
         'gli1',
         'gnaq',
         'gnas',
         'h3f3a',
         'hist1h1c',
         'hla',
         'hnf1a',
         'hras',
         'idh1',
         'idh2',
         'ikzf1',
         'il7r',
         'jak1',
         'jak2',
         'kdm5a',
         'kdm5c',
         'kdm6a',
         'kdr',
         'keap1',
         'kit',
         'klf4',
         'kmt2a',
         'kmt2b',
         'kmt2c',
         'kmt2d',
```

```
 'knstrn',
 'kras',
 'lats1',
 'map2k1',
 'map2k2',
 'map2k4',
 'map3k1',
 'med12',
 'mef2b',
 'met',
 'mga',
 'mlh1',
 'mpl',
 'msh2',
 'msh6',
 'mtor',
 'myc',
 'mycn',
 'myd88',
 'myod1',
 'ncor1',
 'nf1',
 'nf2',
 'nfe2l2',
 'nfkbia',
 'nkx2',
 'notch1',
 'notch2',
 'npm1',
 'nras',
 'nsd1',
 'ntrk1',
 'ntrk2',
 'ntrk3',
 'nup93',
 'pak1',
 'pax8',
 'pbrm1',
 'pdgfra',
 'pdgfrb',
 'pik3ca',
 'pik3cb',
 'pik3cd',
 'pik3r1',
 'pik3r2',
 'pik3r3',
 'pim1',
 'pms1',
 'pms2',
 'pole',
 'ppm1d',
 'ppp2r1a',
 'ppp6c',
 'prdm1',
 'ptch1',
 'pten',
 'ptpn11',
 'ptprd',
 'ptprt',
 'rab35',
 'rac1',
```

```
 'rad21',
 'rad50',
 'rad51b',
 'rad51c',
 'rad51d',
 'raf1',
 'rasa1',
 'rb1',
 'rbm10',
 'ret',
 'rheb',
 'rhoa',
 'rictor',
 'rit1',
 'rnf43',
 'ros1',
 'runx1',
 'rxra',
 'rybp',
 'sdhb',
 'sdhc',
 'setd2',
 'sf3b1',
 'shoc2',
 'shq1',
 'smad2',
 'smad3',
 'smad4',
 'smarca4',
 'smarcb1',
 'smo',
 'sos1',
 'sox9',
 'spop',
 'src',
 'stag2',
 'stat3',
 'stk11',
 'tcf3',
 'tert',
 'tet1',
 'tet2',
 'tgfbr1',
 'tgfbr2',
 'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

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

train gene feature onehotCoding is converted feature using one-hot encoding method. T

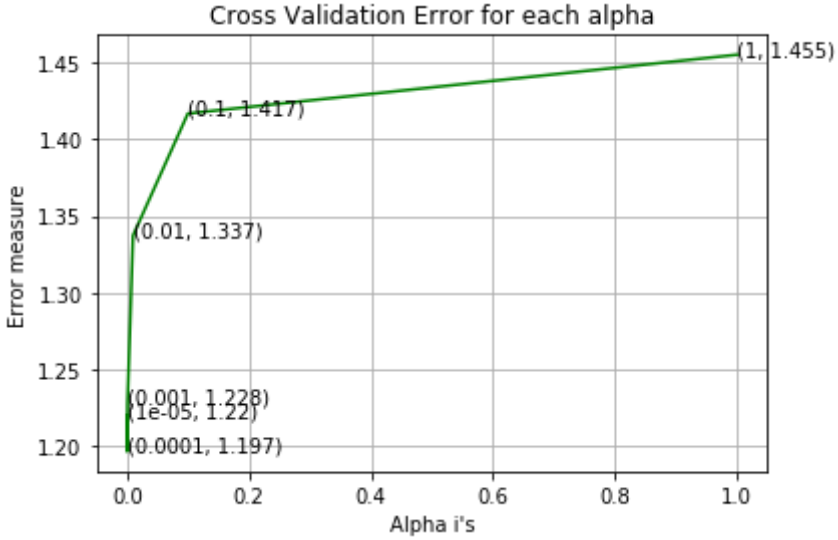
Q4. How good is this gene feature in predicting y_i?

There are many ways to estimate how good a feature is, in predicting y_i. One of the good method feature. In this case, we will build a logistic regression model using only Gene feature (one hot enc

```
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/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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

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_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(
```

```
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:
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
```

```
For values of alpha =  1e-05 The log loss is: 1.2196862811575042
For values of alpha =  0.0001 The log loss is: 1.196641798358647
For values of alpha =  0.001 The log loss is: 1.2280511960589267
For values of alpha =  0.01 The log loss is: 1.3372997135932572
For values of alpha =  0.1 The log loss is: 1.416848537161832
For values of alpha =  1 The log loss is: 1.4551437846180135
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9616445599819275
For values of best alpha =  0.0001 The cross validation log loss is: 1.19664179835864
For values of best alpha =  0.0001 The test log loss is: 1.2583283581324405
```

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

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/
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverag
```

```
Q6. How many data points in Test and CV datasets are covered by the  240  genes in tr
Ans
1. In test data 647 out of 665 : 97.29323308270676
2. In cross validation data 521 out of  532 : 97.93233082706767
```

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?

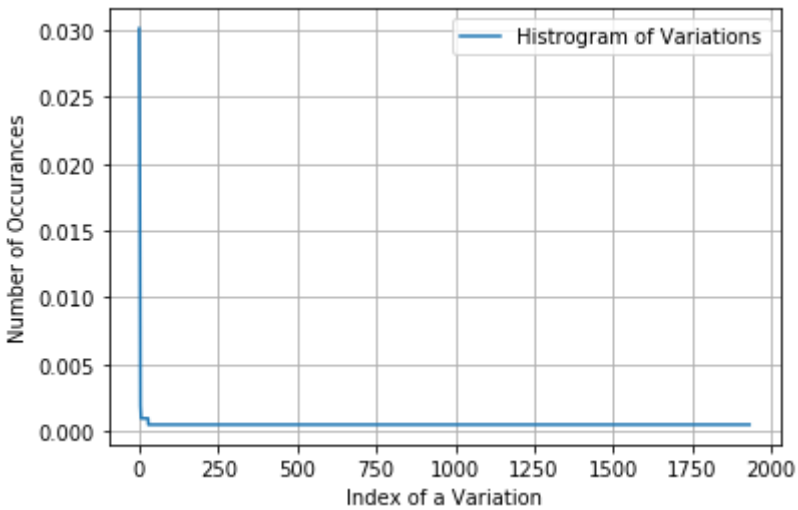
```
unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occured most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1932
Truncating_Mutations    64
Deletion                45
Amplification           38
Fusions                 21
G12V                     4
T58I                     3
Y42C                     2
M1R                      2
Q61R                     2
G67R                     2
Name: Variation, dtype: int64
```

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

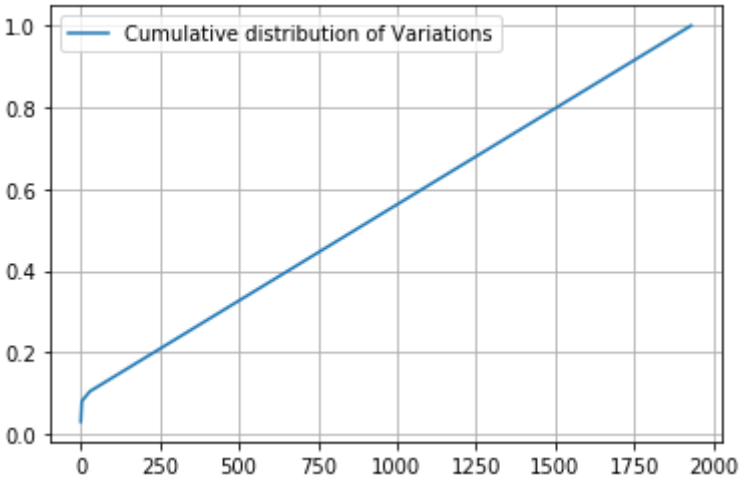
```
Ans: There are 1932 different categories of variations in the train data, and they ar
```

```
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()
```

```
[0.03013183 0.05131827 0.06920904 ... 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.appliedai online/lessons/handling-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
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_d
# 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 cc
```

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

```
# one-hot encoding of variation feature.
variation_vectorizer = TfidfVectorizer()
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variat
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

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

```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encodin
```

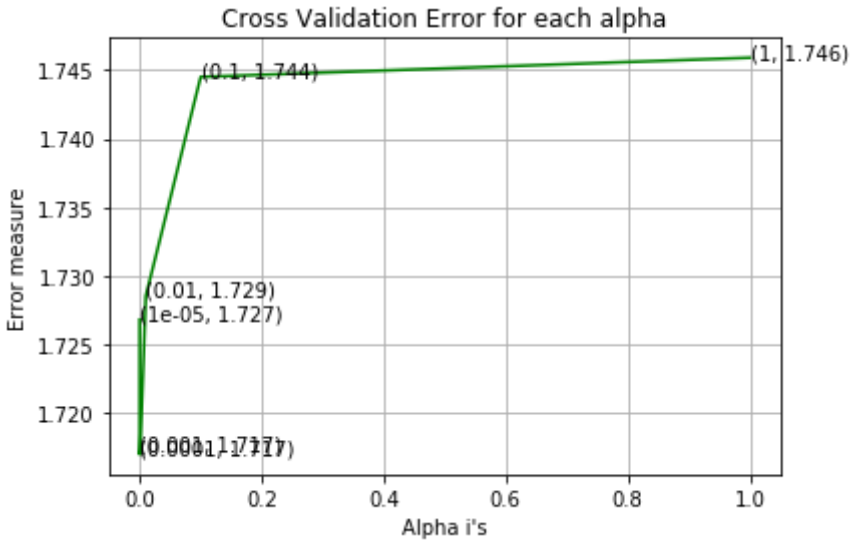
Q10. How good is this Variation feature in predicting y_i?

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/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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

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(
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:
```

```
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
```

```
For values of alpha =  1e-05 The log loss is: 1.7268113070391837
For values of alpha =  0.0001 The log loss is: 1.7169873046873894
For values of alpha =  0.001 The log loss is: 1.7173715562297662
For values of alpha =  0.01 The log loss is: 1.7285280214950953
For values of alpha =  0.1 The log loss is: 1.7444983395011235
For values of alpha =  1 The log loss is: 1.7458978788999457
```




```
For values of best alpha =  0.0001 The train log loss is: 0.666138849940641
For values of best alpha =  0.0001 The cross validation log loss is: 1.71698730468738
For values of best alpha =  0.0001 The test log loss is: 1.7095908167434941
```

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], " ger
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/
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_coverag
```

```
Q12. How many data points are covered by total  1932  genes in test and cross validat
Ans
1. In test data 78 out of 665 : 11.729323308270677
2. In cross validation data 49 out of  532 : 9.210526315789473
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
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
```

```
# 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,
            text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].
            row_index += 1
    return text_feature_responseCoding


# building a CountVectorizer with all the words that occured minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3,max_features=1000)
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
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 occ
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)
```

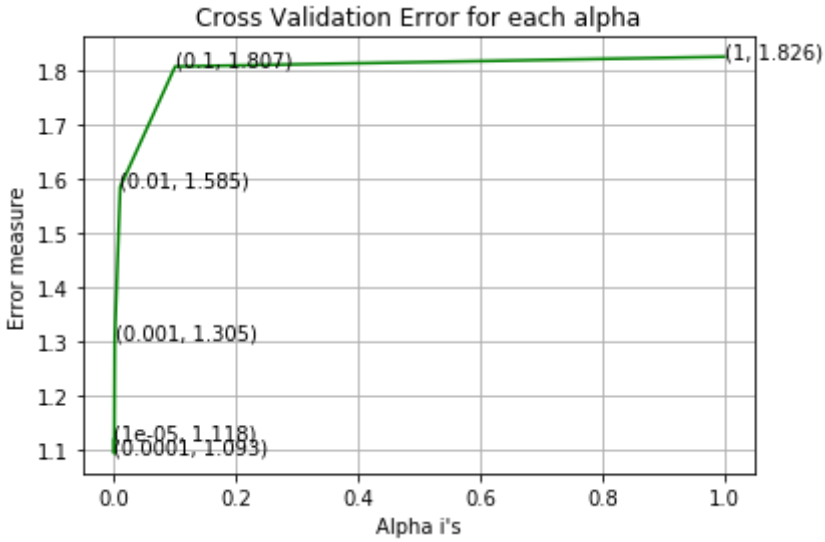
```
total_dict = extract_dictionary_paddle(train_df)


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_featur
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_r
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_respons


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

# 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=Tru
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))


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

```
Counter({3516: 3, 3242: 3, 2947: 3, 2851: 3, 2663: 3, 2646: 3, 9744: 2, 8276: 2, 8169
```

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
```

```
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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

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(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
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:
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
```

```
For values of alpha =  1e-05 The log loss is: 1.1180902361138372
For values of alpha =  0.0001 The log loss is: 1.092857045159537
For values of alpha =  0.001 The log loss is: 1.3046340776271625
For values of alpha =  0.01 The log loss is: 1.5850821003235362
For values of alpha =  0.1 The log loss is: 1.807413747288979
For values of alpha =  1 The log loss is: 1.825519846312339
```




```
For values of best alpha =  0.0001 The train log loss is: 0.9420862854495291
For values of best alpha =  0.0001 The cross validation log loss is: 1.09285704515953
For values of best alpha =  0.0001 The test log loss is: 1.2697506177586746
```

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

Ans. Yes, it seems like!

```
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"
```

```
3.504 % of word of test data appeared in train data
4.028 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
#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
    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.s
    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,y
        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(w
        else:
            word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
            yes_no = True if word in text.split() else False
```

```
            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,y

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

Stacking the three types of features

```
# 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_feat
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_feature
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehc

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

test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding))
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_variati
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variation_
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_featur

train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_resp
test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_respons
cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCodin



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

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 3202)
(number of data points * number of features) in test data =  (665, 3202)
(number of data points * number of features) in cross validation data = (532, 3202)
```

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCod
```

```
print("(number of data points * number of features) in test data = ", test_x_responseCodin
print("(number of data points * number of features) in cross validation data =", cv_x_resp
```

```
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

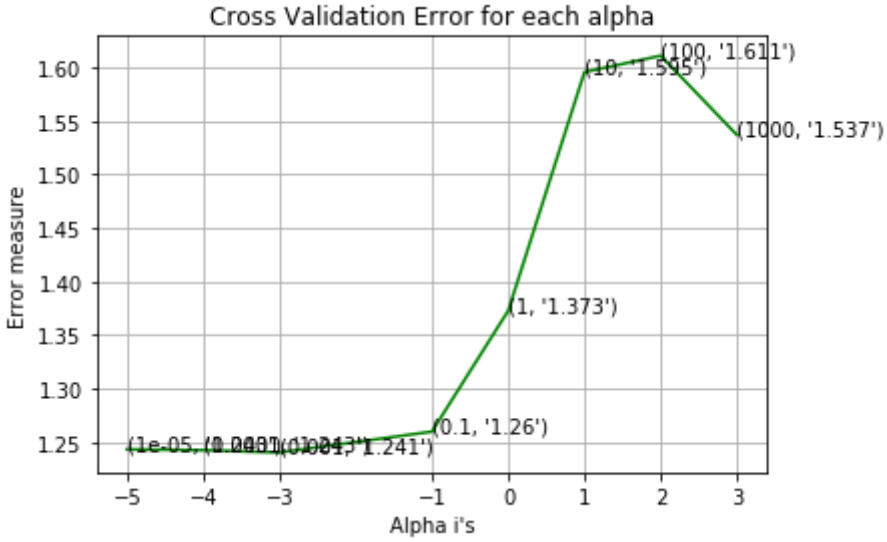
4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modu
# -------------------------
# 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/naiv
# -----------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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/naiv
# -----------------------


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.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 estima
    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(
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
for alpha = 1e-05
Log Loss : 1.2434369280653783
for alpha = 0.0001
Log Loss : 1.242646740693285
for alpha = 0.001
Log Loss : 1.2405020810469456
for alpha = 0.1
Log Loss : 1.259943941894815
for alpha = 1
Log Loss : 1.3727355285780591
for alpha = 10
Log Loss : 1.595450541683481
for alpha = 100
Log Loss : 1.6107506046092936
for alpha = 1000
Log Loss : 1.5368193367112561
```



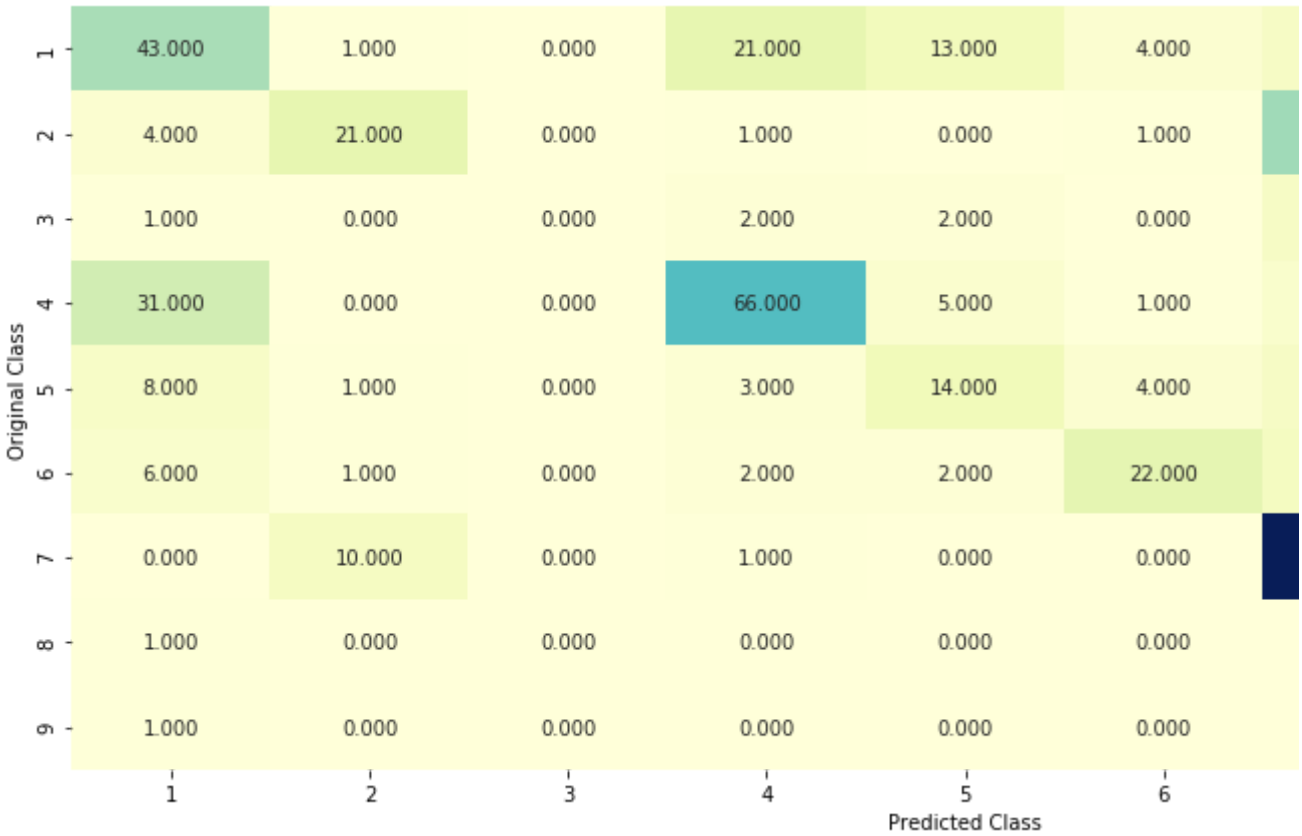

```
For values of best alpha =  0.001 The train log loss is: 0.4446663236298083
For values of best alpha =  0.001 The cross validation log loss is: 1.240502081046945
For values of best alpha =  0.001 The test log loss is: 1.2911328494660967
```

4.1.1.2. Testing the model with best hyper paramters

```
# find more about Multinomial Naive base function here http://scikit-learn.org/stable/modu
# -------------------------
# 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/naiv
# -----------------------


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

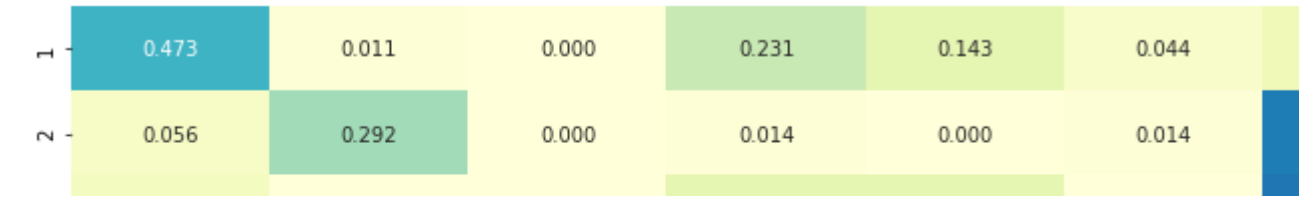
```
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])  Fit the calibrated model
# get_params([deep])  Get parameters for this estimator.
# predict(X)  Predict the target of new samples.
# 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_onehotCod
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

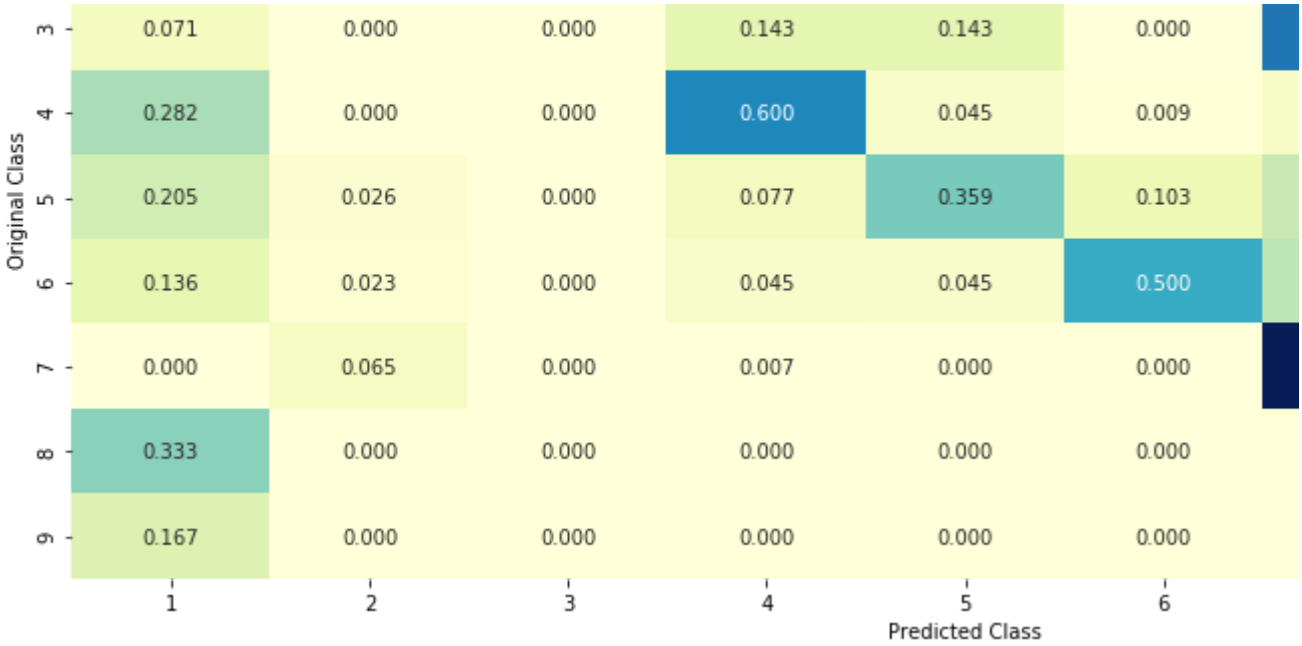
```
Log Loss : 1.2405020810469456
Number of missclassified point : 0.40977443609022557
-------------------- Confusion matrix --------------------
```




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


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





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
print("Actual Class :", test_y[test_point_index])
indices=np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0615 0.6733 0.0126 0.0708 0.0372 0.0278 0.1107 0.0
Actual Class : 2
--------------------------------------------------
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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.056  0.0837 0.0118 0.0645 0.0341 0.0252 0.7191 0.0
Actual Class : 2
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

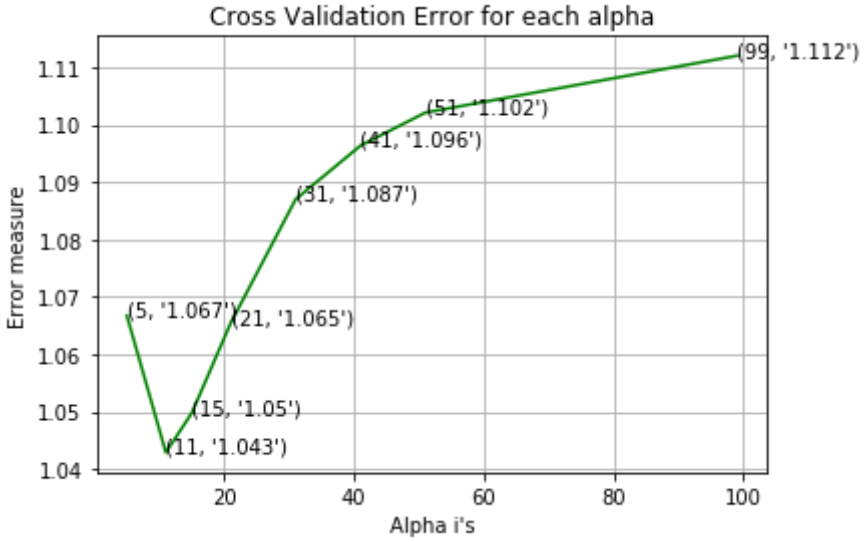
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

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

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


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

```
    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(
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
for alpha = 5
Log Loss : 1.0667280502334866
for alpha = 11
Log Loss : 1.042914009281993
for alpha = 15
Log Loss : 1.0497939647357155
for alpha = 21
Log Loss : 1.0654753018649226
for alpha = 31
Log Loss : 1.0869915420692449
for alpha = 41
Log Loss : 1.0963424982920769
for alpha = 51
Log Loss : 1.102095675318619
for alpha = 99
Log Loss : 1.1119909570876292
```

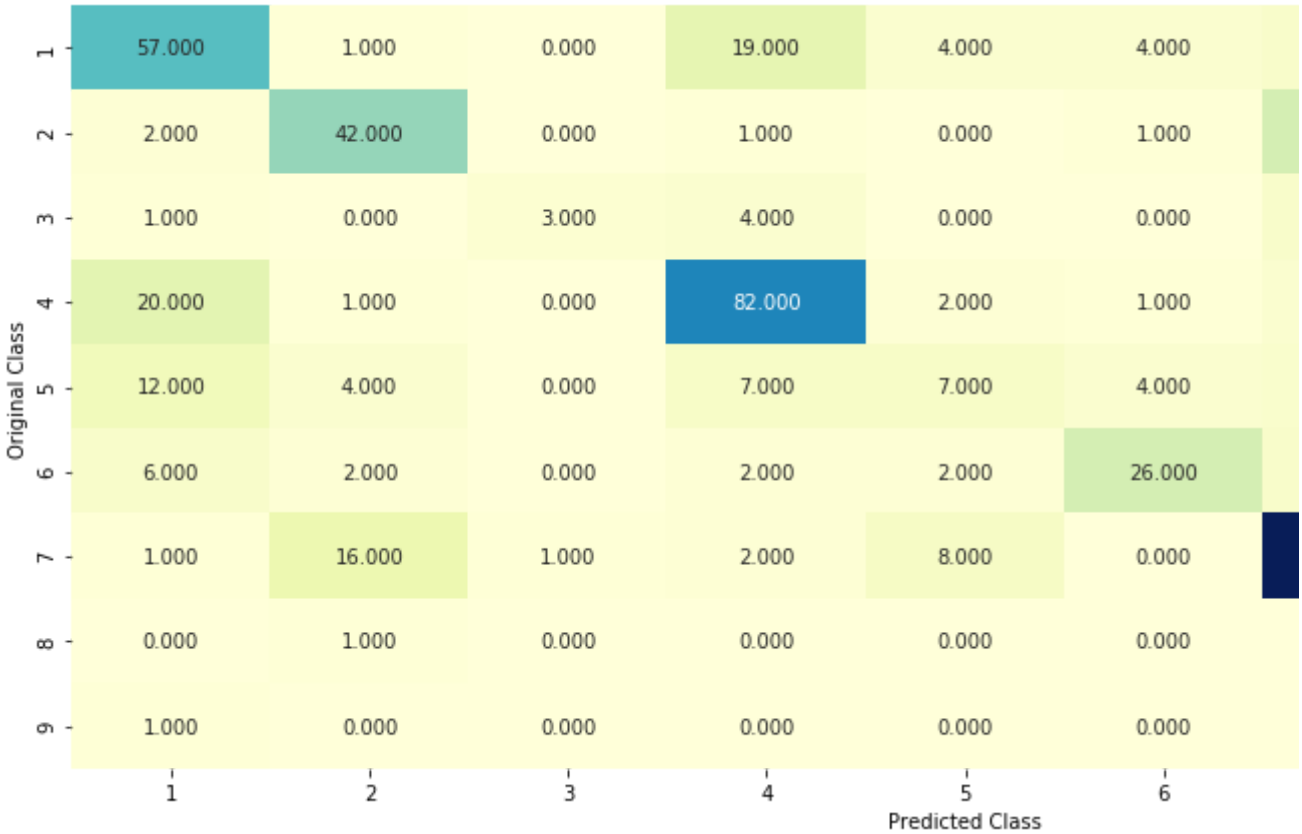



```
For values of best alpha =  11 The train log loss is: 0.5722617567923619
For values of best alpha =  11 The cross validation log loss is: 1.042914009281993
For values of best alpha =  11 The test log loss is: 1.167874383009327
```

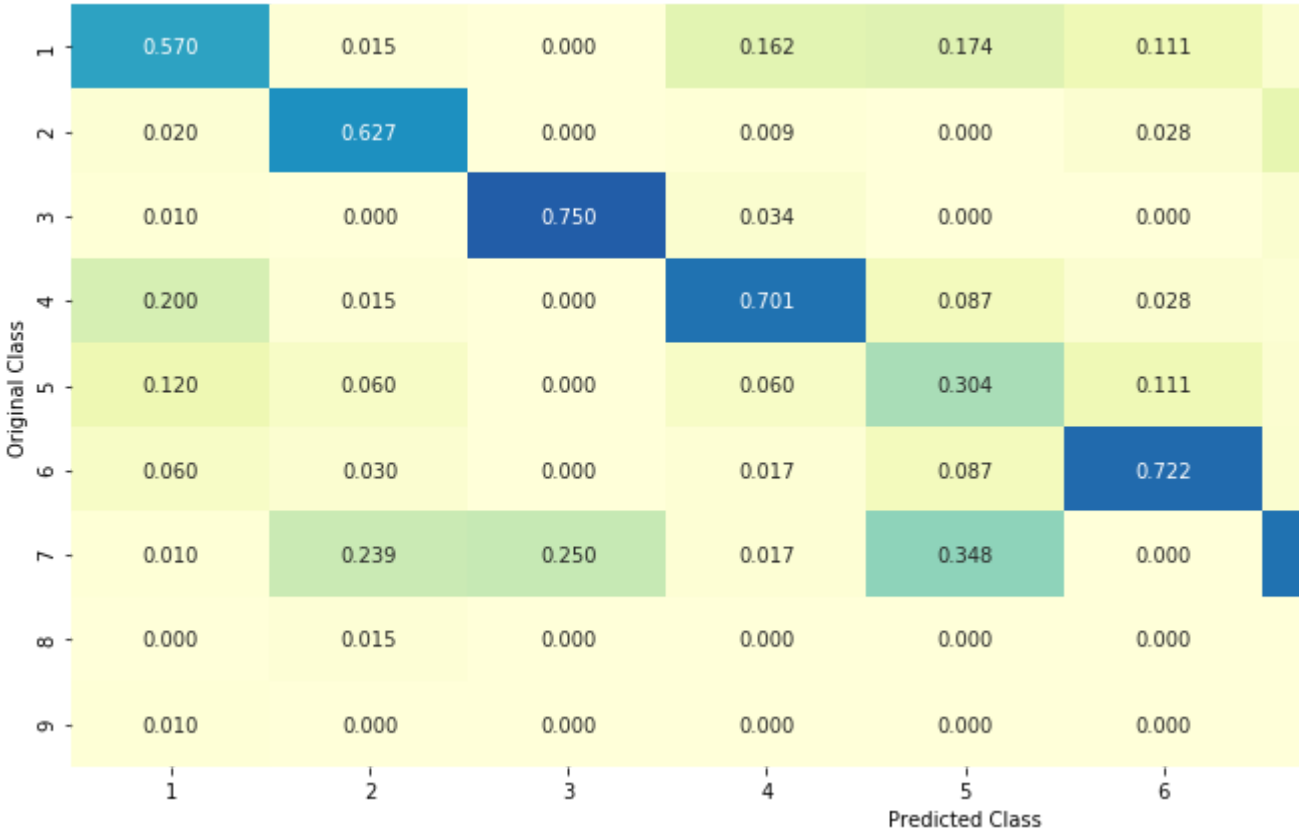
4.2.2. Testing the model with best hyper paramters

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

# methods of
# fit(X, y) : Fit the model using X as training data and y as target values
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-ne
#-------------------------------------
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv
```

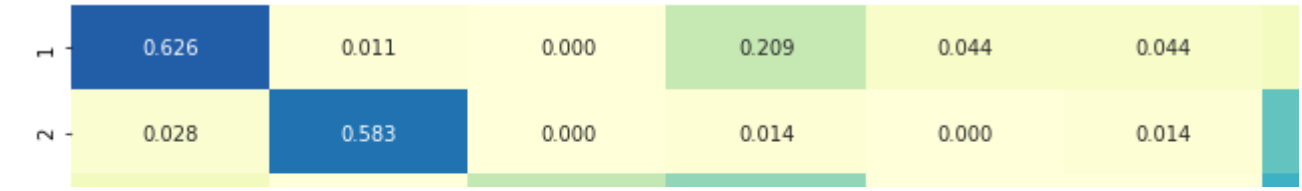
```
Log loss : 1.042914009281993
Number of mis-classified points : 0.3458646616541353
-------------------- Confusion matrix --------------------
```

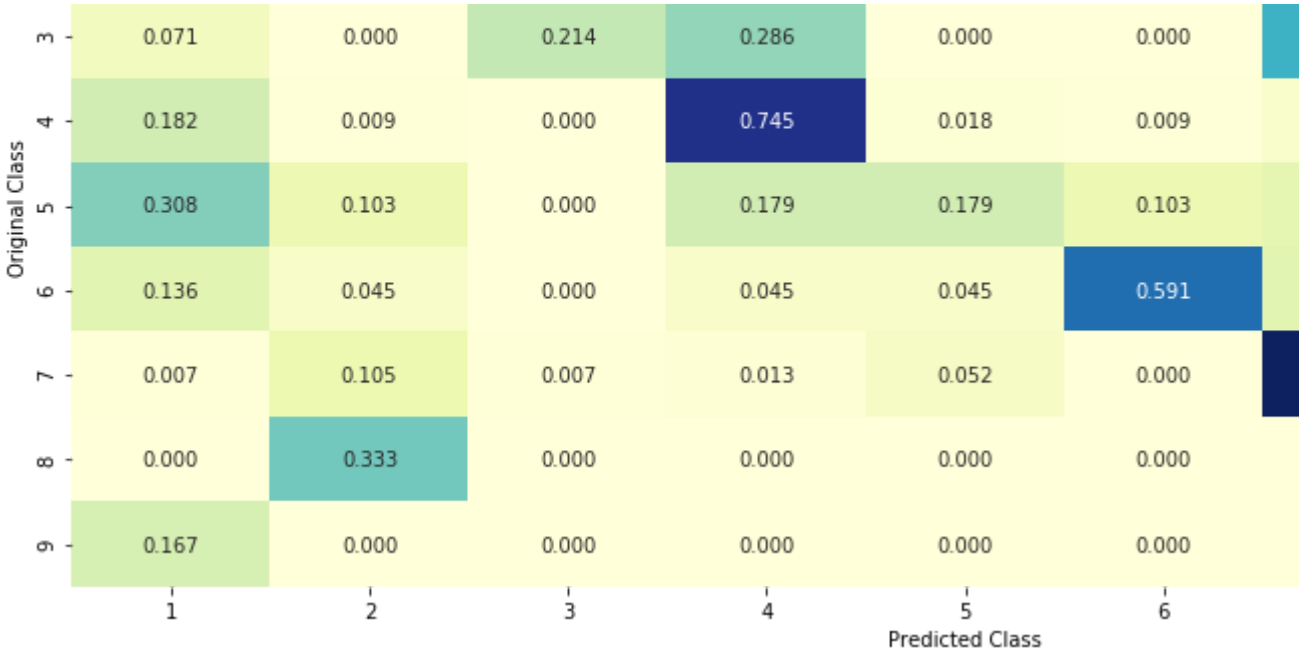



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





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[0].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[b
print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes"
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 2
Actual Class : 2
The  11  nearest neighbours of the test points belongs to classes [2 2 2 2 2 7 6 7 7
Fequency of nearest points : Counter({2: 7, 7: 3, 6: 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[b
```

```
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[b
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test p
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 2
the k value for knn is 11 and the nearest neighbours of the test points belongs to cl
Fequency of nearest points : Counter({7: 8, 2: 3})
```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

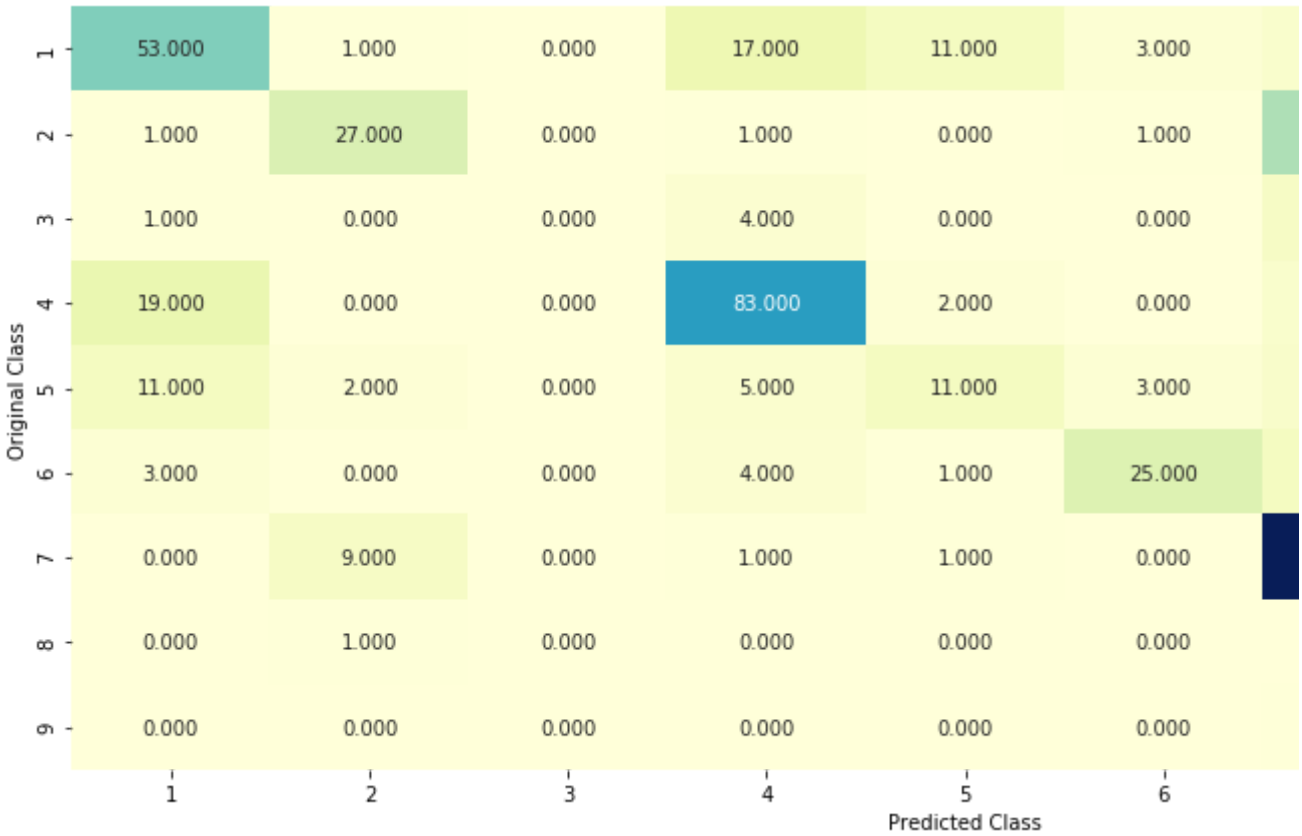
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random
    clf.fit(train_x_onehotCoding, train_y)
```

```
    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 estima
    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='
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(
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

4.3.1.2. Testing the model with best hyper paramters

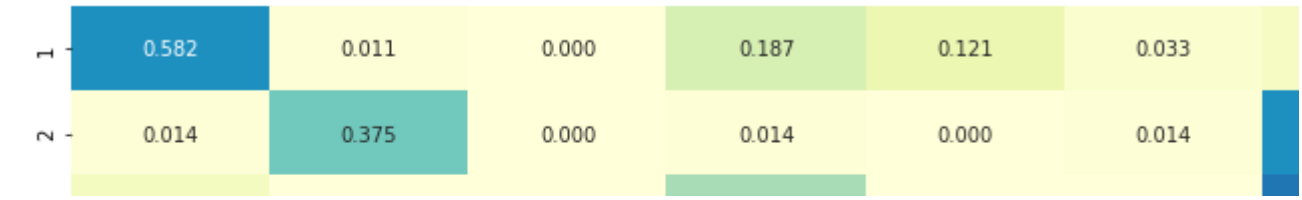
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None,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

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

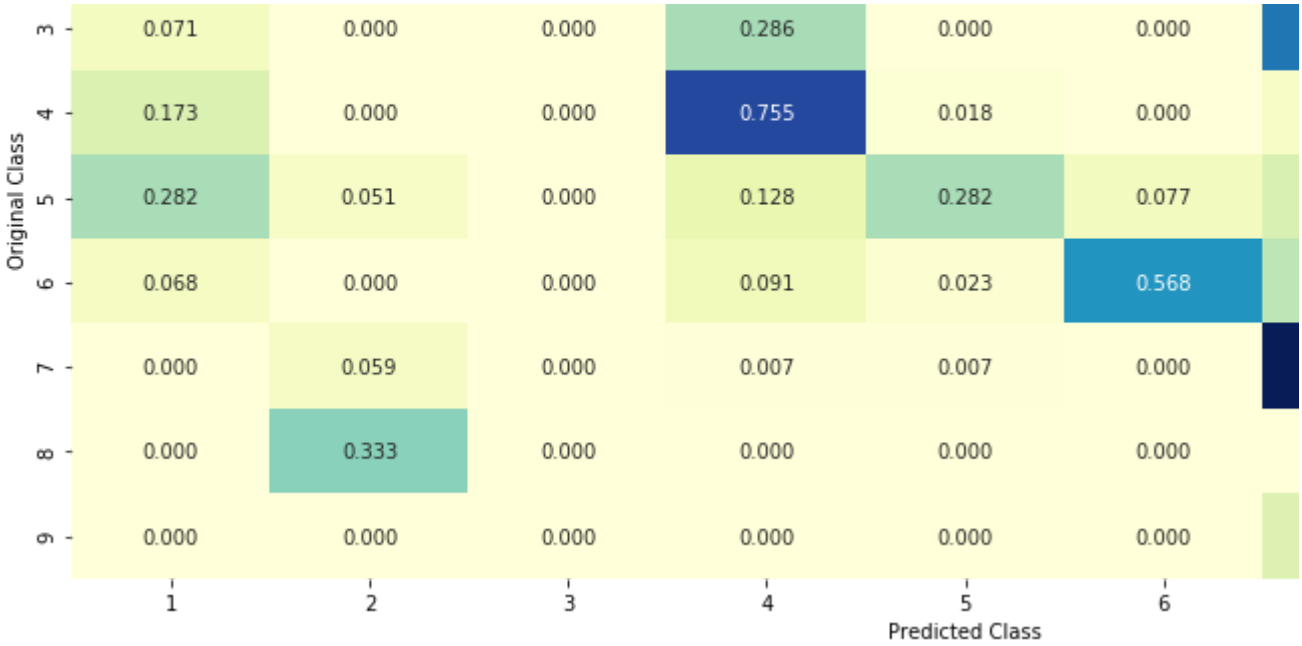
```
Log loss : 0.9949046441610732
Number of mis-classified points : 0.34774436090225563
-------------------- Confusion matrix --------------------
```




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


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





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='
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
```

```
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0733 0.767  0.016  0.0216 0.0217 0.0311 0.0549 0.0
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

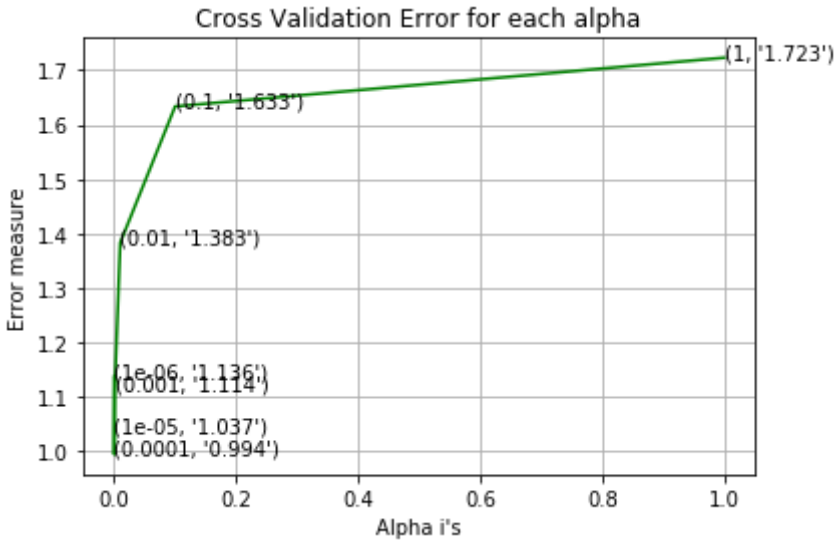
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

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

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


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

```
# ----------------------------
# 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(
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
for alpha = 1e-06
Log Loss : 1.136482140857177
for alpha = 1e-05
Log Loss : 1.0368376251084792
for alpha = 0.0001
Log Loss : 0.9938234479054255
for alpha = 0.001
Log Loss : 1.1143680745282043
for alpha = 0.01
Log Loss : 1.3829721405841202
for alpha = 0.1
Log Loss : 1.6334785650563497
for alpha = 1
Log Loss : 1.722956626558638
```

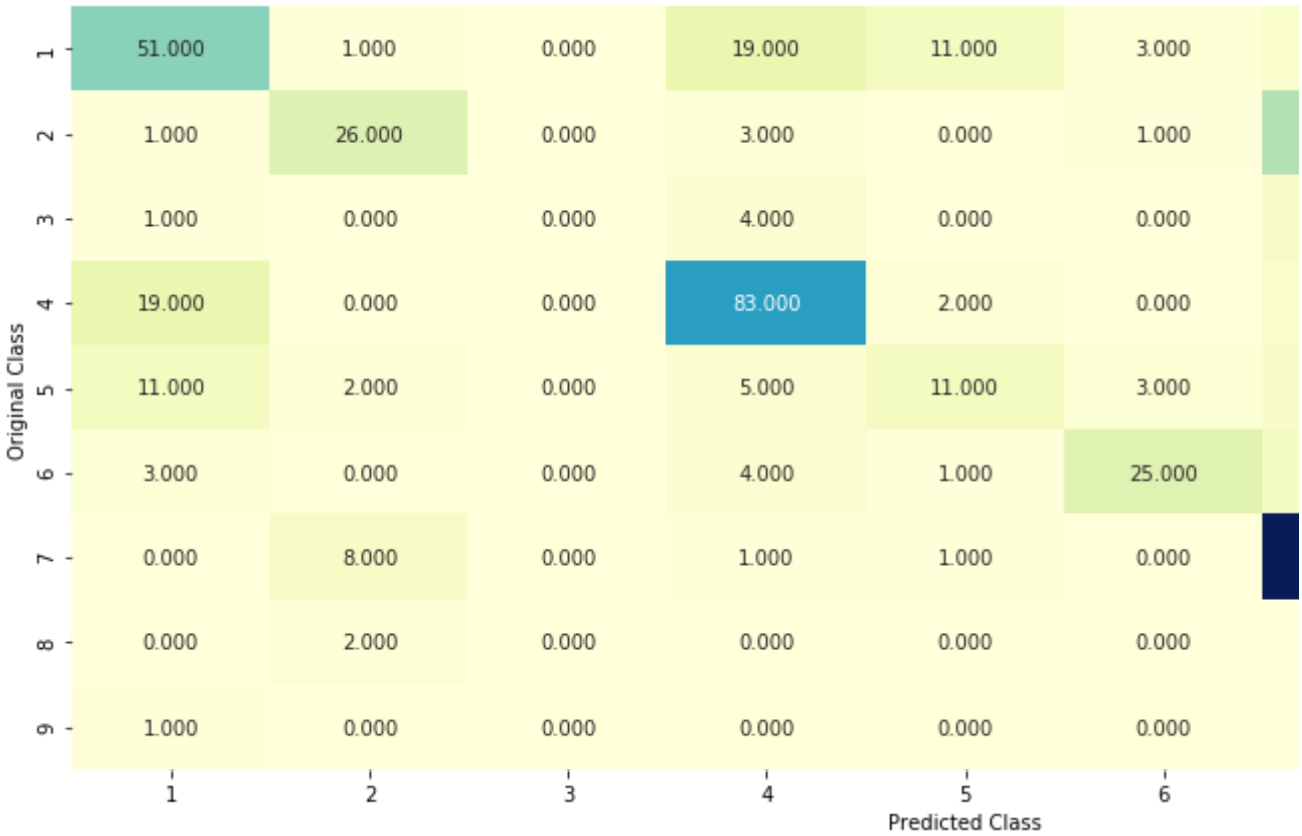



```
For values of best alpha =  0.0001 The train log loss is: 0.37658416351640145
For values of best alpha =  0.0001 The cross validation log loss is: 0.99382344790542
For values of best alpha =  0.0001 The test log loss is: 1.0852118785249474
```

4.3.2.2. Testing model with best hyper parameters

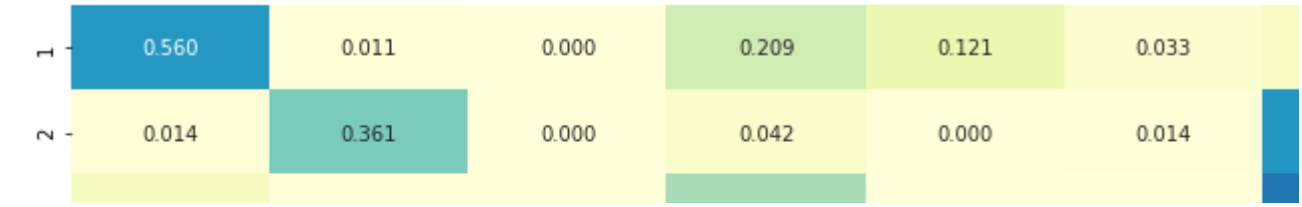
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# 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_onehotCoding, train_y, cv_x_onehotCoding, cv_y,
```

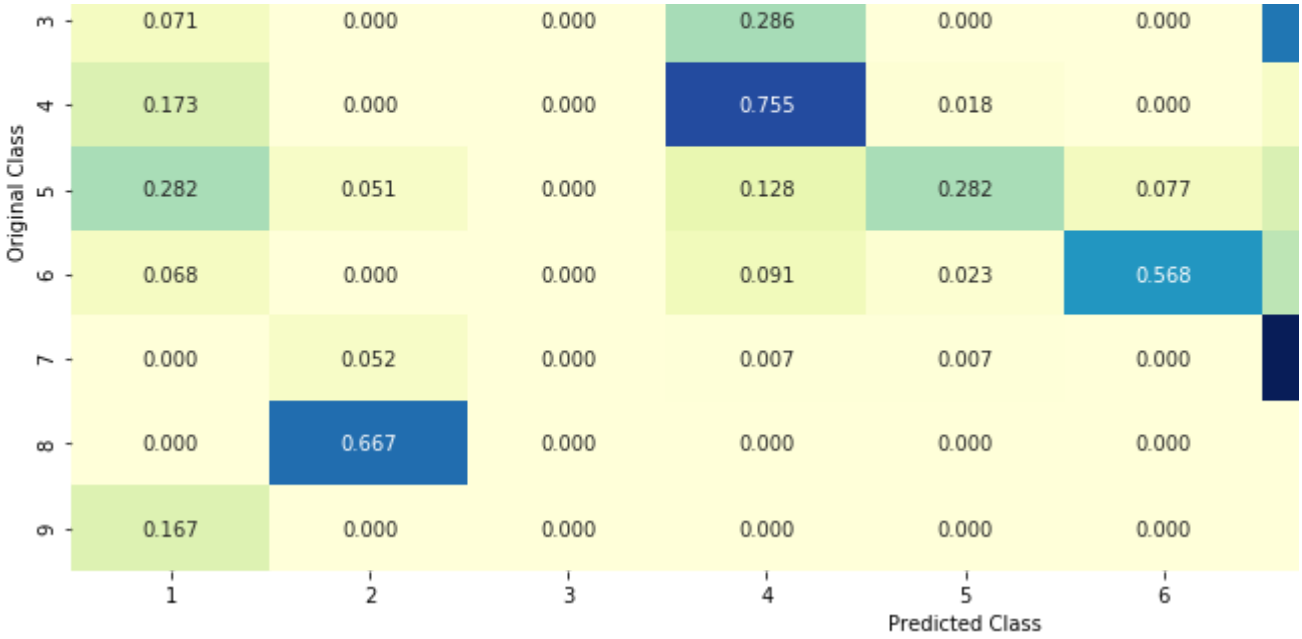
```
Log loss : 0.9938234479054255
Number of mis-classified points : 0.3533834586466165
-------------------- Confusion matrix --------------------
```




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


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





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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0699 0.7681 0.0158 0.0219 0.0228 0.0342 0.0519 0.0
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 are present in query point
```

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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0119 0.1755 0.0479 0.0108 0.0613 0.0049 0.6819 0.0
Actual Class : 2
--------------------------------------------------
92 Text feature [10] present in test data point [True]
Out of the top  500  features  1 are present in query point
```

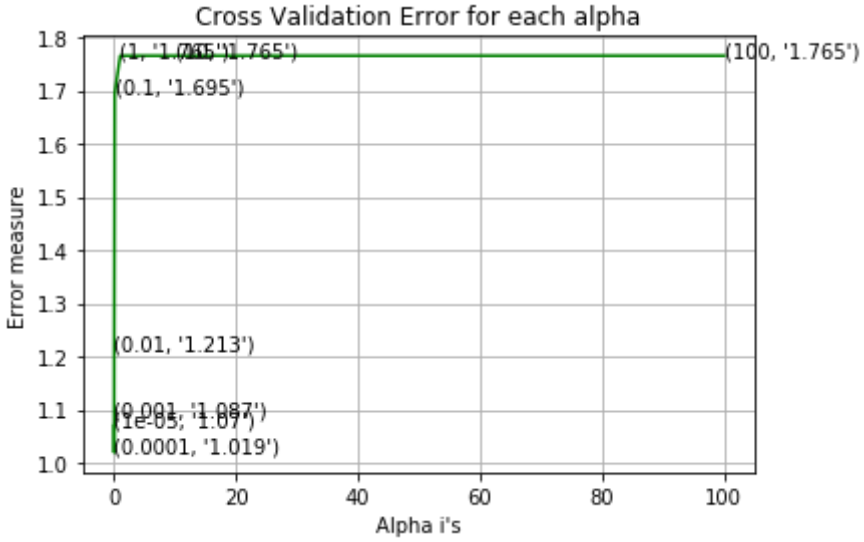
4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

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

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

# 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/math
# --------------------------------



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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', ran
    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='
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(
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:
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
```

```
for C = 1e-05
Log Loss : 1.0698822969097057
for C = 0.0001
Log Loss : 1.0193858139473322
for C = 0.001
Log Loss : 1.0872484058054963
for C = 0.01
Log Loss : 1.2134336253066649
for C = 0.1
Log Loss : 1.6951515535703288
for C = 1
Log Loss : 1.7651671192013214
for C = 10
Log Loss : 1.765165253959348
for C = 100
Log Loss : 1.765166614339717
```



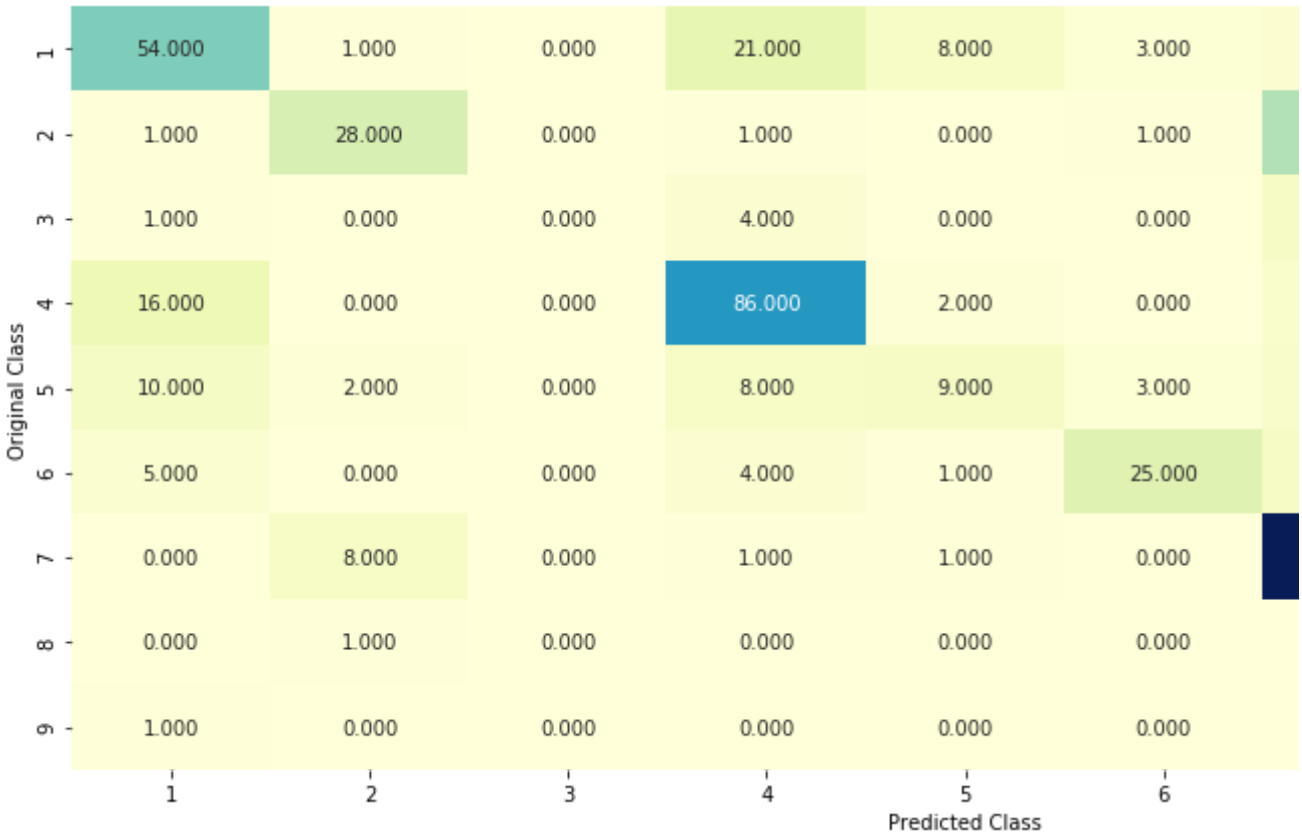

```
For values of best alpha =  0.0001 The train log loss is: 0.3063108332430362
For values of best alpha =  0.0001 The cross validation log loss is: 1.01938581394733
For values of best alpha =  0.0001 The test log loss is: 1.1181703217284822
```

4.4.2. Testing model with best hyper parameters

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

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

# 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/math
# --------------------------------


# 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,c
```

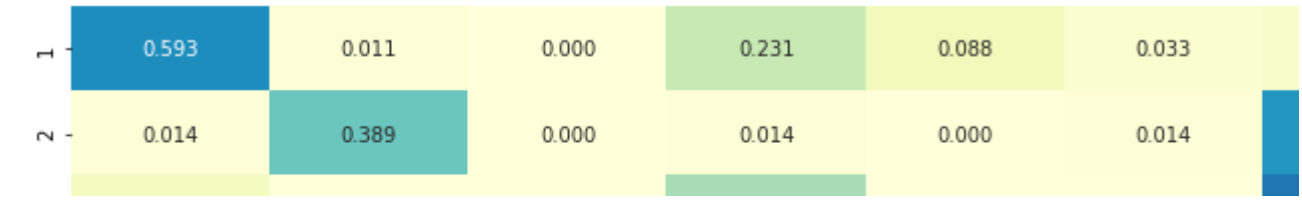
```
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, cl
```

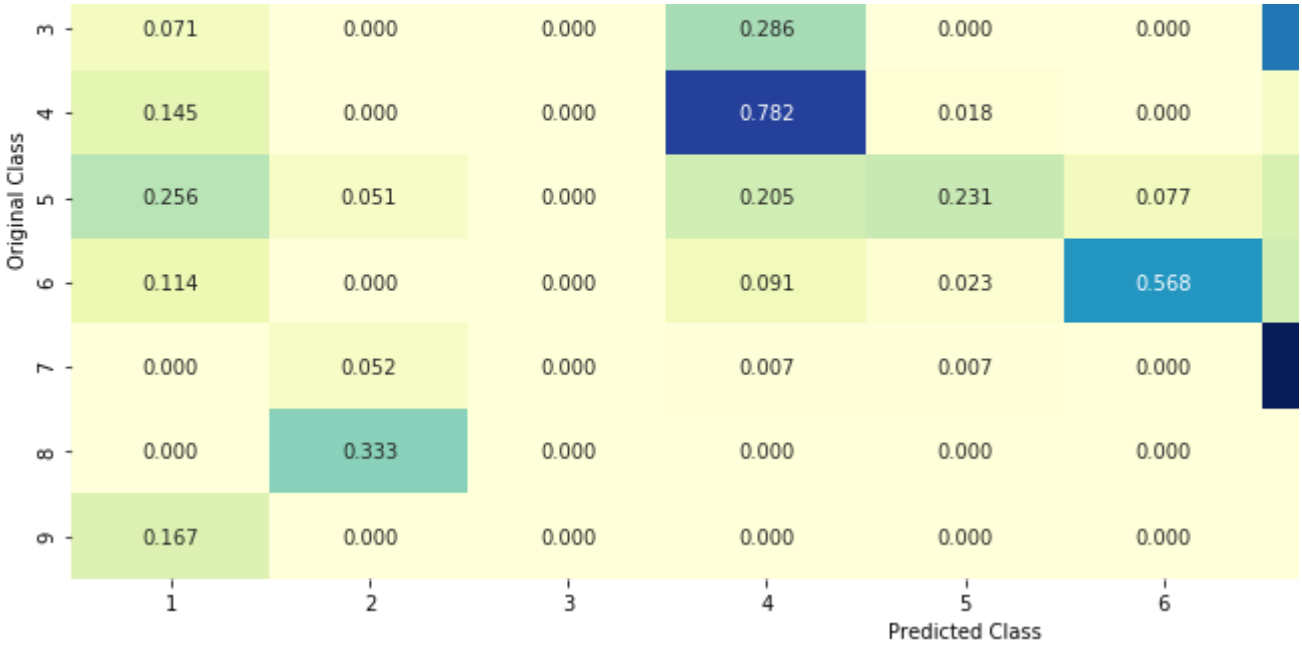
```
Log loss : 1.0193858139473322
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```




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


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





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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0418 0.7849 0.0179 0.0223 0.0472 0.021  0.0517 0.0
Actual Class : 2
--------------------------------------------------
Out of the top  500  features  0 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
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-1*abs(clf.coef_))[predicted_cls-1][:,:no_feature]
```

```
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].il
```

```
Predicted Class : 7
Predicted Class Probabilities: [[0.0299 0.144  0.0357 0.0376 0.0621 0.0041 0.6801 0.0
Actual Class : 2
--------------------------------------------------
313 Text feature [10] present in test data point [True]
464 Text feature [11] present in test data point [True]
Out of the top  500  features  2 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=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Nc
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# --------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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 = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
```

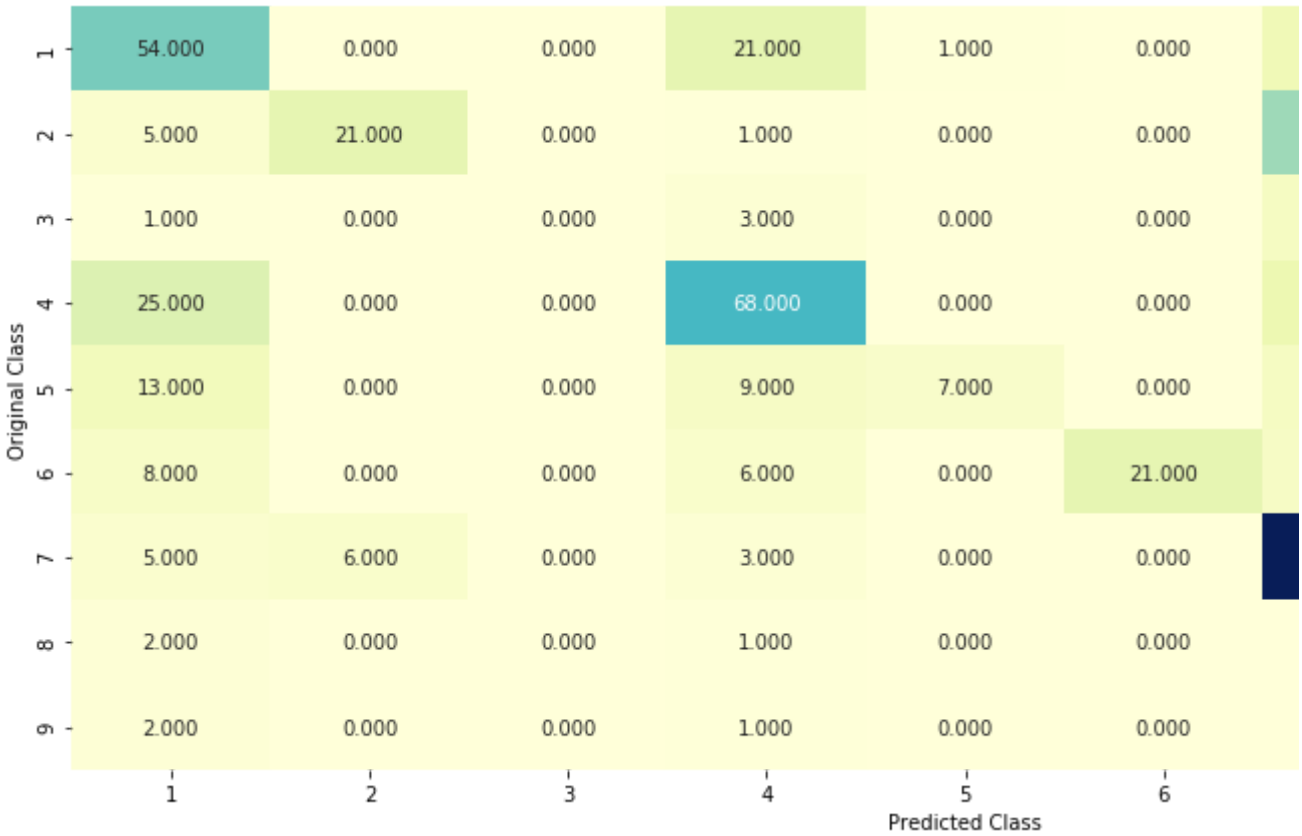
```
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, 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=1
        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_
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_
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:
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation l
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:"
```

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2123898639923436
for n_estimators = 100 and max depth =  10
Log Loss : 1.2079599488392754
for n_estimators = 200 and max depth =  5
Log Loss : 1.198475205032409
for n_estimators = 200 and max depth =  10
Log Loss : 1.2010852612150646
for n_estimators = 500 and max depth =  5
Log Loss : 1.1872142079564816
for n_estimators = 500 and max depth =  10
Log Loss : 1.196300214483638
for n_estimators = 1000 and max depth =  5
Log Loss : 1.1841425171896085
for n_estimators = 1000 and max depth =  10
Log Loss : 1.1919204507234764
for n_estimators = 2000 and max depth =  5
Log Loss : 1.1837999271630917
for n_estimators = 2000 and max depth =  10
Log Loss : 1.1905664396534934
For values of best estimator =  2000 The train log loss is: 0.8939627894440503
For values of best estimator =  2000 The cross validation log loss is: 1.183799927163
For values of best estimator =  2000 The test log loss is: 1.2623306496209639
```

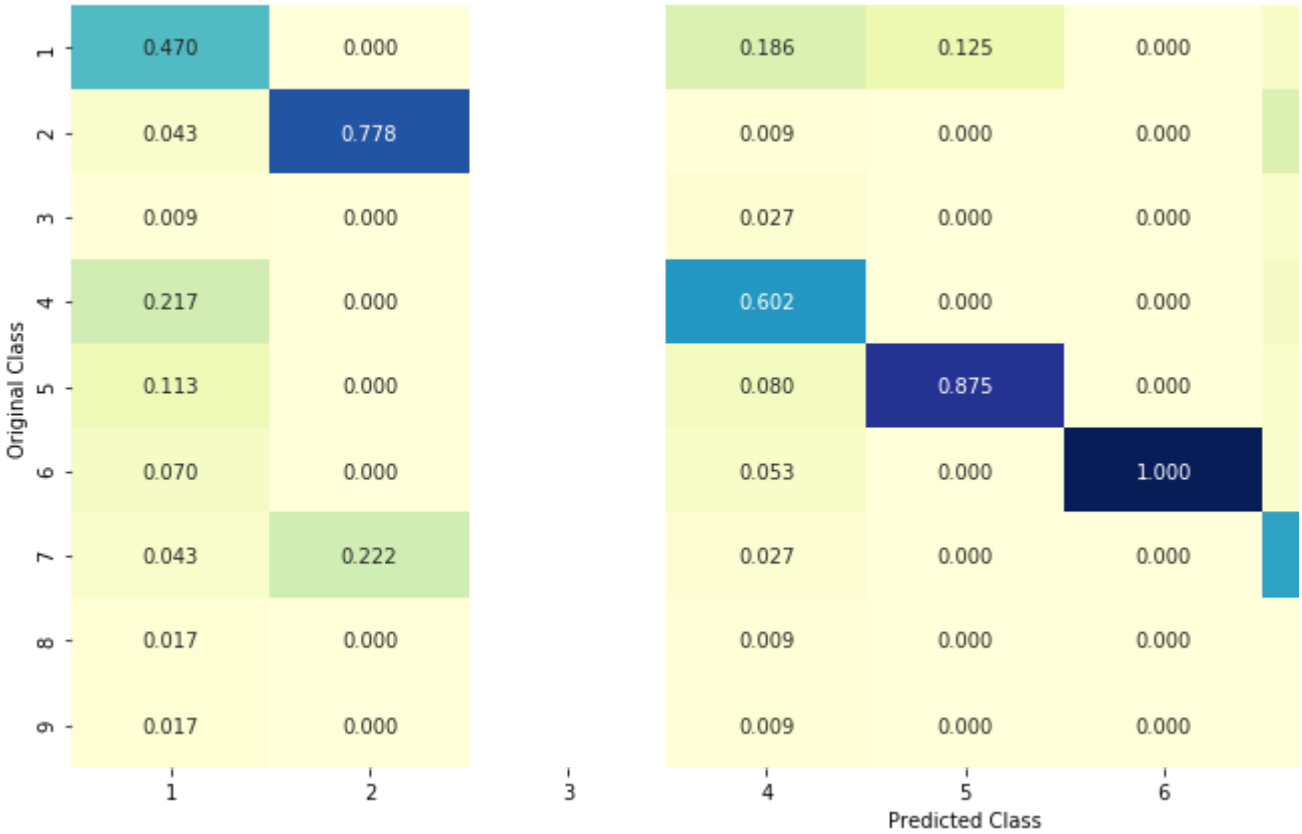
4.5.2. Testing model with best hyper parameters (One Hot Encoding)

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

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

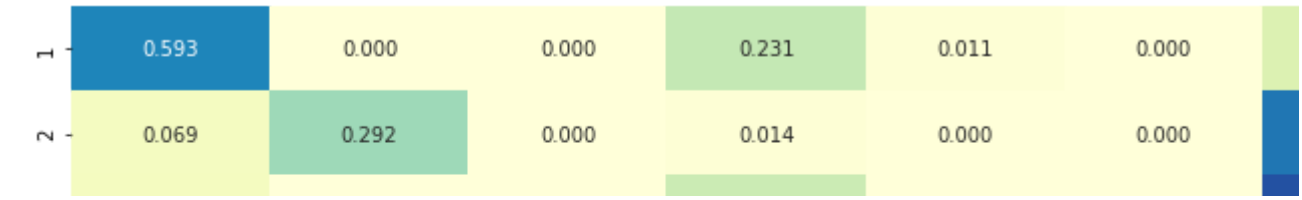
```
Log loss : 1.1837999271630917
Number of mis-classified points : 0.4116541353383459
-------------------- Confusion matrix --------------------
```

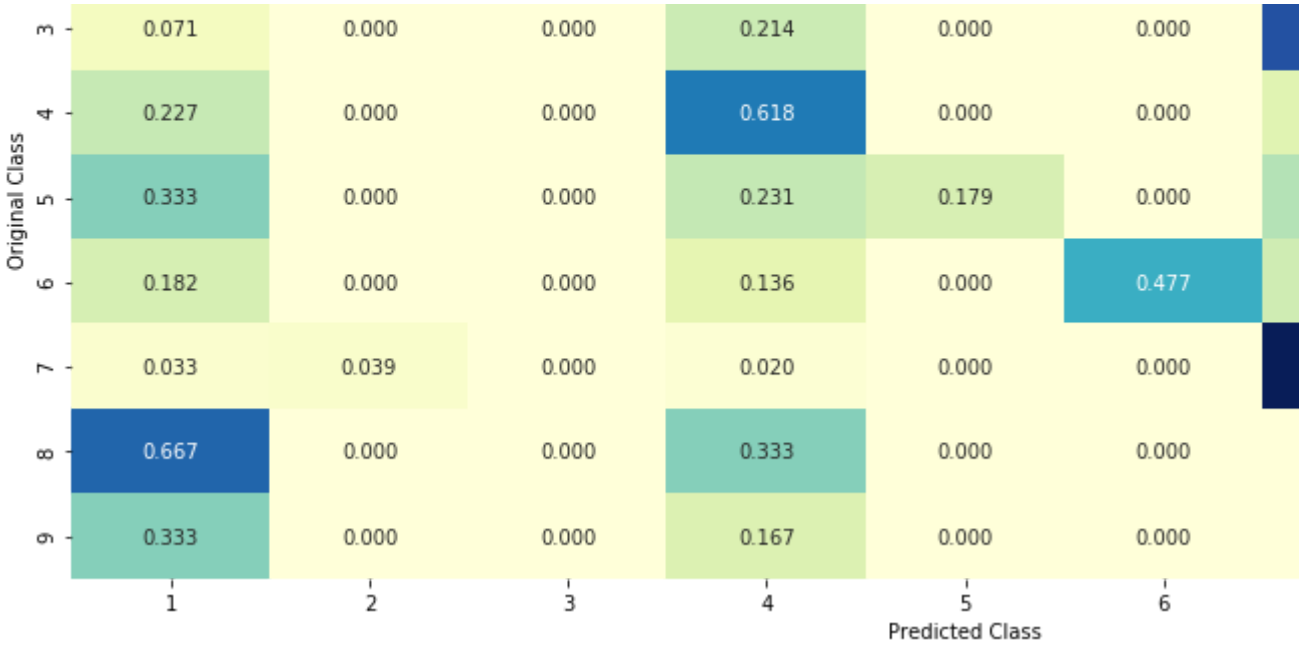



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





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_
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
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0492 0.3968 0.0301 0.0857 0.0569 0.042  0.3261 0.0
Actual Class : 2
--------------------------------------------------
12 Text feature [109] present in test data point [True]
Out of the top  100  features  1 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
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[
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0138 0.5763 0.0149 0.0202 0.0332 0.026  0.3098 0.0
Actuall Class : 2
--------------------------------------------------
42 Text feature [104] present in test data point [True]
Out of the top  100  features  1 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=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=Nc
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# ---------------------------------


# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/ge
# ----------------------------
# 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 = [10,50,100,200,500,1000]
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
        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=1
        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_
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_
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:",lo
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log 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
```

```
for n_estimators = 10 and max depth =  2
Log Loss : 2.086875645592438
for n_estimators = 10 and max depth =  3
Log Loss : 1.657743511320897
for n_estimators = 10 and max depth =  5
Log Loss : 1.5346621771119413
for n_estimators = 10 and max depth =  10
Log Loss : 1.7465073792602968
for n_estimators = 50 and max depth =  2
Log Loss : 1.6660379753142083
for n_estimators = 50 and max depth =  3
Log Loss : 1.4642083057356599
for n_estimators = 50 and max depth =  5
Log Loss : 1.33386874914904
for n_estimators = 50 and max depth =  10
Log Loss : 1.6129944300444556
for n_estimators = 100 and max depth =  2
Log Loss : 1.5067387343673515
for n_estimators = 100 and max depth =  3
Log Loss : 1.4431070930095704
for n_estimators = 100 and max depth =  5
Log Loss : 1.2990703795958765
for n_estimators = 100 and max depth =  10
Log Loss : 1.5681175089890185
for n_estimators = 200 and max depth =  2
Log Loss : 1.5458662668211995
for n_estimators = 200 and max depth =  3
Log Loss : 1.441997159468693
for n_estimators = 200 and max depth =  5
Log Loss : 1.3882248733842144
for n_estimators = 200 and max depth =  10
Log Loss : 1.6077148634769651
for n_estimators = 500 and max depth =  2
Log Loss : 1.5837982259371914
for n_estimators = 500 and max depth =  3
Log Loss : 1.4756831763741975
for n_estimators = 500 and max depth =  5
Log Loss : 1.418354940420824
for n_estimators = 500 and max depth =  10
Log Loss : 1.6487332065239972
for n_estimators = 1000 and max depth =  2
Log Loss : 1.5641483252214516
for n_estimators = 1000 and max depth =  3
Log Loss : 1.47874883985345
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4049997063196757
for n_estimators = 1000 and max depth =  10
Log Loss : 1.6377662246080875
For values of best alpha =  100 The train log loss is: 0.0660813739997366
For values of best alpha =  100 The cross validation log loss is: 1.2990703795958758
For values of best alpha =  100 The test log loss is: 1.3516019428735186
```

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

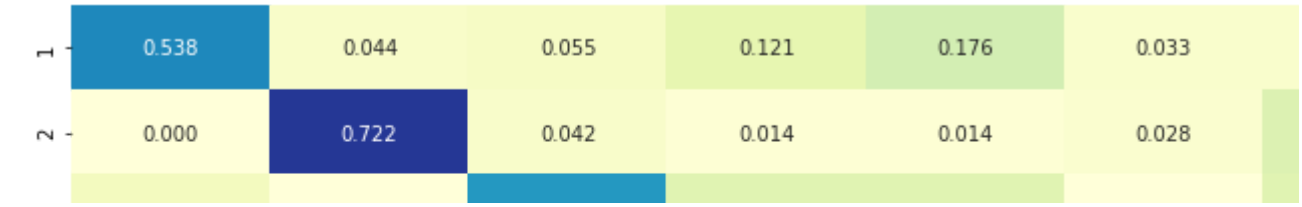
```
# --------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
```

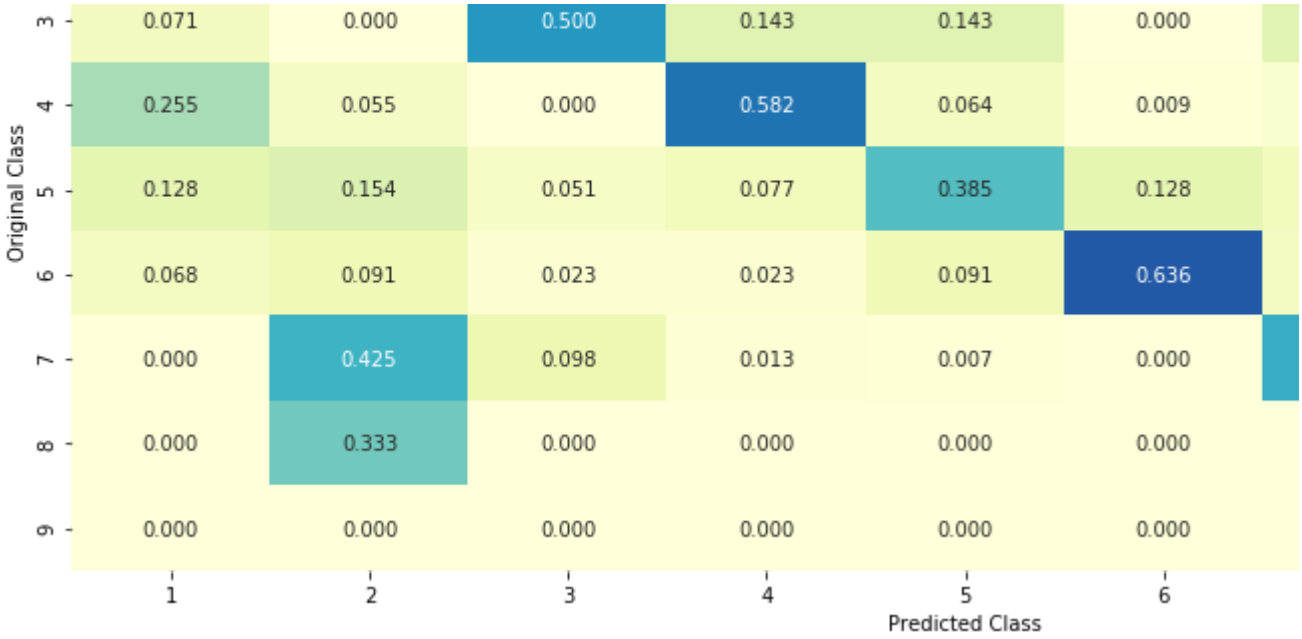
```
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# class_weight=None)

# 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/rand
# --------------------------------

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

```
Log loss : 1.299070379595876
Number of mis-classified points : 0.4567669172932331
-------------------- Confusion matrix --------------------
```


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


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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', 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_responseCodi
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")
    else:
        print("Text is important feature")
```

```
Predicted Class : 2
Predicted Class Probabilities: [[0.0537 0.6091 0.0948 0.0424 0.0226 0.0391 0.0449 0.0
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene 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
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene 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_responseCodi
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")
    else:
        print("Text is important feature")
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.0135 0.3142 0.3264 0.0183 0.0231 0.0274 0.2389 0.0
Actual Class : 2
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Gene 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
Variation is important feature
Text is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene 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/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# 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 Desc
# predict(X)  Predict class labels for samples in X.

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


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

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

# 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/math
# ---------------------------------


# read more about support vector machines with linear kernals here http://scikit-learn.org
# ---------------------------------
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=Non
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=No
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, v
# 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/rand
# ---------------------------------


clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', randc
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_
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
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba
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_onehot
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
```

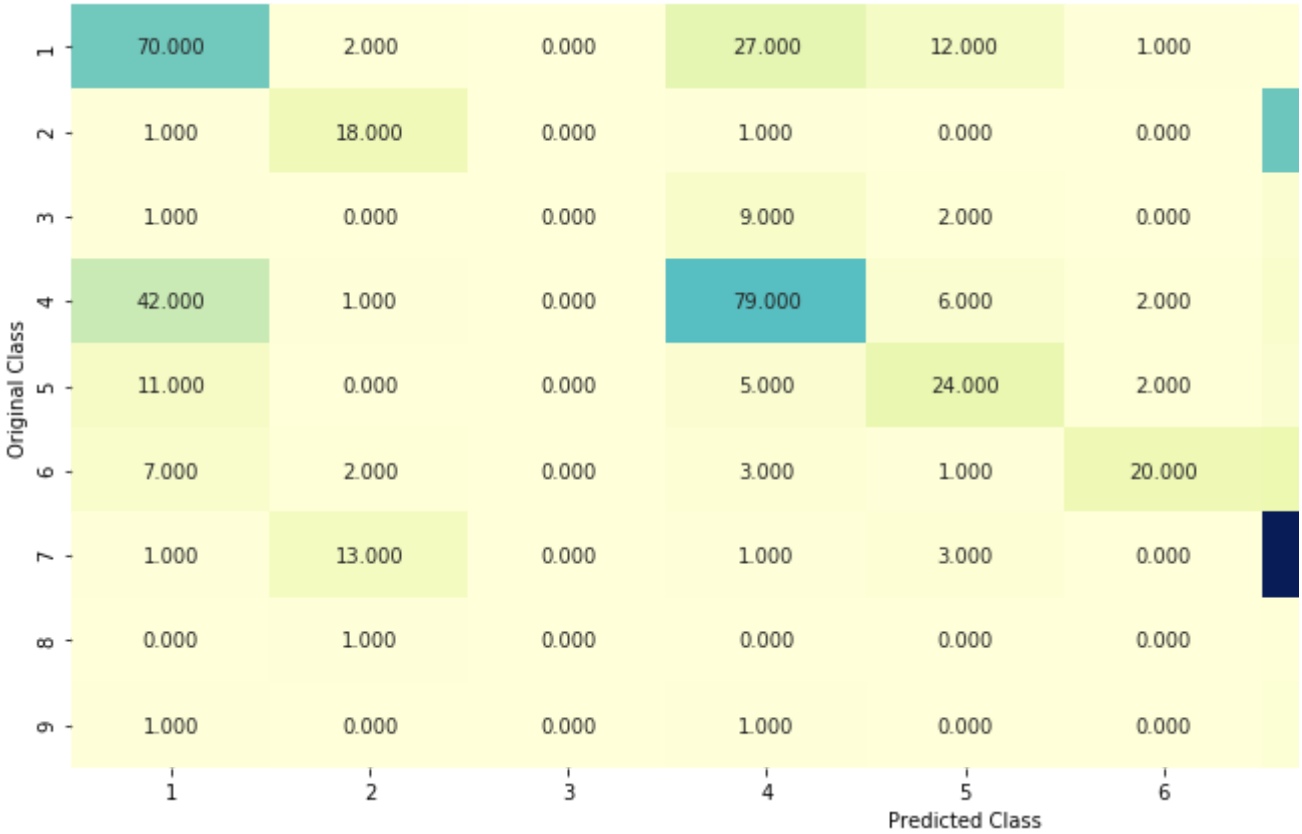
```
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999
for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss
    log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.06
Support vector machines : Log Loss: 1.77
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 1.817
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 1.714
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.344
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.289
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.656
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 2.036
```

4.7.2 testing the model with the best hyper parameters

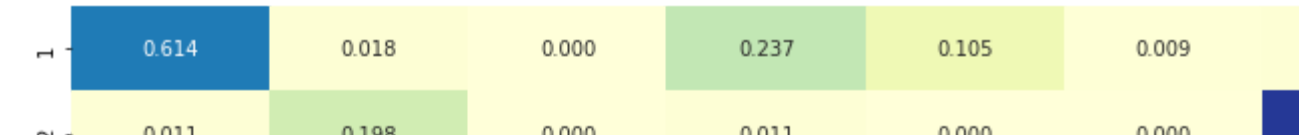
```
lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr,
sclf.fit(train_x_onehotCoding, train_y)

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_onehotCodi
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

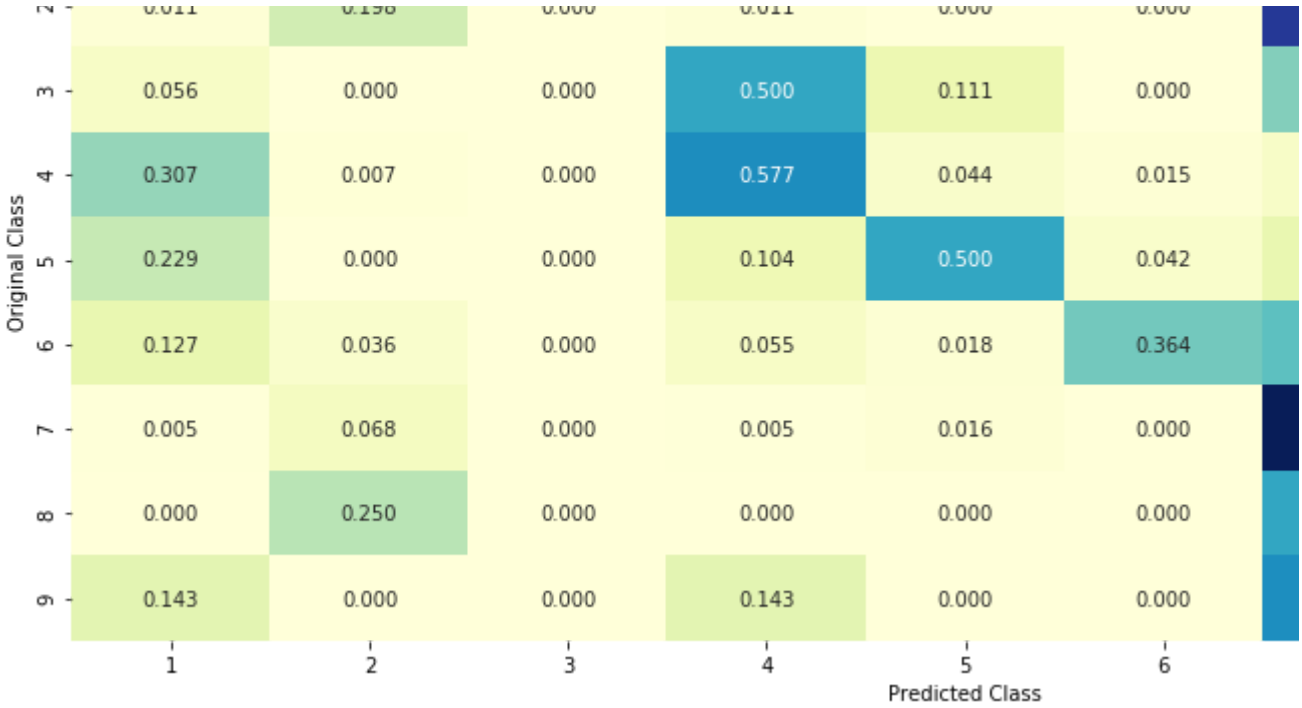
```
Log loss (train) on the stacking classifier : 0.3329931046985514
Log loss (CV) on the stacking classifier : 1.2893915049964633
Log loss (test) on the stacking classifier : 1.3508757312562063
Number of missclassified point : 0.42105263157894735
-------------------- Confusion matrix --------------------
```




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


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

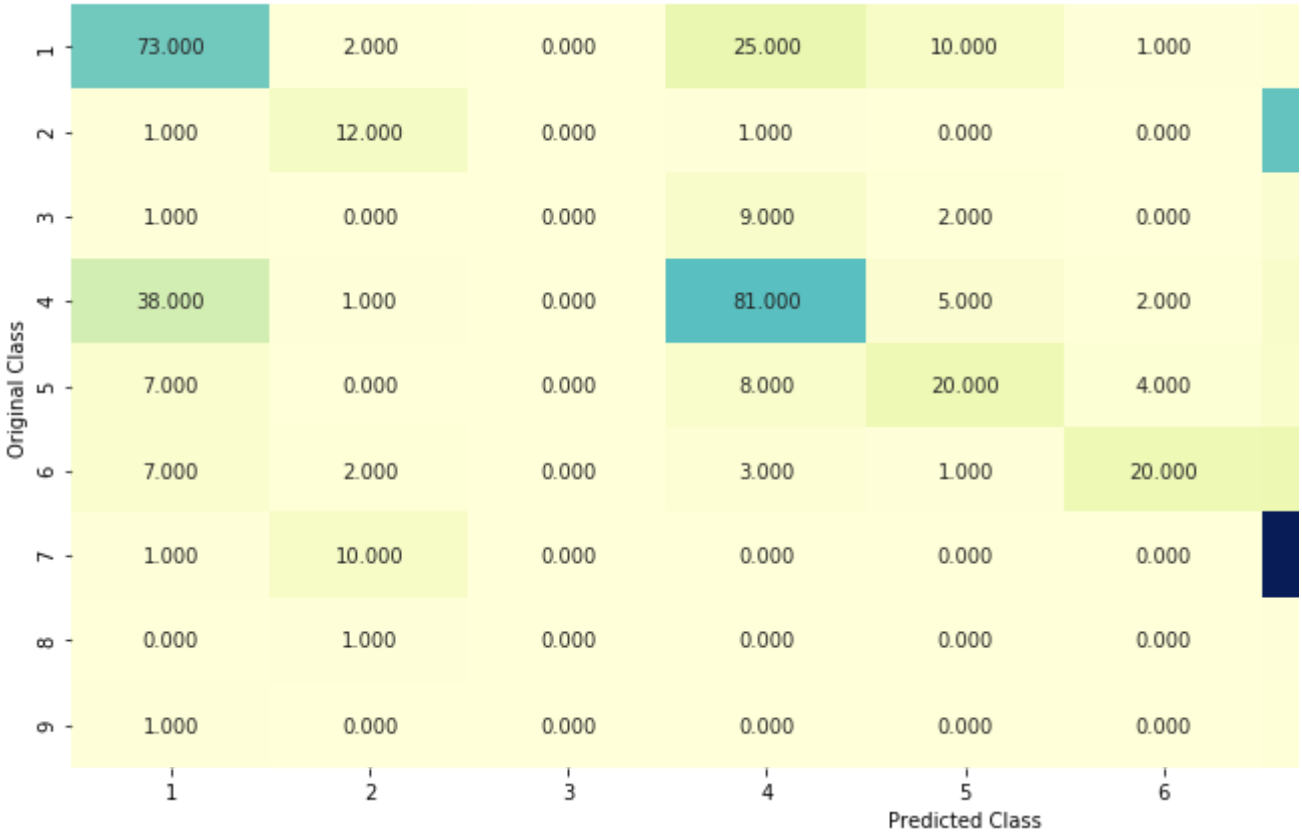




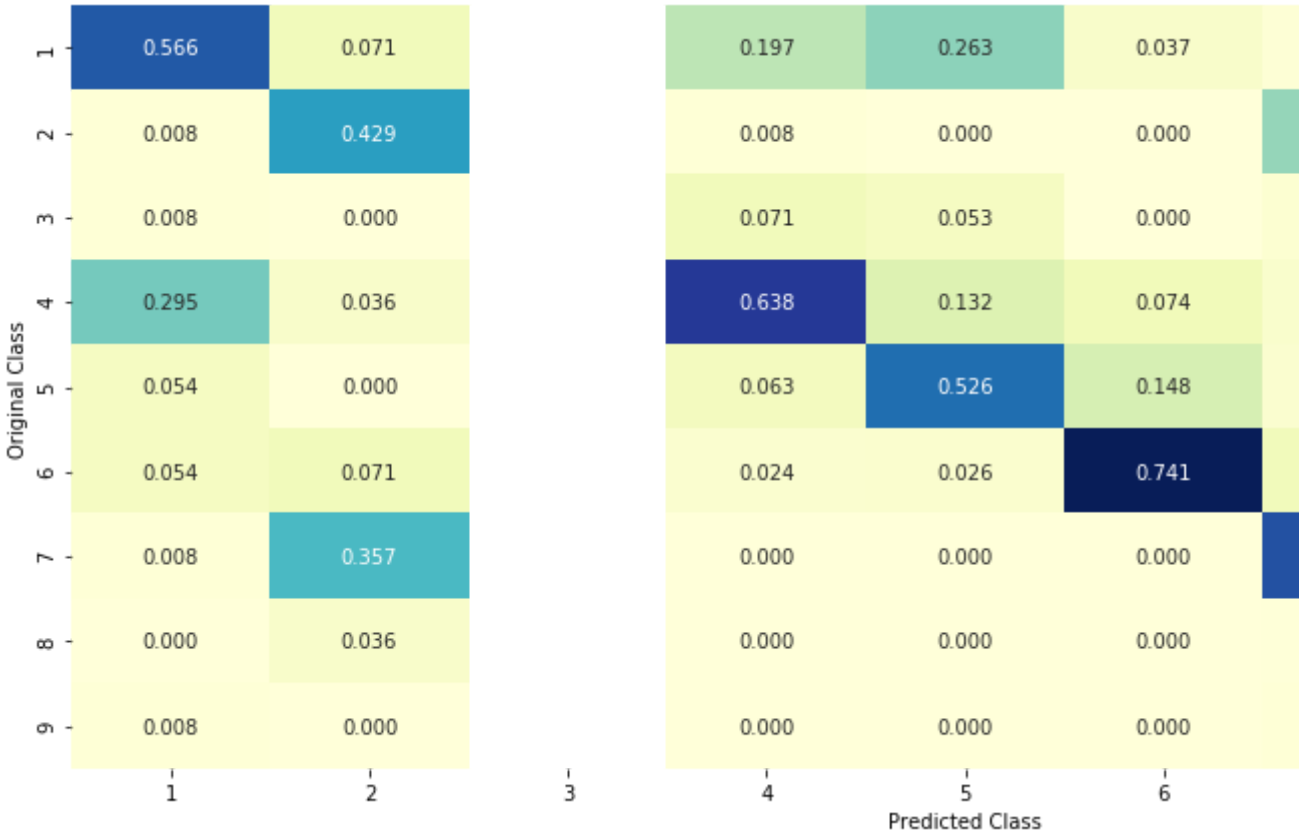
4.7.3 Maximum Voting classifier

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

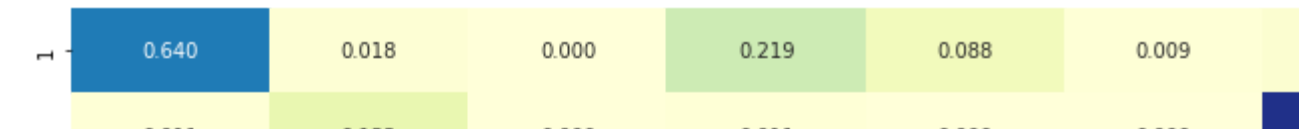
```
Log loss (train) on the VotingClassifier: 0.7916048327656435
Log loss (CV) on the VotingClassifier: 1.2176324976602917
Log loss (test) on the VotingClassifier: 1.2826968479830854
Number of missclassified point : 0.41353383458646614
-------------------- Confusion matrix --------------------
```

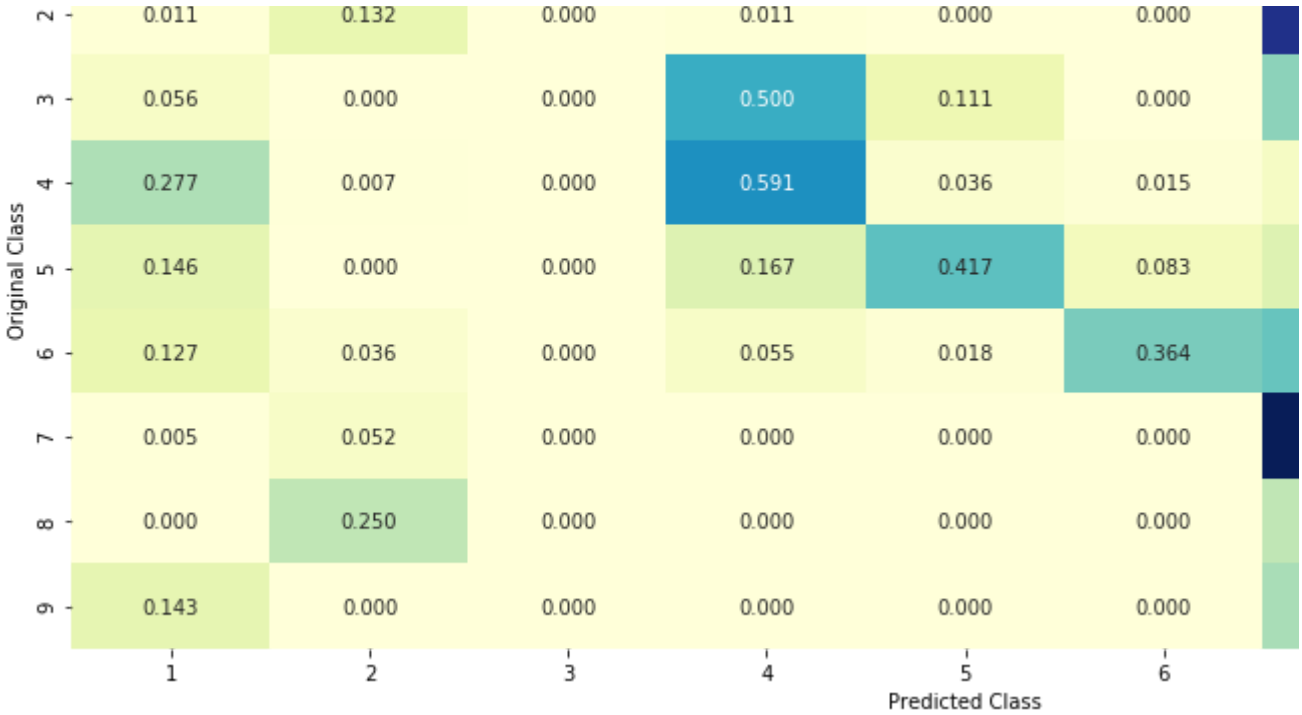



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer and run the same c
2. Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss t

WE WILL BE TRAINING 2 MODELS OF LOGISTIC REGRESSION WITHout BALANCED CLASS WEIGH

1.MODEL TRAINED WITH FEATURES WITHOUT PERFORMING ANY FEATURE ENGINEERING

2.MODEL TRAINED WITH FEATURES BY PERFORMING FEATURE ENGINEERING TECHNIQUES

2.1-TEXT_DATA-TFIDF VECTORIZATION

2.2-MEAN ENCODING FOR OF GENE AND VARIATION features

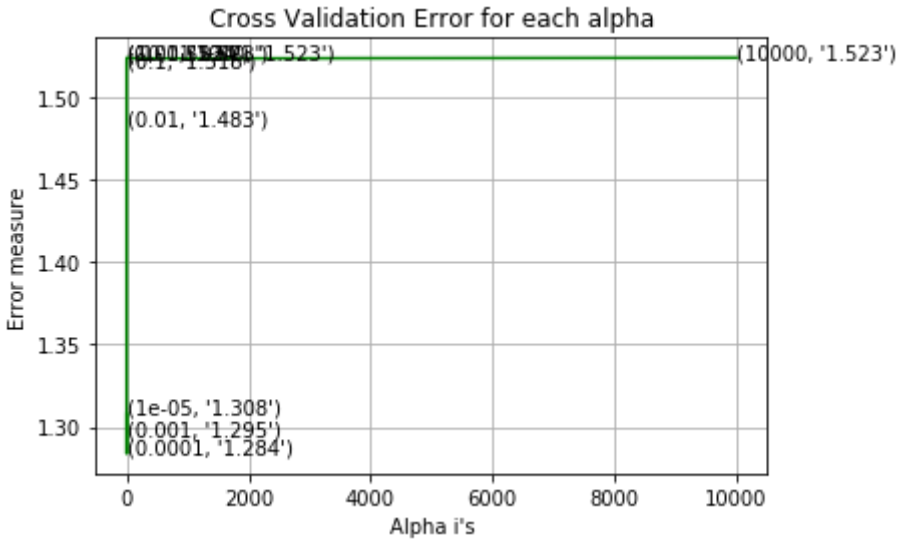
```
train_df.columns
var_train=train_df['Variation'].values;
var_test=test_df['Variation'].values;
var_cv=cv_df['Variation'].values

gene_train=train_df['Gene'].values;
gene_test=test_df['Gene'].values;
gene_cv=cv_df['Gene'].values

text_train=train_df['TEXT'].values;
text_test=test_df['TEXT'].values;
text_cv=cv_df['TEXT'].values
```

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer=CountVectorizer(min_df=10,ngram_range=(1, 2))
var_train=vectorizer.fit_transform(var_train)
var_test=vectorizer.transform(var_test)
var_cv=vectorizer.transform(var_cv)

gene_train=vectorizer.fit_transform(gene_train)
gene_test=vectorizer.transform(gene_test)
gene_cv=vectorizer.transform(gene_cv)


text_train=vectorizer.fit_transform(text_train)
text_test=vectorizer.transform(text_test)
text_cv=vectorizer.transform(text_cv)


from scipy.sparse import hstack
data_train=hstack([var_train,gene_train,text_train]).tocsr()
data_test=hstack([var_test,gene_test,text_test]).tocsr()
data_cv=hstack([var_cv,gene_cv,text_cv]).tocsr()


print(data_train.shape)
print(data_test.shape)
print(data_cv.shape)
```

```
(2124, 235934)
(665, 235934)
(532, 235934)
```

```
cv_values=[]
alpha=[10 ** x for x in range(-5, 5)]
for c in tqdm(alpha):
  print("for alpha =", c)
  lr = LogisticRegression(random_state=0, C=c,class_weight='balanced',n_jobs=-1)
  clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
  clf.fit(data_train,y_train)
  pre_cv=clf.predict_proba(data_cv)
  print(c,log_loss(y_cv, pre_cv))
  cv_values.append(log_loss(y_cv, pre_cv))
print(alpha,cv_values)
print(len(alpha),len(cv_values))
print(type(cv_values))
fig, ax = plt.subplots()
ax.plot(alpha, cv_values,c='g')
for i, t in enumerate(np.round(cv_values,3)):
  ax.annotate((alpha[i],str(t)), (alpha[i],cv_values[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]for alpha = 1e-05

 10%|▉         | 1/10 [04:18<38:46, 258.53s/it]1e-05 1.307704760189509
for alpha = 0.0001

 20%|▊         | 2/10 [08:33<34:19, 257.42s/it]0.0001 1.2837278442535625
for alpha = 0.001

 30%|▋         | 3/10 [12:43<29:46, 255.15s/it]0.001 1.2948793444371294
for alpha = 0.01

 40%|▌         | 4/10 [16:50<25:16, 252.72s/it]0.01 1.482630497950825
for alpha = 0.1

 50%|▍         | 5/10 [20:54<20:50, 250.06s/it]0.1 1.5182725411359548
for alpha = 1

 60%|▎         | 6/10 [24:58<16:33, 248.46s/it]1 1.5230694403732323
for alpha = 10

 70%|▏         | 7/10 [29:06<12:24, 248.14s/it]10 1.5235605057949146
for alpha = 100

 80%|▌         | 8/10 [33:12<08:14, 247.45s/it]100 1.5231432546278938
for alpha = 1000

 90%|▉         | 9/10 [37:16<04:06, 246.55s/it]1000 1.5230428598845918
for alpha = 10000

100%|▉         | 10/10 [41:20<00:00, 245.72s/it]
10000 1.523460007982243
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000] [1.307704760189509, 1.2837
10 10
<class 'list'>
```



from above graph best value for C=0.001

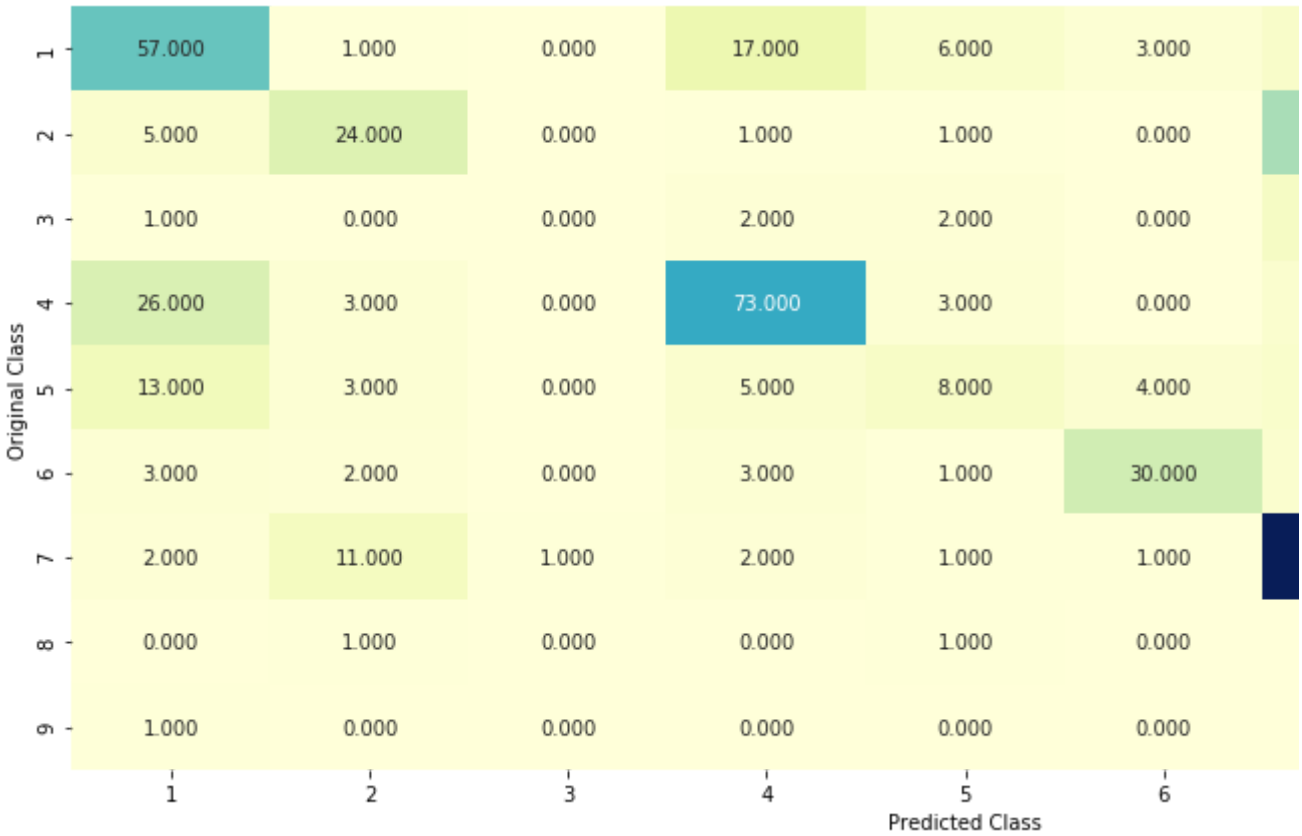
```
l = LogisticRegression(random_state=0, C=0.001,class_weight='balanced',n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=l,method='sigmoid')
clf.fit(data_train,y_train)
```

```
clf.fit(data_train,y_train)
pre_test=clf.predict_proba(data_test)
print("Log Loss value for the test data is ",log_loss(y_test, pre_test))
```

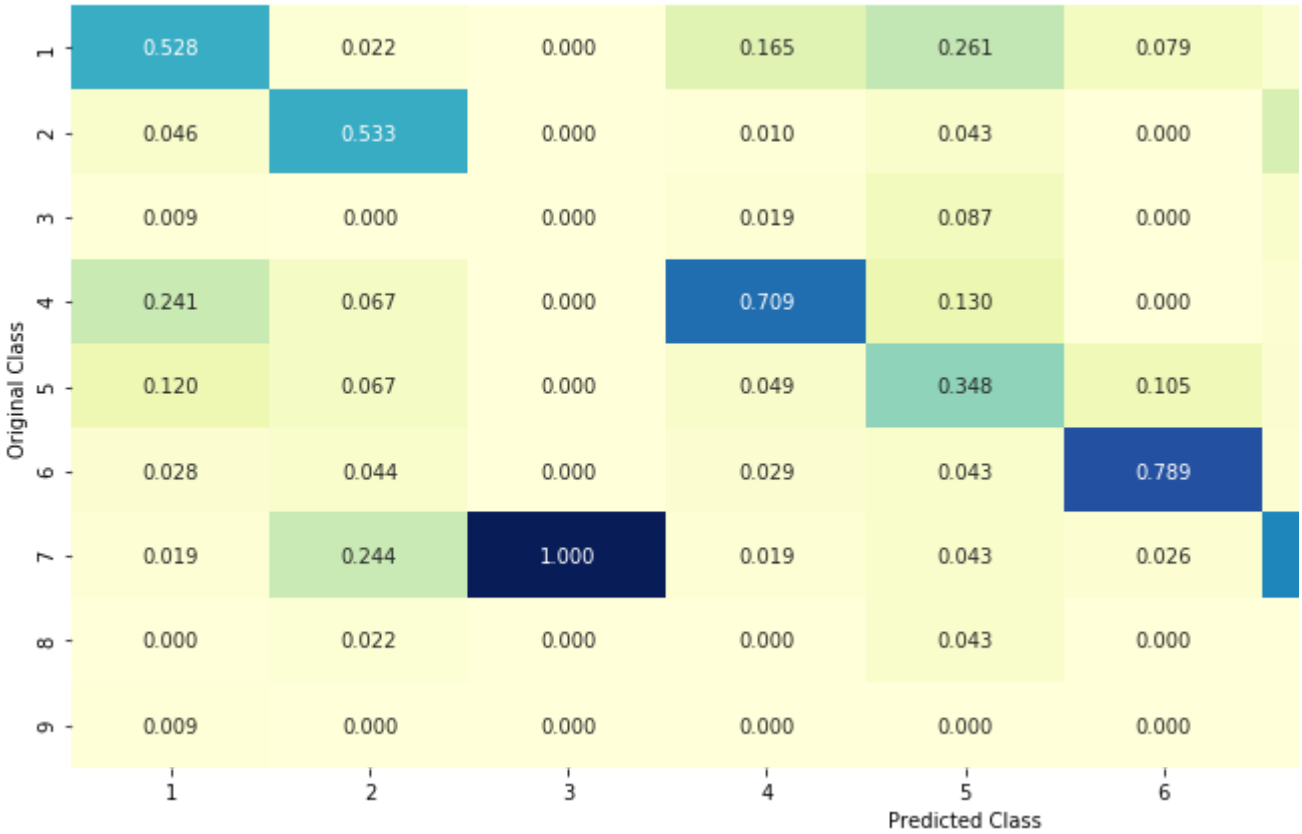
```
Log Loss value for the test data is  1.3722961595858407
```

```
predict_and_plot_confusion_matrix(data_train, y_train,data_cv,y_cv, clf)
```

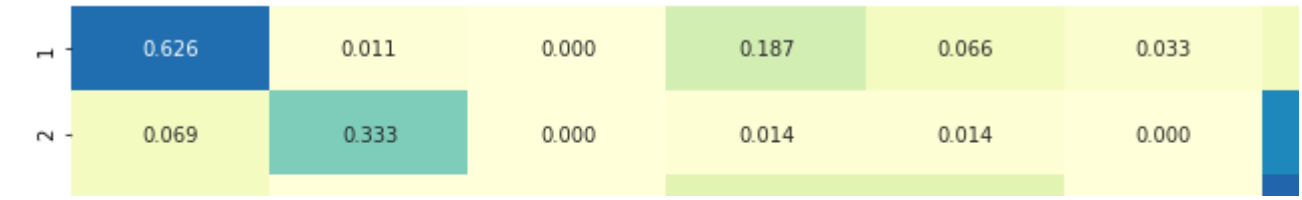
```
Log loss : 1.0824708168768304
Number of mis-classified points : 0.37406015037593987
-------------------- Confusion matrix --------------------
```

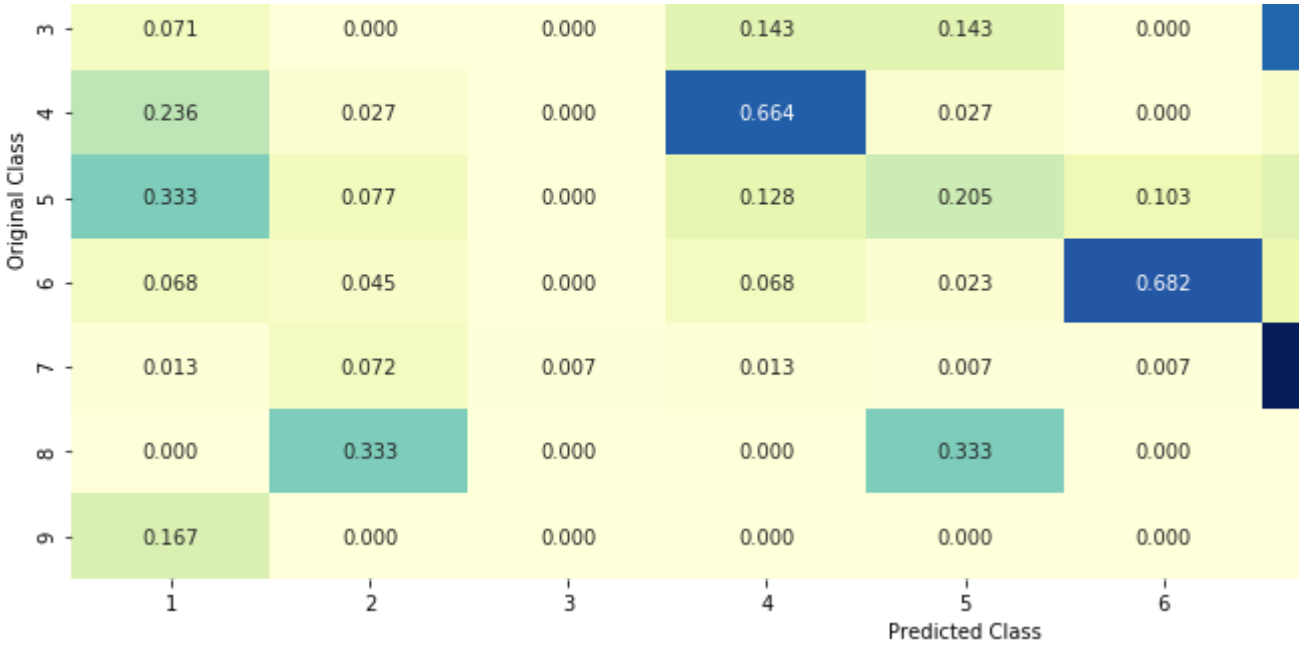



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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





Now we will do feature engineering on all the 3 features so t

We will select top 5000 features using idf values from text data and vectorize them using TFIDF ve

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_text= TfidfVectorizer( min_df=10,max_features=5000)
vectorizer_tfidf_text.fit(train_df["TEXT"])
text_tfidf_train = vectorizer_tfidf_text.transform(train_df["TEXT"])
text_tfidf_test = vectorizer_tfidf_text.transform(test_df["TEXT"])
text_tfidf_cv = vectorizer_tfidf_text.transform(cv_df["TEXT"])
print("Shape of matrix after one hot encoding ",text_tfidf_train.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_test.shape)
print("Shape of matrix after one hot encoding ",text_tfidf_cv.shape)
```

```
Shape of matrix after one hot encoding  (2124, 5000)
Shape of matrix after one hot encoding  (665, 5000)
Shape of matrix after one hot encoding  (532, 5000)
```

```
print(var_train.shape)
print(gene_train.shape)
print(text_train.shape)
print(var_test.shape)
print(gene_test.shape)
print(text_test.shape)
print(var_cv.shape)
print(gene_cv.shape)
print(text_cv.shape)
```

```
(2124, 5)
(2124, 55)
(2124, 222813)
(665, 5)
(665, 55)
(665, 222813)
(532, 5)
(532, 55)
(532, 222813)
```

```
vargen_train=pd.DataFrame()
vargen_test=pd.DataFrame()
vargen_cv=pd.DataFrame()
```

We will combine 2 features in single column

```
vargen_train['var_gene']=train_df['Variation']+' '+train_df['Gene']
vargen_test['var_gene']=test_df['Variation']+' '+test_df['Gene']
vargen_cv['var_gene']=cv_df['Variation']+' '+cv_df['Gene']
```

```
vargene_train=pd.DataFrame()
vargene_test=pd.DataFrame()
vargene_cv=pd.DataFrame()
```

```
var_n_train=pd.DataFrame()
var_n_test=pd.DataFrame()
var_n_cv=pd.DataFrame()
```

Feature engineering steps:

1.We will count length for the combined var_gene feature

2..We will select top 10 words from variation and gene andperform mean encoding i.e check if wo present 1 is appended else 0 is appended.

3.TDFIDF Vectorization for text and select 5000 best words

mean encoding Variation feature

```
top_10_var=[ x for x in train_df['Variation'].value_counts().sort_values(ascending=False).
for i in top_10_var:
  var_n_train[i]=np.where(train_df['Variation']==i,1,0)
  var_n_test[i]=np.where(test_df['Variation']==i,1,0)
  var_n_cv[i]=np.where(cv_df['Variation']==i,1,0)
```

```
gene_n_train=pd.DataFrame()
gene_n_test=pd.DataFrame()
gene_n_cv=pd.DataFrame()
```

Mean encoding Gene feature

```
top_10_gene=[ x for x in train_df['Gene'].value_counts().sort_values(ascending=False).head
for i in top_10_gene:
  gene_n_train[i]=np.where(train_df['Gene']==i,1,0)
  gene_n_test[i]=np.where(test_df['Gene']==i,1,0)
  gene_n_cv[i]=np.where(cv_df['Gene']==i,1,0)
```

```
text_n_train=pd.DataFrame()
text_n_test=pd.DataFrame()
text_n_cv=pd.DataFrame()
```

```
count=pd.DataFrame()
line=[]
cnt_train=[]
cnt_test=[]
cnt_cv=[]
cnt_text_train=[]
cnt_text_test=[]
cnt_text_cv=[]
```

counting number of words in Var_gene feature

```
for i in vargen_train['var_gene']:
  for j in i:
    line.append(j)
  cnt_train.append(len(line))
for i in vargen_test['var_gene']:
  for j in i:
    line.append(j)
  cnt_test.append(len(line))
for i in vargen_cv['var_gene']:
  for j in i:
    line.append(j)
  cnt_cv.append(len(line))
cnt_train=pd.DataFrame(cnt_train)
cnt_test=pd.DataFrame(cnt_test)
cnt_cv=pd.DataFrame(cnt_cv)
print(cnt_train.shape)
print(cnt_test.shape)
print(cnt_cv.shape)
```

```
(2124, 1)
(665, 1)
(532, 1)
```

Normalizing length of text

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
normalizer.fit(cnt_train.values.reshape(-1,1))
cnt_train = normalizer.transform(cnt_train.values.reshape(-1,1))
cnt_cv = normalizer.transform(cnt_cv.values.reshape(-1,1))
cnt_test = normalizer.transform(cnt_test.values.reshape(-1,1))
print("After vectorizations")
print(cnt_train.shape, y_train.shape)
print(cnt_cv.shape, y_cv.shape)
print(cnt_test.shape, y_test.shape)
```

```
After vectorizations
(2124, 1) (2124,)
(532, 1) (532,)
(665, 1) (665,)
```

Stacking the feature engineered features

```
data_train_fe=hstack([var_train,gene_train,text_tfidf_train,var_n_train,gene_n_train,cnt_t
data_test_fe=hstack([var_test,gene_test,text_tfidf_test,var_n_test,gene_n_test,cnt_test]).
data_cv_fe=hstack([var_cv,gene_cv,text_tfidf_cv,var_n_cv,gene_n_cv,cnt_cv]).tocsr()
```

```
print(data_train_fe.shape)
print(data_test_fe.shape)
print(data_cv_fe.shape)
```

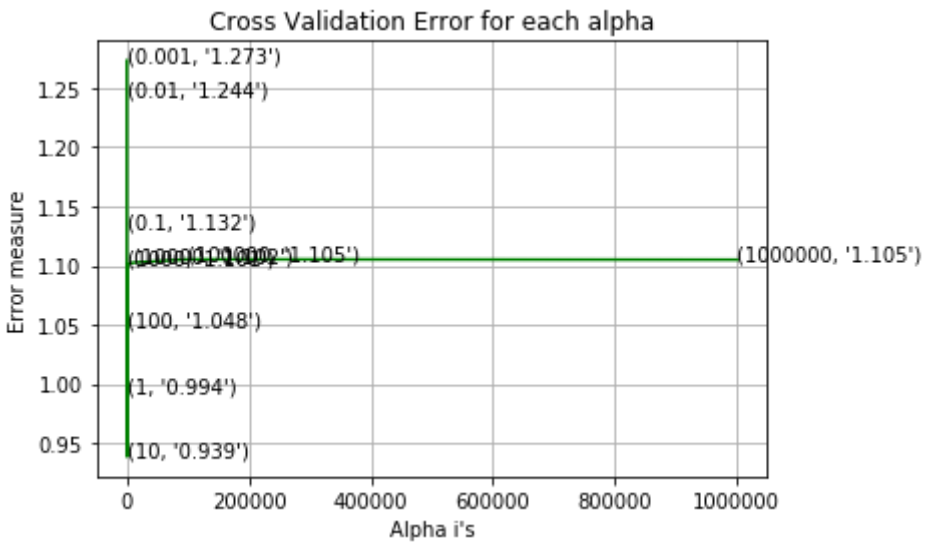
```
(2124, 5081)
(665, 5081)
(532, 5081)
```

Hyperparameter tuning using the CV_data

```
from tqdm import tqdm
cv_values=[]
alpha=[10 ** x for x in range(-3, 7)]
for c in tqdm(alpha):
  print("for alpha =", c)
  lr = LogisticRegression(random_state=0, C=c,n_jobs=-1)
  clf=CalibratedClassifierCV(base_estimator=lr,method='sigmoid')
  clf.fit(data_train_fe,y_train)
  pre_cv=clf.predict_proba(data_cv_fe)
  print(c,log_loss(y_cv, pre_cv))
  cv_values.append(log_loss(y_cv, pre_cv))
print(alpha,cv_values)
print(len(alpha),len(cv_values))
print(type(cv_values))
fig, ax = plt.subplots()
```

```
ax.plot(alpha, cv_values,c='g')
for i, t in enumerate(np.round(cv_values,3)):
  ax.annotate((alpha[i],str(t)), (alpha[i],cv_values[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
```

```
  0%|          | 0/10 [00:00<?, ?it/s]for alpha = 0.001
 10%|▉         | 1/10 [00:08<01:18,  8.69s/it]0.001 1.27309174912667
for alpha = 0.01
 20%|▊         | 2/10 [00:15<01:05,  8.22s/it]0.01 1.244138503182963
for alpha = 0.1
 30%|▉         | 3/10 [00:28<01:05,  9.42s/it]0.1 1.1315744489504467
for alpha = 1
 40%|▊         | 4/10 [00:49<01:17, 12.99s/it]1 0.9937219276589544
for alpha = 10
 50%|▉         | 5/10 [01:10<01:17, 15.41s/it]10 0.9385640085921434
for alpha = 100
 60%|▊         | 6/10 [01:31<01:08, 17.02s/it]100 1.0482154337714413
for alpha = 1000
 70%|▉         | 7/10 [01:52<00:55, 18.33s/it]1000 1.1010144397944943
for alpha = 10000
 80%|▊         | 8/10 [02:13<00:38, 19.15s/it]10000 1.102444900198064
for alpha = 100000
 90%|▉         | 9/10 [02:34<00:19, 19.65s/it]100000 1.1052350566275917
for alpha = 1000000
100%|▊         | 10/10 [02:55<00:00, 20.09s/it]1000000 1.104914598584836
[0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000, 100000, 1000000] [1.27309174912667, 1.244
10 10
<class 'list'>
```

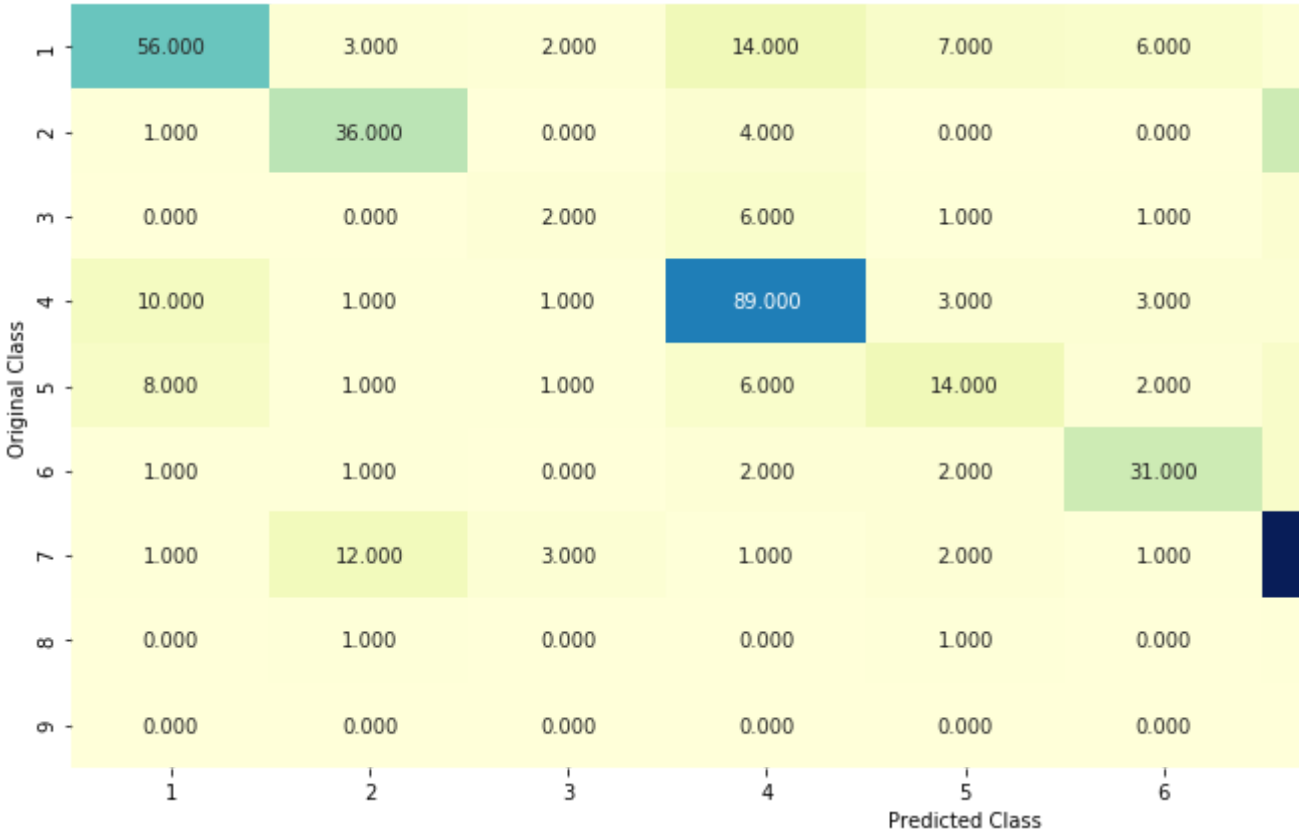


Using the best hyperparameter c=10

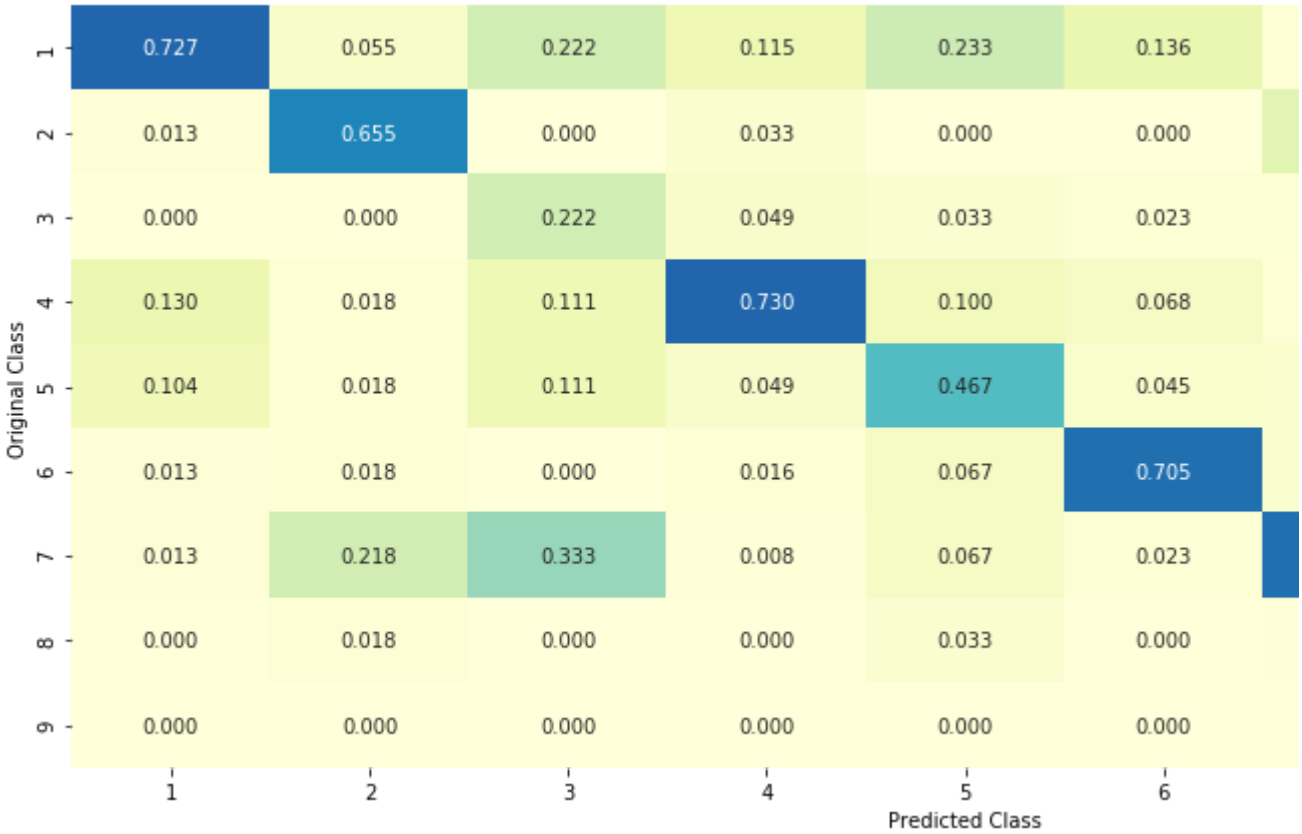
```
l = LogisticRegression(random_state=0, C=10,n_jobs=-1)
clf=CalibratedClassifierCV(base_estimator=l,method='sigmoid')
clf.fit(data_train_fe,y_train)
pre_test=clf.predict_proba(data_test_fe)
print("Log Loss value for the test data is ",log_loss(y_test, pre_test))
```

```
print('Log Loss value for the test data is ',log_loss(y_test, pre_test))
```

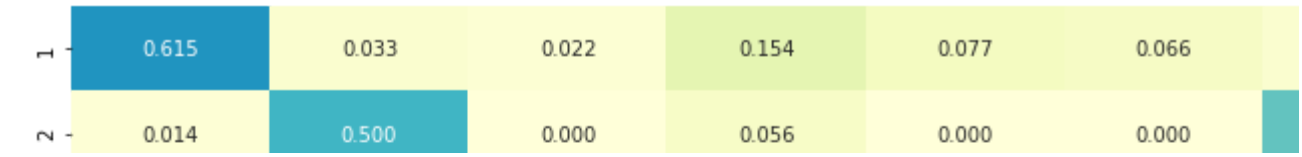
```
Log Loss value for the test data is  0.9887221971467367
Log loss : 0.9088807259280711
Number of mis-classified points : 0.3101503759398496
-------------------- Confusion matrix --------------------
```

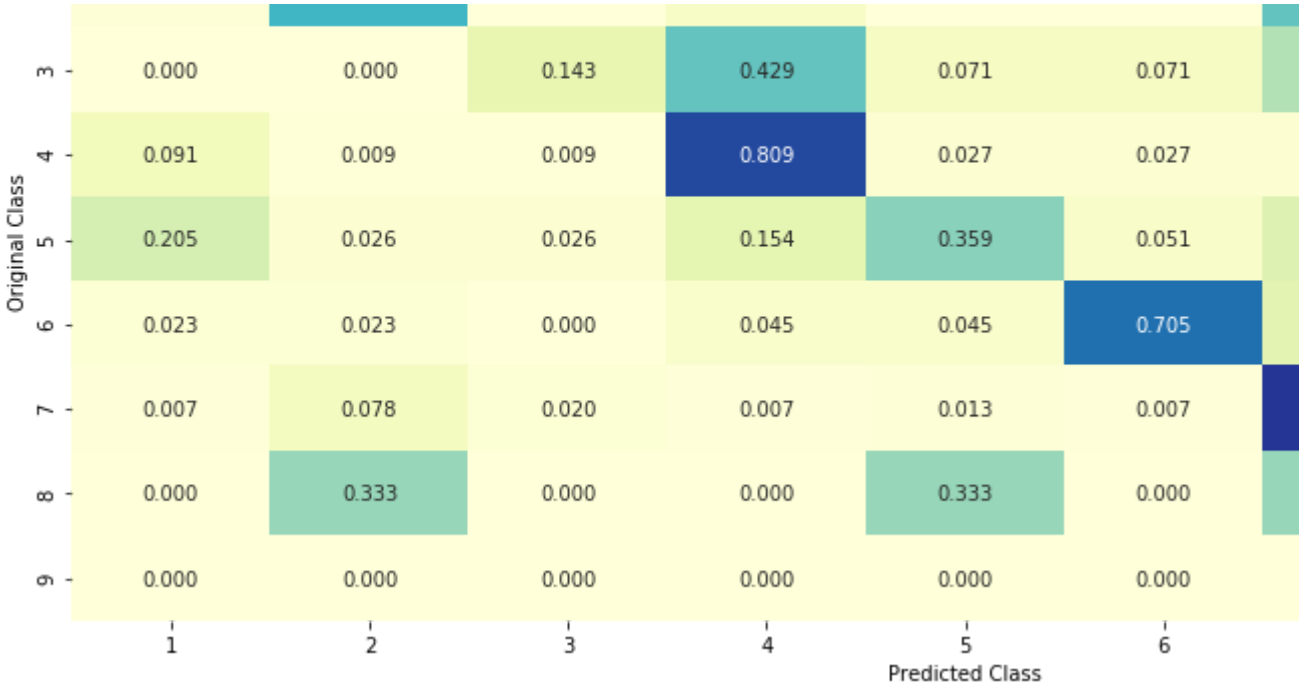



```
-------------------- Precision matrix (Columm Sum=1) --------------------
```




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






```
# Names of models
from prettytable import PrettyTable
model=['Naive Bayes ','KNN','Logistic Regression With Class balancing ','Logistic Regressi

train =[0.4446,0.5723,0.3820,0.3766,0.3066, 0.8939,0.3323,0.7116,0.6608,0.3762,0.9088]
test = [1.2911,1.1678,1.0949,1.0852,1.1182,1.2623,1.3509,1.2826,1.3516,1.3723,0.9887]
cv=[1.2405,1.0429,0.9949,0.9938,1.019,1.1837,1.2894,1.2176,1.2991,1.0824,0.9385]
mp=[40.9,34.58,34.77,35.34,34,41.17,42.11,41.53,45.67,37.406,31.01]
numbering=[1,2,3,4,5,6,7,8,9,10,11]
p = PrettyTable()
p.add_column("S.NO.",numbering)
p.add_column("model",model)
p.add_column("train",train)
p.add_column("test",test)
p.add_column("cv",cv)
p.add_column("% Misclassified Points",mp)
print(p)
```

```
+-------+-------------------------------------------------------------+--------+---
| S.NO. |                              model                          | train  |  t
+-------+-------------------------------------------------------------+--------+---
|   1   |                          Naive Bayes                        | 0.4446 | 1.
|   2   |                              KNN                            | 0.5723 | 1.
|   3   |            Logistic Regression With Class balancing         | 0.382  | 1.
|   4   |          Logistic Regression Without Class balancing        | 0.3766 | 1.
|   5   |                          Linear SVM                         | 0.3066 | 1.
|   6   |         Random Forest Classifier With One hot Encoding      | 0.8939 | 1.
|   7   |         Random Forest Classifier With Response Coding       | 0.3323 | 1.
|   8   |                     Stack Models:LR+NB+SVM                  | 0.7116 | 1.
|   9   |                   Maximum Voting classifier                 | 0.6608 | 1.
|   10  | CountVectorizer Features, including both unigrams and bigrams | 0.3762 | 1.
|   11  |                   after feature engineering                 | 0.9088 | 0.
+-------+-------------------------------------------------------------+--------+---
```

Conclusion:

After performing feature engineering the loss for train test and cv are below 1.