

# 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 literature.

### 1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

1. <https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25> (<https://www.forbes.com/sites/matthewherper/2017/06/03/a-new-cancer-drug-helped-almost-everyone-who-took-it-almost-heres-what-it-teaches-us/#2a44ee2f6b25>)
2. <https://www.youtube.com/watch?v=UwbuW7oK8rk> (<https://www.youtube.com/watch?v=UwbuW7oK8rk>)
3. <https://www.youtube.com/watch?v=qxXRKVompl8> (<https://www.youtube.com/watch?v=qxXRKVompl8>)

### 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> (<https://www.kaggle.com/c/msk-redefining-cancer-treatment/data>)
- We have two data files: one contains the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.
- Both these data files have a common column called ID
- Data file's information:
  - training\_variants (ID , Gene, Variations, Class)
  - training\_text (ID, Text)

#### 2.1.2. Example Data Point

##### *training\_variants*

---

ID, Gene, Variation, Class  
0, FAM58A, Truncating Mutations, 1  
1, CBL, W802\*, 2  
2, CBL, Q249E, 2  
...

##### *training\_text*

---

ID, Text  
0|Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we

demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome. Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to 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 classification problem

### 2.2.2. Performance Metric

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation> (<https://www.kaggle.com/c/msk-redefining-cancer-treatment#evaluation>)

Metric(s):

- Multi class log-loss
- Confusion matrix

### 2.2.3. Machine Learning 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 probabilities => Metric is Log-loss. \* No Latency constraints.

### 2.3. Train, CV and Test Datasets

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

## 3. Exploratory Data Analysis

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import re
4 import time
5 import warnings
6 import numpy as np
7 from nltk.corpus import stopwords
8 from sklearn.decomposition import TruncatedSVD
9 from sklearn.preprocessing import normalize
10 from sklearn.feature_extraction.text import CountVectorizer
11 from sklearn.manifold import TSNE
12 import seaborn as sns
13 from sklearn.neighbors import KNeighborsClassifier
14 from sklearn.metrics import confusion_matrix
15 from sklearn.metrics.classification import accuracy_score, log_loss
16 from sklearn.feature_extraction.text import TfidfVectorizer
17 from sklearn.linear_model import SGDClassifier
18 from imblearn.over_sampling import SMOTE
19 from collections import Counter
20 from scipy.sparse import hstack
21 from sklearn.multiclass import OneVsRestClassifier
22 from sklearn.svm import SVC
23 from sklearn.model_selection import StratifiedKFold
24 from collections import Counter, defaultdict
25 from sklearn.calibration import CalibratedClassifierCV
26 from sklearn.naive_bayes import MultinomialNB
27 from sklearn.naive_bayes import GaussianNB
28 from sklearn.model_selection import train_test_split
29 from sklearn.model_selection import GridSearchCV
30 import math
31 from sklearn.metrics import normalized_mutual_info_score
32 from sklearn.ensemble import RandomForestClassifier
33 warnings.filterwarnings("ignore")
34
```

```

35 from mlxtend.classifier import StackingClassifier
36
37 from sklearn import model_selection
38 from sklearn.linear_model import LogisticRegression
39

```

## 3.1. Reading Data

### 3.1.1. Reading Gene and Variation Data

```

1 data = pd.read_csv('training/training_variants')
2 print('Number of data points : ', data.shape[0])
3 print('Number of features : ', data.shape[1])
4 print('Features : ', data.columns.values)
5 data.head()

```

Number of data points : 3321

Number of features : 4

Features : ['ID' 'Gene' 'Variation' 'Class']

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

training/training\_variants is a comma separated file containing the description of the genetic mutations used for training.

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

```

1 # note the separator in this file
2 data_text = pd.read_csv("training/training_text", sep="\|", engine="python", names=["ID", "TEXT"], skiprows:
3 print('Number of data points : ', data_text.shape[0])
4 print('Number of features : ', data_text.shape[1])
5 print('Features : ', data_text.columns.values)
6 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...

```

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

```

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...
2	2	CBL	Q249E	2	Abstract Background Non-small cell lung canc...
3	3	CBL	N454D	3	Recent evidence has demonstrated that acquired...
4	4	CBL	L399V	4	Oncogenic mutations in the monomeric Casitas B...

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

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

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

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

	ID	Gene	Variation	Class	TEXT
<b>1109</b>	1109	FANCA	S1088F	1	FANCA S1088F

## Feature Engineering

Feature engineering reference - <https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/> (<https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/>)



```
1  #Variation share
2  result['Variation_Share'] = result.apply(lambda r: sum([1 for w in r['Variation'].split(' ') if w in r[
3
4  #Text word count
5  result["Text_Word_Count"] = result["TEXT"].apply(lambda x: len(x.split()))
6
7  #Text Character count
8  result['Text_Character_Count'] = result['TEXT'].apply(lambda x: len(str(x)))
9
10 #Text average Length
11 result['Text_Avg_length'] = result['Text_Character_Count'] / result['Text_Word_Count']
12
13 #Gene Character count
14 result['Gene_Character_Count'] = result['Gene'].apply(lambda x: len(str(x)))
15
16 #Variation_Character_Count
17 result['Variation_Character_Count'] = result['Variation'].apply(lambda x: len(str(x)))
18
19 #Var word count
20 result["Var_Word_Count"] = result["Variation"].apply(lambda x: len(x.split()))
21
22 #Var average Length
23 result['var_Avg_length'] = result['Variation_Character_Count'] / result['Var_Word_Count']
24
25 #Stopword text count
26 from nltk.corpus import stopwords
27 stop = stopwords.words('english')
28
29 result['stopwordst'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x in stop]))
30
31 #Speacial character count
32 result['hastags'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.startswith('#'))])
33
34 #Numerics count
```

```

35 result['numerics'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.isdigit()]))
36
37 #Uppercase word count
38 result['upper'] = result['TEXT'].apply(lambda x: len([x for x in x.split() if x.isupper()]))
39
40 # Label encoder
41 from sklearn import preprocessing
42 for c in result.columns:
43     if result[c].dtype == 'object':
44         if c == 'Gene':
45             lbl = preprocessing.LabelEncoder()
46             result[c+'_lbl_enc'] = lbl.fit_transform(result[c].values)
47         elif c == 'Variation':
48             lbl = preprocessing.LabelEncoder()
49             result[c+'_lbl_enc'] = lbl.fit_transform(result[c].values)

```

```
1 result.head(2)
```

	ID	Gene	Variation	Class	TEXT	Variation_Share	Text_Word_Count	Text_Character_Count	Text_A
0	0	FAM58A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	1	6089	39765	6.53062
1	1	CBL	W802*	2	Abstract Background Non-small cell lung canc...	1	5722	36831	6.43673

### 3.1.3. Preprocessing of text

```
1  # loading stop words from nltk library
2  stop_words = set(stopwords.words('english'))
3
4
5  def nlp_preprocessing(total_text, index, column):
6      if type(total_text) is not int:
7          string = ""
8          # replace every special char with space
9          total_text = re.sub('[^a-zA-Z0-9\n]', ' ', total_text)
10         # replace multiple spaces with single space
11         total_text = re.sub('\s+', ' ', total_text)
12         # converting all the chars into lower-case.
13         total_text = total_text.lower()
14
15         for word in total_text.split():
16             # if the word is a not a stop word then retain that word from the data
17             if not word in stop_words:
18                 string += word + " "
19
20         data_text[column][index] = string
```

```
1 #text processing stage.
2 start_time = time.clock()
3 for index, row in data_text.iterrows():
4     if type(row['TEXT']) is str:
5         nlp_preprocessing(row['TEXT'], index, 'TEXT')
6     else:
7         print("there is no text description for id:",index)
8 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 : 1660.0839012061915 seconds
```

```
1 #Creating dummies for Gene_lbl_enc & Variation_lbl_enc
2 result = pd.concat([result, pd.get_dummies(result['Gene_lbl_enc']).\
3                     rename(columns=lambda x: 'Gene_lbl_enc_' + str(x))], axis=1)
4
5 result = pd.concat([result, pd.get_dummies(result['Variation_lbl_enc']).\
6                     rename(columns=lambda x: 'Variation_lbl_enc_' + str(x))], axis=1)
```

```
1 result.head(2)
```

Gene	Variation	Class	TEXT	Variation_Share	Text_Word_Count	Text_Character_Count	Text_Avg_length
158A	Truncating Mutations	1	Cyclin-dependent kinases (CDKs) regulate a var...	1	6089	39765	6.530629
-	W802*	2	Abstract Background Non-small cell lung canc...	1	5722	36831	6.436735

```
79 columns
```

### Checking feature importance of only newly created variables using Random Forest

```
1 temp_y=result[['Class']]
2 temp_train=result.drop(['ID', 'Gene', 'Variation', 'Class', 'TEXT', 'Gene_lbl_enc', 'Variation_lbl_enc'],axis=
```

```
1 ## Import the random forest model.
2 from sklearn.ensemble import RandomForestClassifier
3 ## This line instantiates the model.
4 rf = RandomForestClassifier()
5 ## Fit the model on your training data.
6 rf.fit(temp_train, temp_y)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                        max_depth=None, max_features='auto', max_leaf_nodes=None,
                        min_impurity_decrease=0.0, min_impurity_split=None,
                        min_samples_leaf=1, min_samples_split=2,
                        min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                        oob_score=False, random_state=None, verbose=0,
                        warm_start=False)
```

```

1 import pandas as pd
2 feature_importances = pd.DataFrame(rf.feature_importances_,
3                                   index = temp_train.columns,
4                                   columns=['importance']).sort_values('importance',ascending=False)

```

```

1 #Feature importance of newly created variables
2 feature_importances[:15]

```

	importance
stopwordst	0.053990
numerics	0.053875
Text_Word_Count	0.052921
Text_Avg_length	0.051516
upper	0.051413
Text_Character_Count	0.049281
var_Avg_length	0.030022
Variation_Character_Count	0.028296
Gene_Character_Count	0.023299
Variation_Share	0.018827
Gene_lbl_enc_31	0.015835
hastags	0.015086
Gene_lbl_enc_252	0.013824
Gene_lbl_enc_196	0.013398
Variation_lbl_enc_2629	0.012533

### 3.1.4. Test, Train and Cross Validation Split

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

```
1 y_true = result['Class'].values
2 result.Gene      = result.Gene.str.replace('\s+', '_')
3 result.Variation = result.Variation.str.replace('\s+', '_')
4
5 # split the data into test and train by maintaining same distribution of output variable 'y_true' [stratified]
6 X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
7 # split the train data into train and cross validation by maintaining same distribution of output variable
8 train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train, test_size=0.2)
```

We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

```
1 print('Number of data points in train data:', train_df.shape[0])
2 print('Number of data points in test data:', test_df.shape[0])
3 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

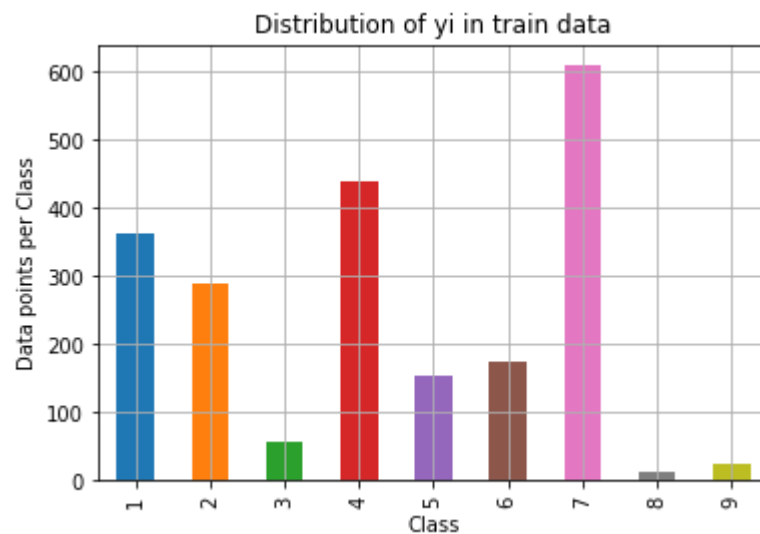
```
1  # it returns a dict, keys as class labels and values as the number of data points in that class
2  train_class_distribution = train_df['Class'].value_counts().sortlevel()
3  test_class_distribution = test_df['Class'].value_counts().sortlevel()
4  cv_class_distribution = cv_df['Class'].value_counts().sortlevel()
5
6  my_colors = 'rgbkymc'
7  train_class_distribution.plot(kind='bar')
8  plt.xlabel('Class')
9  plt.ylabel('Data points per Class')
10 plt.title('Distribution of yi in train data')
11 plt.grid()
12 plt.show()
13
14 # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
15 # -(train_class_distribution.values): the minus sign will give us in decreasing order
16 sorted_yi = np.argsort(-train_class_distribution.values)
17 for i in sorted_yi:
18     print('Number of data points in class', i+1, ':', train_class_distribution.values[i], '(', np.round(
19
20
21 print('-'*80)
22 my_colors = 'rgbkymc'
23 test_class_distribution.plot(kind='bar')
24 plt.xlabel('Class')
25 plt.ylabel('Data points per Class')
26 plt.title('Distribution of yi in test data')
27 plt.grid()
28 plt.show()
29
30 # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
31 # -(train_class_distribution.values): the minus sign will give us in decreasing order
32 sorted_yi = np.argsort(-test_class_distribution.values)
33 for i in sorted_yi:
34     print('Number of data points in class', i+1, ':', test_class_distribution.values[i], '(', np.round(
```



```

35
36 print('-'*80)
37 my_colors = 'rgbkymc'
38 cv_class_distribution.plot(kind='bar')
39 plt.xlabel('Class')
40 plt.ylabel('Data points per Class')
41 plt.title('Distribution of yi in cross validation data')
42 plt.grid()
43 plt.show()
44
45 # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
46 # -(train_class_distribution.values): the minus sign will give us in decreasing order
47 sorted_yi = np.argsort(-train_class_distribution.values)
48 for i in sorted_yi:
49     print('Number of data points in class', i+1, ': ', cv_class_distribution.values[i], '(', np.round((cv_
50

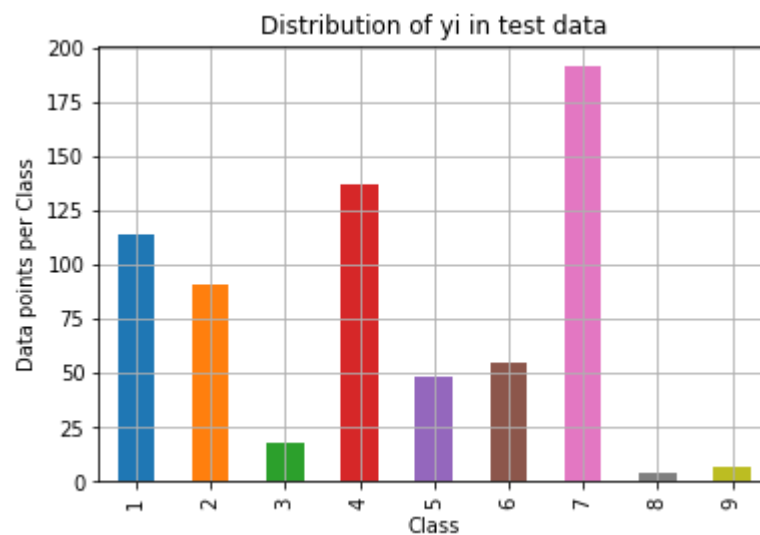
```



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 %)

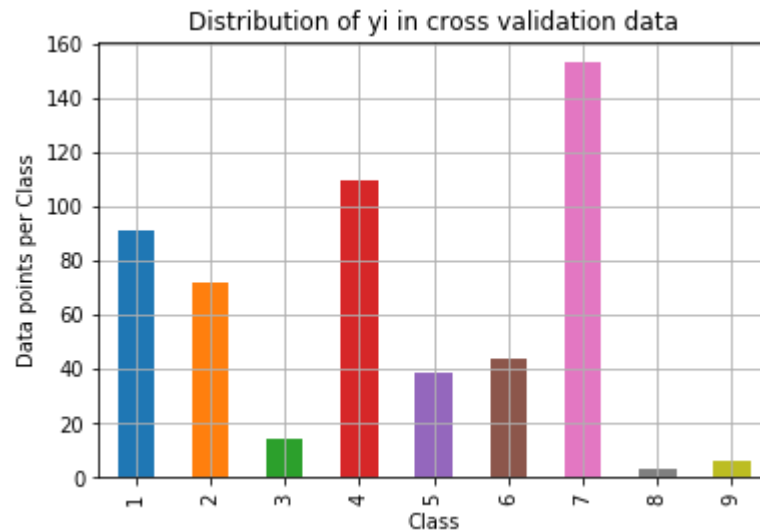
Number of data points in class 6 : 176 ( 8.286 %)  
Number of data points in class 5 : 155 ( 7.298 %)  
Number of data points in class 3 : 57 ( 2.684 %)  
Number of data points in class 9 : 24 ( 1.13 %)  
Number of data points in class 8 : 12 ( 0.565 %)

---



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 probabilities randomly such that they sum to 1.

```

1  # This function plots the confusion matrices given y_i, y_i_hat.
2  def plot_confusion_matrix(test_y, predict_y):
3      C = confusion_matrix(test_y, predict_y)
4      # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
5
6      A = (((C.T)/(C.sum(axis=1))).T)
7      #divid each element of the confusion matrix with the sum of elements in that column
8
9      # C = [[1, 2],
10     #      [3, 4]]
11     # C.T = [[1, 3],
12     #        [2, 4]]
13     # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two dimensional
14     # C.sum(axix =1) = [[3, 7]]
15     # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
16     #                             [2/3, 4/7]]
17
18     # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
19     #                             [3/7, 4/7]]
20     # sum of row elements = 1
21
22     B = (C/C.sum(axis=0))
23     #divid each element of the confusion matrix with the sum of elements in that row
24     # C = [[1, 2],
25     #      [3, 4]]
26     # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two dimensional
27     # C.sum(axix =0) = [[4, 6]]
28     # (C/C.sum(axis=0)) = [[1/4, 2/6],
29     #                       [3/4, 4/6]]
30
31     labels = [1,2,3,4,5,6,7,8,9]
32     # representing A in heatmap format
33     print("-"*20, "Confusion matrix", "-"*20)
34     plt.figure(figsize=(20,7))

```

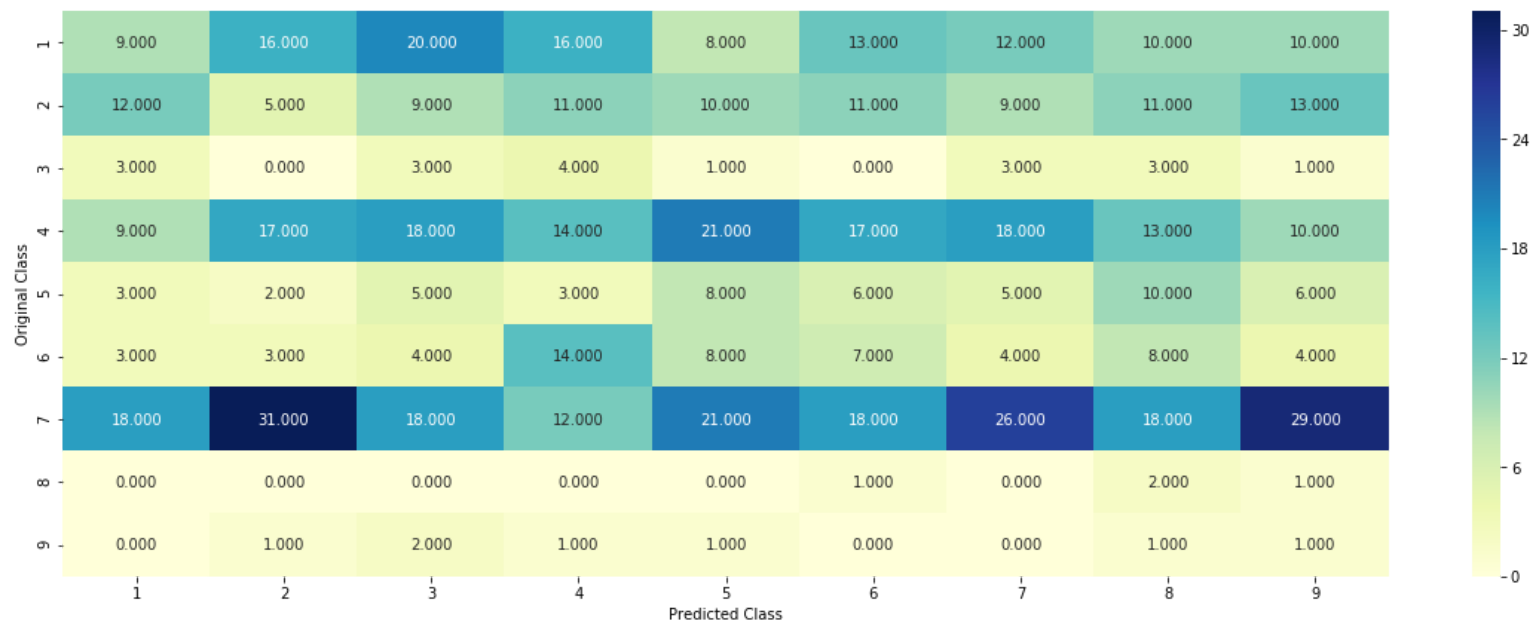
```
35 sns.heatmap(C, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
36 plt.xlabel('Predicted Class')
37 plt.ylabel('Original Class')
38 plt.show()
39
40 print("-"*20, "Precision matrix (Column Sum=1)", "-"*20)
41 plt.figure(figsize=(20,7))
42 sns.heatmap(B, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
43 plt.xlabel('Predicted Class')
44 plt.ylabel('Original Class')
45 plt.show()
46
47 # representing B in heatmap format
48 print("-"*20, "Recall matrix (Row sum=1)", "-"*20)
49 plt.figure(figsize=(20,7))
50 sns.heatmap(A, annot=True, cmap="YlGnBu", fmt=".3f", xticklabels=labels, yticklabels=labels)
51 plt.xlabel('Predicted Class')
52 plt.ylabel('Original Class')
53 plt.show()
```

```
1  # we need to generate 9 numbers and the sum of numbers should be 1
2  # one solution is to generate 9 numbers and divide each of the numbers by their sum
3  # ref: https://stackoverflow.com/a/18662466/4084039
4  test_data_len = test_df.shape[0]
5  cv_data_len = cv_df.shape[0]
6
7  # we create a output array that has exactly same size as the CV data
8  cv_predicted_y = np.zeros((cv_data_len,9))
9  for i in range(cv_data_len):
10     rand_probs = np.random.rand(1,9)
11     cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
12  print("Log loss on Cross Validation Data using Random Model",log_loss(y_cv,cv_predicted_y, eps=1e-15))
13
14
15  # Test-Set error.
16  #we create a output array that has exactly same as the test data
17  test_predicted_y = np.zeros((test_data_len,9))
18  for i in range(test_data_len):
19     rand_probs = np.random.rand(1,9)
20     test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
21  print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))
22
23  predicted_y =np.argmax(test_predicted_y, axis=1)
24  plot_confusion_matrix(y_test, predicted_y+1)
```

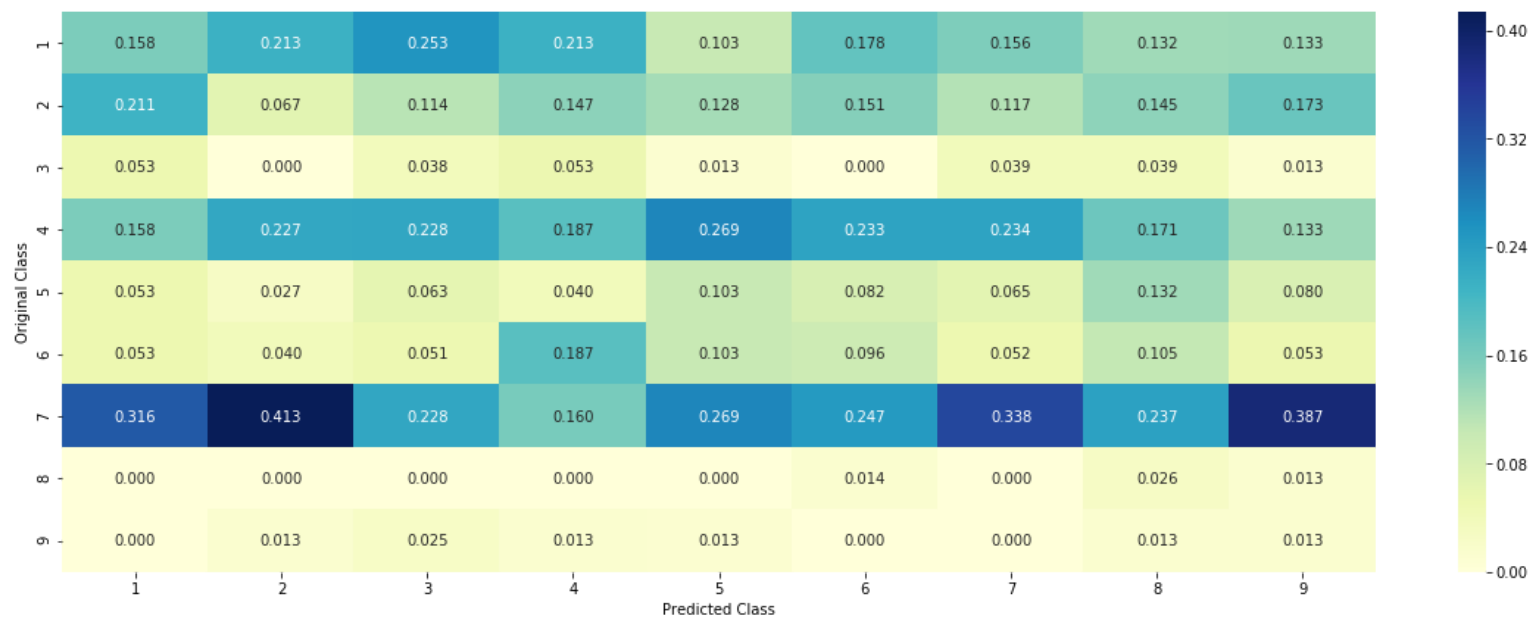
Log loss on Cross Validation Data using Random Model 2.4718508339563896

Log loss on Test Data using Random Model 2.445593547359025

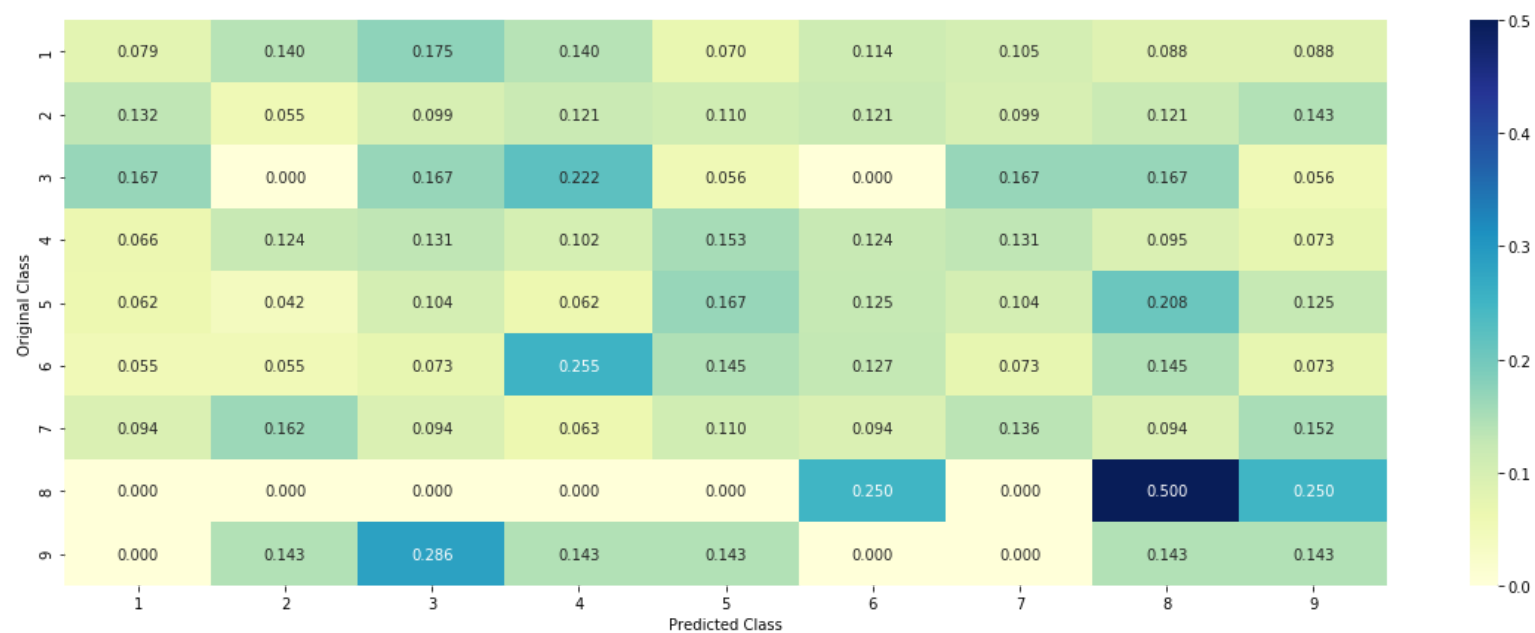
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



### 3.3 Univariate Analysis



```

1  # code for response coding with Laplace smoothing.
2  # alpha : used for laplace smoothing
3  # feature: ['gene', 'variation']
4  # df: ['train_df', 'test_df', 'cv_df']
5  # algorithm
6  # -----
7  # Consider all unique values and the number of occurrences of given feature in train data dataframe
8  # build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number of
9  # gv_dict is like a look up table, for every gene it store a (1*9) representation of it
10 # for a value of feature in df:
11 # if it is in train data:
12 # we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
13 # if it is not there is train:
14 # we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
15 # return 'gv_fea'
16 # -----
17
18 # get_gv_fea_dict: Get Gene variation Feature Dict
19 def get_gv_fea_dict(alpha, feature, df):
20     # value_count: it contains a dict like
21     # print(train_df['Gene'].value_counts())
22     # output:
23     #      {BRCA1      174
24     #      TP53       106
25     #      EGFR        86
26     #      BRCA2       75
27     #      PTEN        69
28     #      KIT         61
29     #      BRAF         60
30     #      ERBB2        47
31     #      PDGFRA       46
32     #      ...}
33     # print(train_df['Variation'].value_counts())
34     # output:

```

```

35 # {
36 #   Truncating_Mutations          63
37 #   Deletion                      43
38 #   Amplification                 43
39 #   Fusions                      22
40 #   Overexpression                3
41 #   E17K                         3
42 #   Q61L                         3
43 #   S222D                        2
44 #   P130S                        2
45 # ...
46 # }
47 value_count = train_df[feature].value_counts()
48
49 # gv_dict : Gene Variation Dict, which contains the probability array for each gene/variation
50 gv_dict = dict()
51
52 # denominator will contain the number of time that particular feature occurred in whole data
53 for i, denominator in value_count.items():
54     # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular class
55     # vec is 9 dimensional vector
56     vec = []
57     for k in range(1,10):
58         # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')])
59         #           ID   Gene          Variation   Class
60         # 2470  2470  BRCA1          S1715C        1
61         # 2486  2486  BRCA1          S1841R        1
62         # 2614  2614  BRCA1           M1R         1
63         # 2432  2432  BRCA1          L1657P        1
64         # 2567  2567  BRCA1          T1685A        1
65         # 2583  2583  BRCA1          E1660G        1
66         # 2634  2634  BRCA1          W1718L        1
67         # cls_cnt.shape[0] will return the number of rows
68
69         cls_cnt = train_df.loc[(train_df['Class']==k) & (train_df[feature]==i)]

```

```

70
71     # cls_cnt.shape[0](numerator) will contain the number of time that particular feature occurs
72     vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))
73
74     # we are adding the gene/variation to the dict as key and vec as value
75     gv_dict[i]=vec
76     return gv_dict
77
78 # Get Gene variation feature
79 def get_gv_feature(alpha, feature, df):
80     # print(gv_dict)
81     #     {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.0681818181818177, 0.13636363636363636],
82     #     'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704081632653061],
83     #     'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.0681818181818177, 0.0681818181818177],
84     #     'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.07878787878787878],
85     #     'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465408805031446],
86     #     'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.072847682119205295],
87     #     'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.07333333333333333],
88     #     ...
89     # }
90     gv_dict = get_gv_fea_dict(alpha, feature, df)
91     # value_count is similar in get_gv_fea_dict
92     value_count = train_df[feature].value_counts()
93
94     # gv_fea: Gene_variation feature, it will contain the feature for each feature value in the data
95     gv_fea = []
96     # for every feature values in the given data frame we will check if it is there in the train data table
97     # 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
98     for index, row in df.iterrows():
99         if row[feature] in dict(value_count).keys():
100             gv_fea.append(gv_dict[row[feature]])
101         else:
102             gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
103     #     gv_fea.append([-1, -1, -1, -1, -1, -1, -1, -1, -1])
104     return gv_fea

```

when we calculate the probability of a feature belongs to any particular class, we apply laplace smoothing

- $(\text{numerator} + 10 \cdot \alpha) / (\text{denominator} + 90 \cdot \alpha)$

### 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?

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

Number of Unique Genes : 230

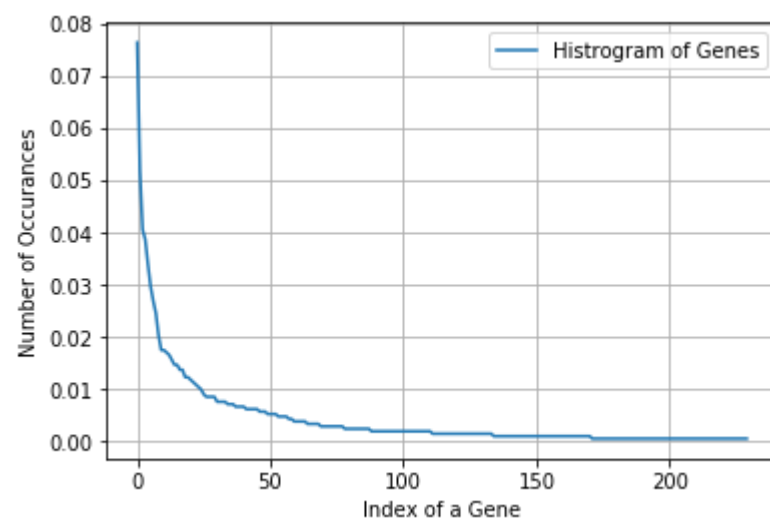
BRCA1	162
TP53	110
EGFR	86
PTEN	82
BRCA2	72
KIT	63
BRAF	57
ERBB2	52
PDGFRA	43
PIK3CA	37

Name: Gene, dtype: int64

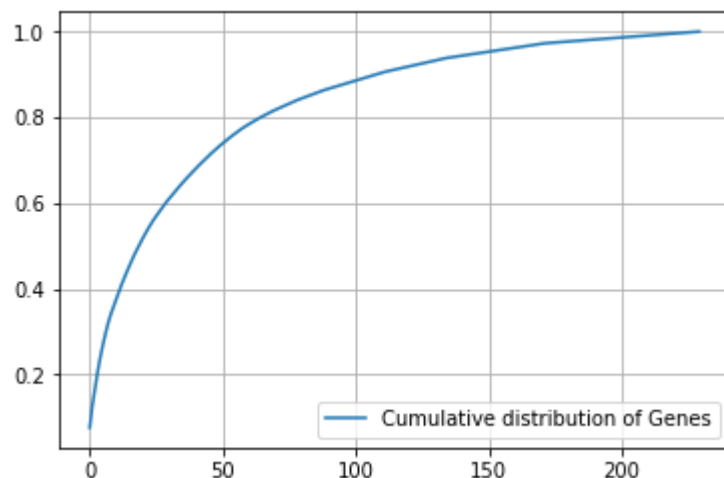
```
1 print("Ans: There are", unique_genes.shape[0], "different categories of genes in the train data, and the
```

Ans: There are 230 different categories of genes in the train data, and they are distributed as follows

```
1 s = sum(unique_genes.values);  
2 h = unique_genes.values/s;  
3 plt.plot(h, label="Histogram of Genes")  
4 plt.xlabel('Index of a Gene')  
5 plt.ylabel('Number of Occurances')  
6 plt.legend()  
7 plt.grid()  
8 plt.show()  
9
```



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



### Q3. How to featurize this Gene feature ?

**Ans.**there are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-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 multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

```
1 #response-coding of the Gene feature
2 # alpha is used for Laplace smoothing
3 alpha = 1
4 # train gene feature
5 train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
6 # test gene feature
7 test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
8 # cross validation gene feature
9 cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
1 print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of
```

```
train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature: (2124, 9)
```

```
1 # one-hot encoding of Gene feature.
2 gene_vectorizer = TfidfVectorizer(max_features=2000)
3 train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
4 test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
5 cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

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

```
2975      KIT
2144    KEAP1
348     CDH1
2525    BRCA1
638    CDKN1B
Name: Gene, dtype: object
```

```
1 gene_vectorizer.get_feature_names()  
  
['abl1',  
 'acvr1',  
 'ago2',  
 'akt1',  
 'akt2',  
 'akt3',  
 'alk',  
 'apc',  
 'ar',  
 'araf',  
 'arid1b',  
 'arid2',  
 'arid5b',  
 'asx11',  
 'atm',  
 'atr',  
 'atrx',  
 'aurka',  
 'aurkb',  
 'b2m',  
 'bap1',
```

```
1 print("train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of  
  
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 238)
```

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

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



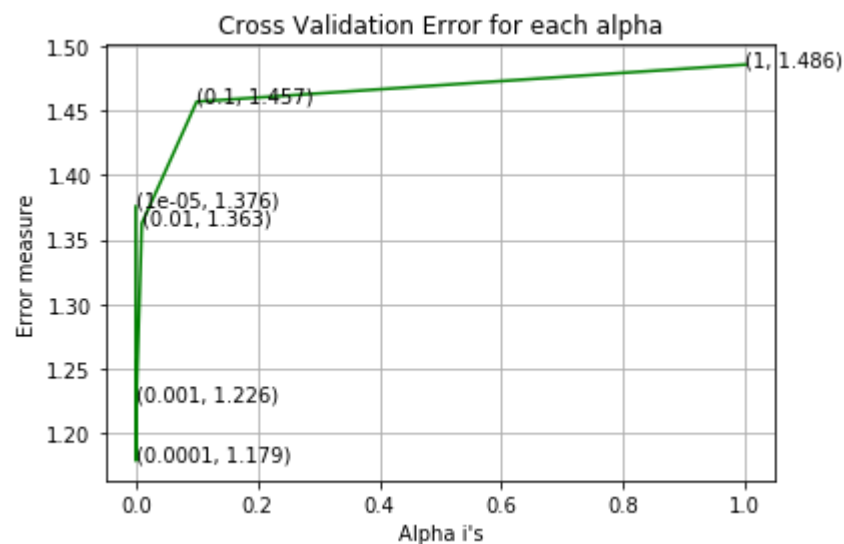
```

1  alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.
2
3  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
4  # -----
5  # default parameters
6  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
7  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
8  # class_weight=None, warm_start=False, average=False, n_iter=None)
9
10 # some of methods
11 # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
12 # predict(X) Predict class labels for samples in X.
13
14 #-----
15 # video link:
16 #-----
17
18
19 cv_log_error_array=[]
20 for i in alpha:
21     clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
22     clf.fit(train_gene_feature_onehotCoding, y_train)
23     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
24     sig_clf.fit(train_gene_feature_onehotCoding, y_train)
25     predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
26     cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
27     print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_,
28
29 fig, ax = plt.subplots()
30 ax.plot(alpha, cv_log_error_array, c='g')
31 for i, txt in enumerate(np.round(cv_log_error_array, 3)):
32     ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
33 plt.grid()
34 plt.title("Cross Validation Error for each alpha")

```

```
35 plt.xlabel("Alpha i's")
36 plt.ylabel("Error measure")
37 plt.show()
38
39
40 best_alpha = np.argmin(cv_log_error_array)
41 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
42 clf.fit(train_gene_feature_onehotCoding, y_train)
43 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
44 sig_clf.fit(train_gene_feature_onehotCoding, y_train)
45
46 predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
47 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
48 predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
49 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_train, predict_y))
50 predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
51 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
52
```

```
For values of alpha = 1e-05 The log loss is: 1.3756874785823499
For values of alpha = 0.0001 The log loss is: 1.178970369093293
For values of alpha = 0.001 The log loss is: 1.2258660995375157
For values of alpha = 0.01 The log loss is: 1.3625757602340773
For values of alpha = 0.1 The log loss is: 1.4568409042356278
For values of alpha = 1 The log loss is: 1.4855086547288696
```



For values of best alpha = 0.0001 The train log loss is: 1.0393883822582672

For values of best alpha = 0.0001 The cross validation log loss is: 1.178970369093293

For values of best alpha = 0.0001 The test log loss is: 1.193353061879979

**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.

```

1 print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " ")
2
3 test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
4 cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
5
6 print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/test_df.shape
7 print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage/cv_df.shape

```

Q6. How many data points in Test and CV datasets are covered by the 230 genes in train dataset?

Ans

1. In test data 643 out of 665 : 96.69172932330827

2. In cross validation data 511 out of 532 : 96.05263157894737

### 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?

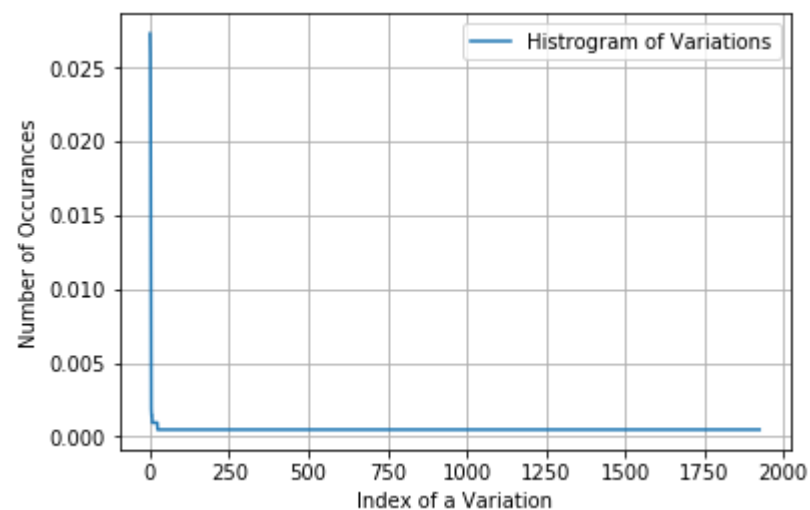
```
1 unique_variations = train_df['Variation'].value_counts()
2 print('Number of Unique Variations :', unique_variations.shape[0])
3 # the top 10 variations that occurred most
4 print(unique_variations.head(10))
```

```
Number of Unique Variations : 1926
Truncating_Mutations    58
Deletion                 55
Amplification            44
Fusions                  21
Overexpression           4
T58I                     3
Q61H                     3
P130S                    2
R170W                    2
F384L                    2
Name: Variation, dtype: int64
```

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

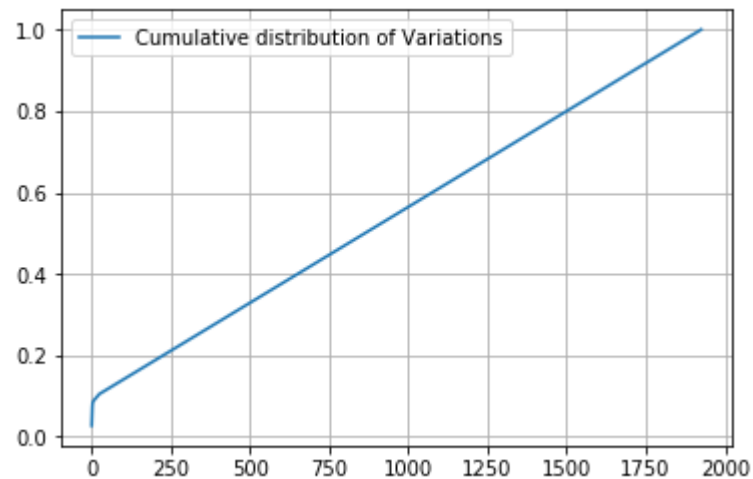
Ans: There are 1926 different categories of variations in the train data, and they are distributed as follows

```
1 s = sum(unique_variations.values);  
2 h = unique_variations.values/s;  
3 plt.plot(h, label="Histogram of Variations")  
4 plt.xlabel('Index of a Variation')  
5 plt.ylabel('Number of Occurances')  
6 plt.legend()  
7 plt.grid()  
8 plt.show()
```



```
1 c = np.cumsum(h)
2 print(c)
3 plt.plot(c,label='Cumulative distribution of Variations')
4 plt.grid()
5 plt.legend()
6 plt.show()
```

```
[0.02730697 0.05320151 0.07391714 ... 0.99905838 0.99952919 1. ]
```



**Q9.** How to featurize this Variation feature ?

**Ans.** There are two ways we can featurize this variable check out this video:

<https://www.appliedaicourse.com/course/applied-ai-course-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

```
1 # alpha is used for Laplace smoothing
2 alpha = 1
3 # train gene feature
4 train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", train_df))
5 # test gene feature
6 test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", test_df))
7 # cross validation gene feature
8 cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_df))
```

```
1 print("train_variation_feature_responseCoding is a converted feature using the response coding method. "
```

train\_variation\_feature\_responseCoding is a converted feature using the response coding method. The shape of Variation feature: (2124, 9)

```
1 # one-hot encoding of variation feature.
2 variation_vectorizer = TfidfVectorizer(max_features=2000)
3 train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
4 test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
5 cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
1 print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. Tl
```

train\_variation\_feature\_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 1960)

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

Let's build a model just like the earlier!

```

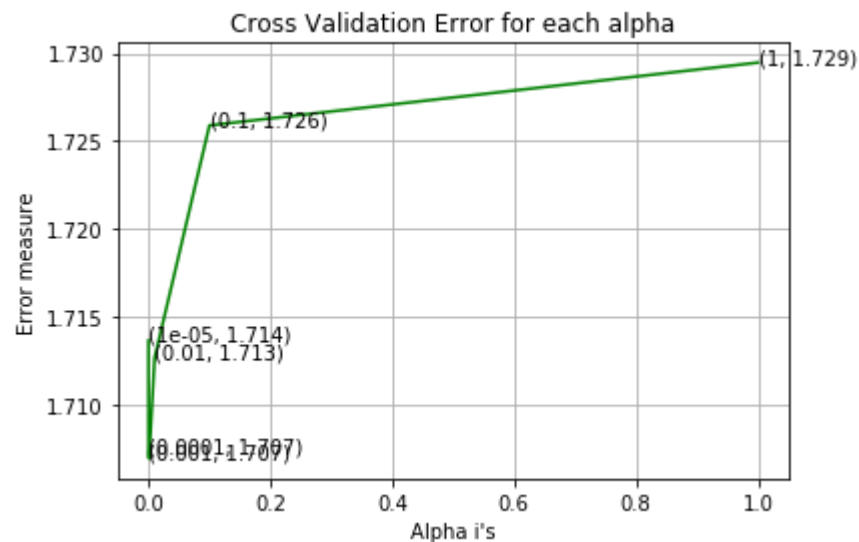
1  alpha = [10 ** x for x in range(-5, 1)]
2
3  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
4  # -----
5  # default parameters
6  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
7  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
8  # class_weight=None, warm_start=False, average=False, n_iter=None)
9
10 # some of methods
11 # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
12 # predict(X) Predict class labels for samples in X.
13
14 #-----
15 # video link:
16 #-----
17
18
19 cv_log_error_array=[]
20 for i in alpha:
21     clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
22     clf.fit(train_variation_feature_onehotCoding, y_train)
23
24     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
25     sig_clf.fit(train_variation_feature_onehotCoding, y_train)
26     predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
27
28     cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
29     print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_,
30
31 fig, ax = plt.subplots()
32 ax.plot(alpha, cv_log_error_array, c='g')
33 for i, txt in enumerate(np.round(cv_log_error_array, 3)):
34     ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))

```



```
35 plt.grid()
36 plt.title("Cross Validation Error for each alpha")
37 plt.xlabel("Alpha i's")
38 plt.ylabel("Error measure")
39 plt.show()
40
41
42 best_alpha = np.argmin(cv_log_error_array)
43 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
44 clf.fit(train_variation_feature_onehotCoding, y_train)
45 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
46 sig_clf.fit(train_variation_feature_onehotCoding, y_train)
47
48 predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
49 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
50 predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
51 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_train, predict_y))
52 predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
53 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
54
```

```
For values of alpha = 1e-05 The log loss is: 1.713634477973893
For values of alpha = 0.0001 The log loss is: 1.7072255111504657
For values of alpha = 0.001 The log loss is: 1.7069042678586714
For values of alpha = 0.01 The log loss is: 1.7126361998090043
For values of alpha = 0.1 The log loss is: 1.7258891461289891
For values of alpha = 1 The log loss is: 1.729466992933819
```



For values of best alpha = 0.001 The train log loss is: 1.083433098305849

For values of best alpha = 0.001 The cross validation log loss is: 1.7069042678586714

For values of best alpha = 0.001 The test log loss is: 1.7286928922936888

**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.

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

Q12. How many data points are covered by total 1926 genes in test and cross validation data sets?

Ans

1. In test data 65 out of 665 : 9.774436090225564

2. In cross validation data 57 out of 532 : 10.714285714285714

### 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 predicting  $y_i$ ?
5. Is the text feature stable across train, test and CV datasets?

```
1  # cls_text is a data frame
2  # for every row in data fram consider the 'TEXT'
3  # split the words by space
4  # make a dict with those words
5  # increment its count whenever we see that word
6
7  def extract_dictionary_paddle(cls_text):
8      dictionary = defaultdict(int)
9      for index, row in cls_text.iterrows():
10         for word in row['TEXT'].split():
11             dictionary[word] +=1
12     return dictionary
```

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

```
1 # building a CountVectorizer with all the words that occurred minimum 3 times in train data
2 text_vectorizer = TfidfVectorizer(max_features=2000,min_df=3,ngram_range=(1,4))
3 train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
4 # getting all the feature names (words)
5 train_text_features= text_vectorizer.get_feature_names()
6
7 # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*number of features)
8 train_text_fea_counts = train_text_feature_onehotCoding.sum(axis=0).A1
9
10 # zip(list(text_features),text_fea_counts) will zip a word with its number of times it occurred
11 text_fea_dict = dict(zip(list(train_text_features),train_text_fea_counts))
12
13
14 print("Total number of unique words in train data :", len(train_text_features))
```

Total number of unique words in train data : 2000

```
1 # Collecting all the genes and variations in a single list
2 corpus = []
3 for word in data['Gene'].values:
4     corpus.append(word)
5 for word in data['Variation'].values:
6     corpus.append(word)
```

```
1  # Training TfidfVectorizer on the 'corpus' list
2  text1 = TfidfVectorizer()
3  text2 = text1.fit_transform(corpus)
4  text1_features = text1.get_feature_names()
5
6  # Transforming the train_df['TEXT']
7  train_text = text1.transform(train_df['TEXT'])
8
9  # Transforming the test_df['TEXT']
10 test_text = text1.transform(test_df['TEXT'])
11
12 # Transforming the cv_df['TEXT']
13 cv_text = text1.transform(cv_df['TEXT'])
14
15 # Normalizing the train_text
16 train_text = normalize(train_text,axis=0)
17
18 # Normalizing the test_text
19 test_text = normalize(test_text,axis=0)
20
21 # Normalizing the cv_text
22 cv_text = normalize(cv_text,axis=0)
```

```
1 dict_list = []
2 # dict_list =[] contains 9 dictionaries each corresponds to a class
3 for i in range(1,10):
4     cls_text = train_df[train_df['Class']==i]
5     # build a word dict based on the words in that class
6     dict_list.append(extract_dictionary_paddle(cls_text))
7     # append it to dict_list
8
9 # dict_list[i] is build on i'th class text data
10 # total_dict is build on whole training text data
11 total_dict = extract_dictionary_paddle(train_df)
12
13
14 confuse_array = []
15 for i in train_text_features:
16     ratios = []
17     max_val = -1
18     for j in range(0,9):
19         ratios.append((dict_list[j][i]+10)/(total_dict[i]+90))
20     confuse_array.append(ratios)
21 confuse_array = np.array(confuse_array)
```

```
1 #response coding of text features
2 train_text_feature_responseCoding = get_text_responsecoding(train_df)
3 test_text_feature_responseCoding = get_text_responsecoding(test_df)
4 cv_text_feature_responseCoding = get_text_responsecoding(cv_df)
```

```
1 # https://stackoverflow.com/a/16202486
2 # we convert each row values such that they sum to 1
3 train_text_feature_responseCoding = (train_text_feature_responseCoding.T/train_text_feature_responseCoding.sum(axis=1)).T
4 test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.sum(axis=1)).T
5 cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.sum(axis=1)).T
```

```
1  # don't forget to normalize every feature
2  train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)
3
4  # we use the same vectorizer that was trained on train data
5  test_text_feature_onehotCoding = text_vectorizer.transform(test_df['TEXT'])
6  # don't forget to normalize every feature
7  test_text_feature_onehotCoding = normalize(test_text_feature_onehotCoding, axis=0)
8
9  # we use the same vectorizer that was trained on train data
10 cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
11 # don't forget to normalize every feature
12 cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

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

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

```
Counter({7.582913059697105: 2, 6.954502361233818: 2, 6.92884441057595: 2, 6.491825765471072: 2, 6.0269280272039705: 2, 6.0  
26865511109753: 2, 6.007257168264756: 2, 5.986224038400959: 2, 5.825896903538086: 2, 5.443757179165208: 2, 5.4018880141161  
8: 2, 5.033556051640021: 2, 838.9649148250226: 1, 689.5113976653439: 1, 524.5996572694528: 1, 513.3692049544735: 1, 312.46  
832754691997: 1, 216.0256820799255: 1, 161.38602157952326: 1, 155.38883428781867: 1, 150.18938696593236: 1, 148.2833122823  
3858: 1, 138.4186237421294: 1, 120.77048844508948: 1, 117.03019427894581: 1, 116.65734221565393: 1, 109.51136717704541: 1,  
105.40123045769566: 1, 98.68518932283726: 1, 93.81745371298084: 1, 83.02446960308896: 1, 82.03727718043083: 1, 77.69727608  
529624: 1, 74.90636196843934: 1, 74.38387685963873: 1, 73.97414655771198: 1, 73.01751095519587: 1, 70.1350981156944: 1, 7  
0.05441442422553: 1, 69.9828569131422: 1, 67.12782151945035: 1, 65.00051008304725: 1, 64.74629384888914: 1, 63.69993270861  
112: 1, 62.8363410809882: 1, 60.04879476590562: 1, 59.74054896125315: 1, 59.16890981232979: 1, 54.21293670277593: 1, 53.82  
741547401608: 1, 53.525371999338404: 1, 51.84427947717972: 1, 51.276782992338966: 1, 50.26709648865007: 1, 48.610254700544  
594: 1, 47.53966959058998: 1, 47.374792791656276: 1, 46.80491448972698: 1, 45.35495022701856: 1, 45.07044922763085: 1, 44.  
22014754439242: 1, 42.90640259302393: 1, 42.58664603456663: 1, 40.94474833475612: 1, 40.83500488494298: 1, 39.733384298906  
04: 1, 37.97514298339917: 1, 37.93892126368196: 1, 37.88257373629638: 1, 36.79219775202431: 1, 36.67525577468372: 1, 36.47  
317104964013: 1, 36.19732431869097: 1, 36.191042395800025: 1, 35.70438529973093: 1, 35.70011267759757: 1, 35.2966905039620  
3: 1, 34.90510958456764: 1, 34.771071859072066: 1, 33.72259851336872: 1, 33.36742081178468: 1, 32.781759156197666: 1, 31.9  
21977558633927: 1, 31.26636538981772: 1, 31.1928246388085: 1, 31.105440227617162: 1, 30.82055574647326: 1, 30.604372178725  
1: 1, 29.80005223795802: 1, 29.2911149916442: 1, 29.193641433238916: 1, 29.15922653447494: 1, 29.088881556813554: 1, 28.92  
550740510566: 1, 28.444387905122454: 1, 28.43567491330705: 1, 28.33135216144406: 1, 28.14985173581848: 1, 27.9993690741402  
97: 1, 27.927348935235806: 1, 27.767335746363536: 1, 26.571693550367872: 1, 26.053865108624347: 1, 25.952065774894177: 1,  
25.951698587394393: 1, 25.938198674798006: 1, 25.795877674035065: 1, 25.716789345594744: 1, 25.62364220289779: 1, 25.45472  
549743893: 1, 25.328589227412056: 1, 25.134299313940453: 1, 24.871636707648953: 1, 24.866898052428702: 1, 24.2578352484237
```



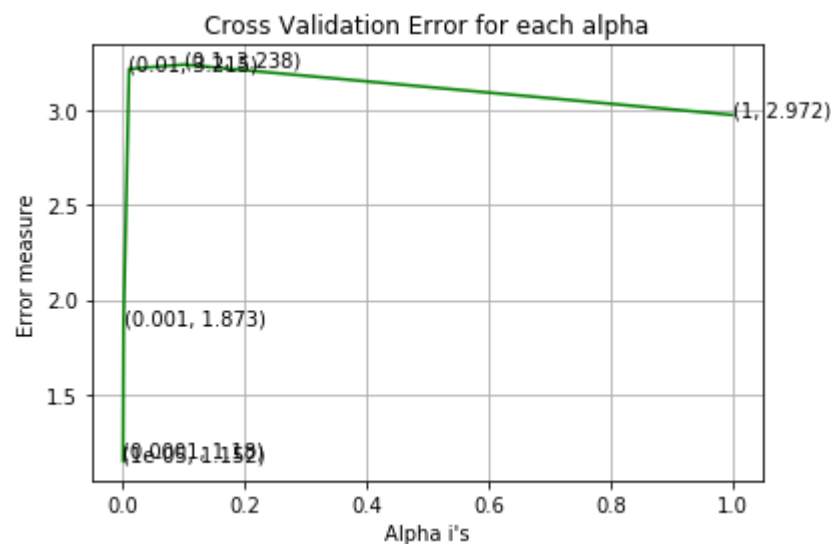
```

1  # Train a Logistic regression+Calibration model using text features which are on-hot encoded
2  alpha = [10 ** x for x in range(-5, 1)]
3
4  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model
5  # -----
6  # default parameters
7  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
8  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
9  # class_weight=None, warm_start=False, average=False, n_iter=None)
10
11 # some of methods
12 # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
13 # predict(X) Predict class labels for samples in X.
14
15 #-----
16 # video link:
17 #-----
18
19
20 cv_log_error_array=[]
21 for i in alpha:
22     clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
23     clf.fit(train_text_feature_onehotCoding, y_train)
24
25     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
26     sig_clf.fit(train_text_feature_onehotCoding, y_train)
27     predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
28     cv_log_error_array.append(log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
29     print('For values of alpha = ', i, "The log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_,
30
31 fig, ax = plt.subplots()
32 ax.plot(alpha, cv_log_error_array, c='g')
33 for i, txt in enumerate(np.round(cv_log_error_array, 3)):
34     ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))

```

```
35 plt.grid()
36 plt.title("Cross Validation Error for each alpha")
37 plt.xlabel("Alpha i's")
38 plt.ylabel("Error measure")
39 plt.show()
40
41
42 best_alpha = np.argmin(cv_log_error_array)
43 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
44 clf.fit(train_text_feature_onehotCoding, y_train)
45 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
46 sig_clf.fit(train_text_feature_onehotCoding, y_train)
47
48 predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
49 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
50 predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
51 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_train, predict_y))
52 predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
53 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
54
```

```
For values of alpha = 1e-05 The log loss is: 1.1524900798435957
For values of alpha = 0.0001 The log loss is: 1.179564490711332
For values of alpha = 0.001 The log loss is: 1.872966144162812
For values of alpha = 0.01 The log loss is: 3.214811679217841
For values of alpha = 0.1 The log loss is: 3.2375632762276934
For values of alpha = 1 The log loss is: 2.972487642596276
```



For values of best alpha = 1e-05 The train log loss is: 0.7686421366571631

For values of best alpha = 1e-05 The cross validation log loss is: 1.1524900798435957

For values of best alpha = 1e-05 The test log loss is: 1.0414197496307163

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

**Ans.** Yes, it seems like!

```

1 def get_intersec_text(df):
2     df_text_vec = TfidfVectorizer(max_features=2000,min_df=3,ngram_range=(1,4))
3     df_text_fea = df_text_vec.fit_transform(df['TEXT'])
4     df_text_features = df_text_vec.get_feature_names()
5
6     df_text_fea_counts = df_text_fea.sum(axis=0).A1
7     df_text_fea_dict = dict(zip(list(df_text_features),df_text_fea_counts))
8     len1 = len(set(df_text_features))
9     len2 = len(set(train_text_features) & set(df_text_features))
10    return len1,len2

```

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

93.45 % of word of test data appeared in train data

92.6 % of word of Cross Validation appeared in train data

## 4. Machine Learning Models

```
1 #Data preparation for ML models.
2
3 #Misc. fonctionns for ML models
4
5
6 def predict_and_plot_confusion_matrix(train_x, train_y, test_x, test_y, clf):
7     clf.fit(train_x, train_y)
8     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
9     sig_clf.fit(train_x, train_y)
10    pred_y = sig_clf.predict(test_x)
11
12    # for calculating log_loss we willl provide the array of probabilities belongs to each class
13    print("Log loss :", log_loss(test_y, sig_clf.predict_proba(test_x)))
14    # calculating the number of data points that are misclassified
15    print("Number of mis-classified points :", np.count_nonzero((pred_y- test_y))/test_y.shape[0])
16    plot_confusion_matrix(test_y, pred_y)
```

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

```
1  # this function will be used just for naive bayes
2  # for the given indices, we will print the name of the features
3  # and we will check whether the feature present in the test point text or not
4  def get_impfeature_names(indices, text, gene, var, no_features):
5      gene_count_vec = TfidfVectorizer(max_features=2000)
6      var_count_vec = TfidfVectorizer(max_features=2000)
7      text_count_vec = TfidfVectorizer(max_features=2000,min_df=3,ngram_range=(1,4))
8
9      gene_vec = gene_count_vec.fit(train_df['Gene'])
10     var_vec = var_count_vec.fit(train_df['Variation'])
11     text_vec = text_count_vec.fit(train_df['TEXT'])
12
13     fea1_len = len(gene_vec.get_feature_names())
14     fea2_len = len(var_count_vec.get_feature_names())
15
16     word_present = 0
17     for i,v in enumerate(indices):
18         if (v < fea1_len):
19             word = gene_vec.get_feature_names()[v]
20             yes_no = True if word == gene else False
21             if yes_no:
22                 word_present += 1
23                 print(i, "Gene feature [{}] present in test data point [{}]" .format(word,yes_no))
24         elif (v < fea1_len+fea2_len):
25             word = var_vec.get_feature_names()[v-(fea1_len)]
26             yes_no = True if word == var else False
27             if yes_no:
28                 word_present += 1
29                 print(i, "variation feature [{}] present in test data point [{}]" .format(word,yes_no))
30         else:
31             word = text_vec.get_feature_names()[v-(fea1_len+fea2_len)]
32             yes_no = True if word in text.split() else False
33             if yes_no:
34                 word_present += 1
```

```

35         print(i, "Text feature [{}] present in test data point [{}].format(word,yes_no))
36
37     print("Out of the top ",no_features," features ", word_present, "are present in query point")

```

Keeping top 15 created variables based on feature importance of Random Forest

```

1  #Keeping separate dataset for newly created feaures
2  train_df_other_feat=train_df[["stopwordst", "numerics", "Text_Word_Count", "Text_Avg_length", "upper".
3  test_df_other_feat=test_df[["stopwordst", "numerics", "Text_Word_Count", "Text_Avg_length", "upper".
4  cv_df_other_feat=cv_df[["stopwordst", "numerics", "Text_Word_Count", "Text_Avg_length", "upper",

```

```

1  train_df_other_feat.head(5)

```

	stopwordst	numerics	Text_Word_Count	Text_Avg_length	upper	Text_Character_Count	var_Avg_leng
<b>2975</b>	3132	542	12050	6.701660	1327	80755	5.0
<b>2144</b>	1322	57	4715	7.002545	545	33017	5.0
<b>348</b>	65	1	187	7.032086	5	1315	5.0
<b>2525</b>	1995	124	6024	6.498672	380	39148	6.0
<b>638</b>	7090	320	23094	6.806920	1477	157199	8.0

```

1  from sklearn.preprocessing import MinMaxScaler
2  sc=MinMaxScaler()
3
4  train_df_other_feat=sc.fit_transform(train_df_other_feat)
5  test_df_other_feat=sc.transform(test_df_other_feat)
6  cv_df_other_feat=sc.transform(cv_df_other_feat)

```

## Stacking the three types of features

```
1  # merging gene, variance and text features
2
3  # building train, test and cross validation data sets
4  # a = [[1, 2],
5  #      [3, 4]]
6  # b = [[4, 5],
7  #      [6, 7]]
8  # hstack(a, b) = [[1, 2, 4, 5],
9  #                 [ 3, 4, 6, 7]]
10
11 train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding, train_variation_feature_onehotCoding))
12 test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding, test_variation_feature_onehotCoding))
13 cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_variation_feature_onehotCoding))
14
15 train_x_onehotCoding = hstack((train_gene_var_onehotCoding, train_text_feature_onehotCoding)).tocsr()
16 train_y = np.array(list(train_df['Class']))
17
18 test_x_onehotCoding = hstack((test_gene_var_onehotCoding, test_text_feature_onehotCoding)).tocsr()
19 test_y = np.array(list(test_df['Class']))
20
21 cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
22 cv_y = np.array(list(cv_df['Class']))
```

```
1 train_x_onehotCoding=hstack((train_x_onehotCoding,train_df_other_feat)).tocsr()
2 test_x_onehotCoding = hstack((test_x_onehotCoding, test_df_other_feat)).tocsr()
3 cv_x_onehotCoding = hstack((cv_x_onehotCoding, cv_df_other_feat)).tocsr()
```

```

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

```

One hot encoding features :

```

(number of data points * number of features) in train data = (2124, 4205)
(number of data points * number of features) in test data = (665, 4205)
(number of data points * number of features) in cross validation data = (532, 4205)

```

```

1 train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding, train_variation_feature_re:
2 test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding, test_variation_feature_respo
3 cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding, cv_variation_feature_responseCod:
4
5 train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_responseCoding))
6 test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_responseCoding))
7 cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_responseCoding))
8
9 train_x_responseCoding=np.hstack((train_x_responseCoding, train_df_other_feat))
10 test_x_responseCoding=np.hstack((test_x_responseCoding, test_df_other_feat))
11 cv_x_responseCoding=np.hstack((cv_x_responseCoding, cv_df_other_feat))

```

```

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

```

Response encoding features :

```

(number of data points * number of features) in train data = (2124, 42)
(number of data points * number of features) in test data = (665, 42)
(number of data points * number of features) in cross validation data = (532, 42)

```

## 4.1. Base Line Model

### 4.1.1. Naive Bayes



#### 4.1.1.1. Hyper parameter tuning

```
1 # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated,
2 # -----
3 # default paramters
4 # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
5
6 # some of methods of MultinomialNB()
7 # fit(X, y[, sample_weight]) Fit Naive Bayes classifier according to X, y
8 # predict(X) Perform classification on an array of test vectors X.
9 # predict_log_proba(X) Return log-probability estimates for the test vector X.
10 # -----
11 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algor
12 # -----
13
14
15 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
16 # -----
17 # default paramters
18 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
19 #
20 # some of the methods of CalibratedClassifierCV()
21 # fit(X, y[, sample_weight]) Fit the calibrated model
22 # get_params([deep]) Get parameters for this estimator.
23 # predict(X) Predict the target of new samples.
24 # predict_proba(X) Posterior probabilities of classification
25 # -----
26 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algor
27 # -----
28
29
30 alpha = [0.00001, 0.0001, 0.001, 0.1, 1, 10, 100,1000]
31 cv_log_error_array = []
32 for i in alpha:
33     print("for alpha =", i)
34     clf = MultinomialNB(alpha=i)
```

```

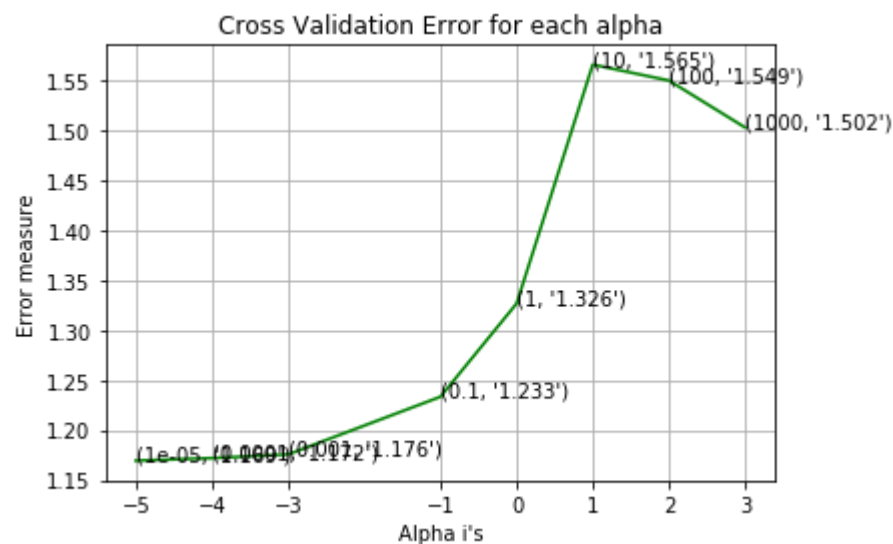
35     clf.fit(train_x_onehotCoding, train_y)
36     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
37     sig_clf.fit(train_x_onehotCoding, train_y)
38     sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
39     cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
40     # to avoid rounding error while multiplying probabilities we use log-probability estimates
41     print("Log Loss :", log_loss(cv_y, sig_clf_probs))
42
43     fig, ax = plt.subplots()
44     ax.plot(np.log10(alpha), cv_log_error_array, c='g')
45     for i, txt in enumerate(np.round(cv_log_error_array, 3)):
46         ax.annotate((alpha[i], str(txt)), (np.log10(alpha[i]), cv_log_error_array[i]))
47     plt.grid()
48     plt.xticks(np.log10(alpha))
49     plt.title("Cross Validation Error for each alpha")
50     plt.xlabel("Alpha i's")
51     plt.ylabel("Error measure")
52     plt.show()
53
54
55     best_alpha = np.argmin(cv_log_error_array)
56     clf = MultinomialNB(alpha=alpha[best_alpha])
57     clf.fit(train_x_onehotCoding, train_y)
58     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
59     sig_clf.fit(train_x_onehotCoding, train_y)
60
61
62     predict_y = sig_clf.predict_proba(train_x_onehotCoding)
63     print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
64     predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
65     print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
66     predict_y = sig_clf.predict_proba(test_x_onehotCoding)
67     print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
68

```

```

for alpha = 1e-05
Log Loss : 1.1694235780352655
for alpha = 0.0001
Log Loss : 1.1719652749721405
for alpha = 0.001
Log Loss : 1.1756255475434816
for alpha = 0.1
Log Loss : 1.2332514647526944
for alpha = 1
Log Loss : 1.3264583926878253
for alpha = 10
Log Loss : 1.5652784657986019
for alpha = 100
Log Loss : 1.5493738728061246
for alpha = 1000
Log Loss : 1.5024243640074617

```



```

For values of best alpha = 1e-05 The train log loss is: 0.5465541436297521
For values of best alpha = 1e-05 The cross validation log loss is: 1.1694235780352655
For values of best alpha = 1e-05 The test log loss is: 1.1550904155479857

```

#### 4.1.1.2. Testing the model with best hyper paramters

```

1  # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated,
2  # -----
3  # default paramters
4  # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)
5
6  # some of methods of MultinomialNB()
7  # fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
8  # predict(X)    Perform classification on an array of test vectors X.
9  # predict_log_proba(X) Return log-probability estimates for the test vector X.
10 # -----
11 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algor
12 # -----
13
14
15 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
16 # -----
17 # default paramters
18 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
19 #
20 # some of the methods of CalibratedClassifierCV()
21 # fit(X, y[, sample_weight])    Fit the calibrated model
22 # get_params([deep])    Get parameters for this estimator.
23 # predict(X)    Predict the target of new samples.
24 # predict_proba(X) Posterior probabilities of classification
25 # -----
26
27 clf = MultinomialNB(alpha=alpha[best_alpha])
28 clf.fit(train_x_onehotCoding, train_y)
29 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
30 sig_clf.fit(train_x_onehotCoding, train_y)
31 sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
32 # to avoid rounding error while multiplying probabilites we use log-probability estimates
33 print("Log Loss :", log_loss(cv_y, sig_clf_probs))
34 print("Number of missclassified point :", np.count_nonzero((sig_clf.predict(cv_x_onehotCoding) - cv_y)))

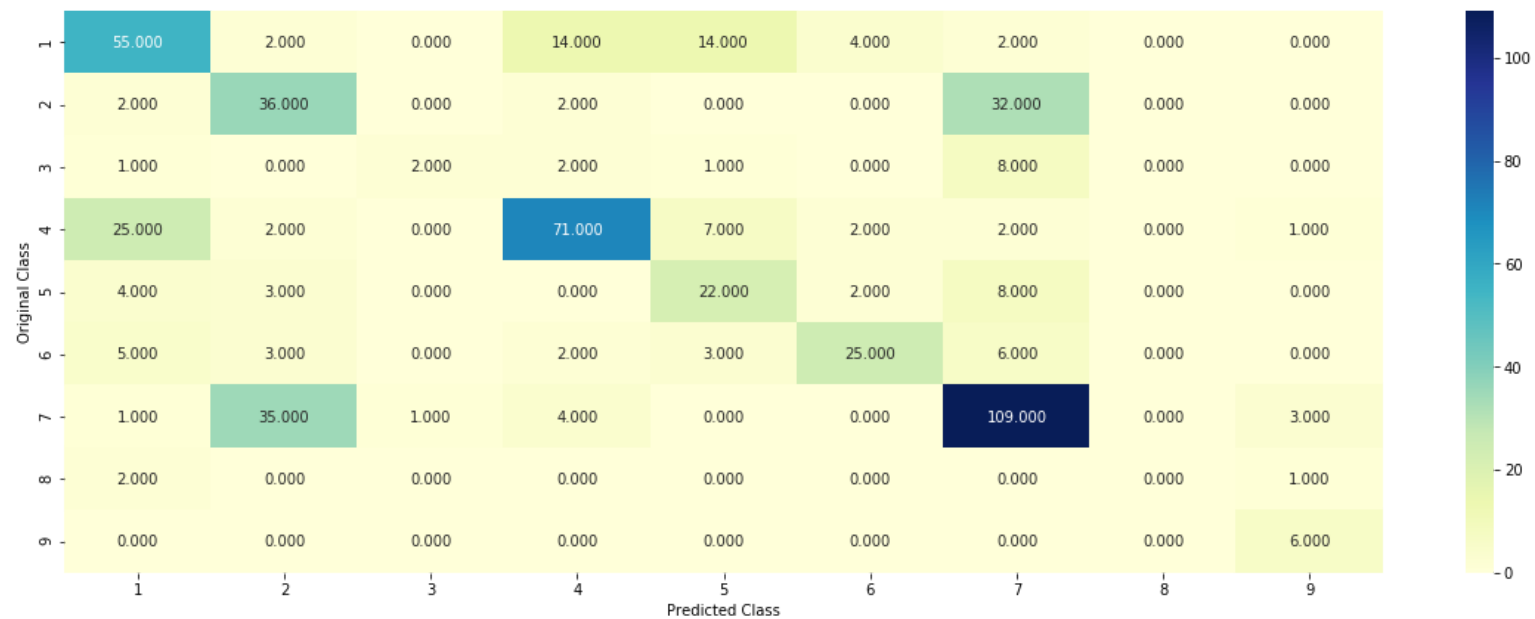
```

```
35 | plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray()))
```

Log Loss : 1.1694235780352655

Number of misclassified point : 0.38721804511278196

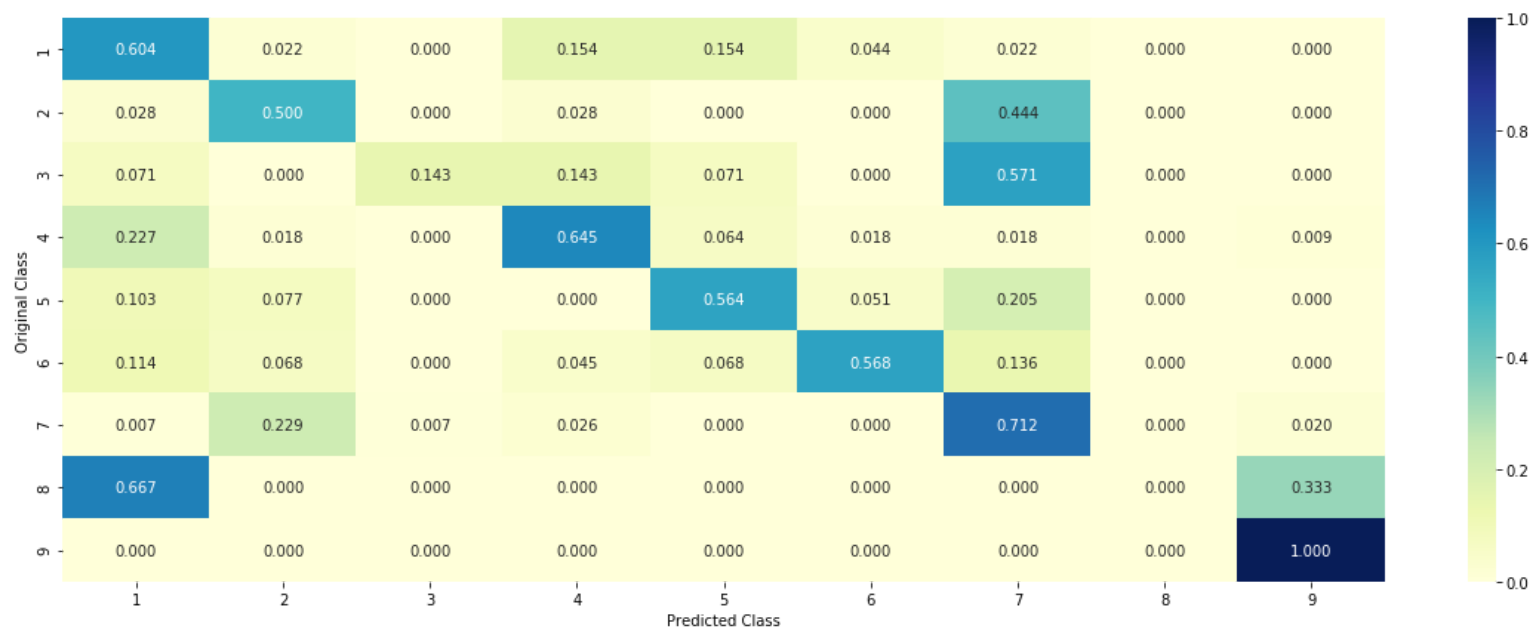
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



#### 4.1.1.3. Feature Importance, Correctly classified point





```
1 test_point_index = 1
2 no_feature = 100
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
8 print("-"*50)
9 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_
```

Predicted Class : 7

Predicted Class Probabilities: [[0.0911 0.1284 0.0141 0.1119 0.0334 0.0365 0.5764 0.0057 0.0025]]

Actual Class : 7

-----

```
16 Text feature [presence] present in test data point [True]
17 Text feature [kinase] present in test data point [True]
18 Text feature [well] present in test data point [True]
19 Text feature [activating] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [cell] present in test data point [True]
22 Text feature [inhibitor] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [recently] present in test data point [True]
29 Text feature [shown] present in test data point [True]
30 Text feature [potential] present in test data point [True]
31 Text feature [also] present in test data point [True]
32 Text feature [obtained] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [activation] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [showed] present in test data point [True]
37 Text feature [however] present in test data point [True]
38 Text feature [expressing] present in test data point [True]
39 Text feature [addition] present in test data point [True]
40 Text feature [found] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [previously] present in test data point [True]
```

43 Text feature [factor] present in test data point [True]  
44 Text feature [compared] present in test data point [True]  
45 Text feature [treated] present in test data point [True]  
46 Text feature [inhibition] present in test data point [True]  
47 Text feature [higher] present in test data point [True]  
48 Text feature [observed] present in test data point [True]  
49 Text feature [described] present in test data point [True]  
50 Text feature [may] present in test data point [True]  
51 Text feature [similar] present in test data point [True]  
52 Text feature [total] present in test data point [True]  
53 Text feature [furthermore] present in test data point [True]  
54 Text feature [studies] present in test data point [True]  
55 Text feature [using] present in test data point [True]  
56 Text feature [without] present in test data point [True]  
57 Text feature [concentrations] present in test data point [True]  
58 Text feature [1a] present in test data point [True]  
59 Text feature [various] present in test data point [True]  
60 Text feature [including] present in test data point [True]  
61 Text feature [mutations] present in test data point [True]  
62 Text feature [respectively] present in test data point [True]  
63 Text feature [12] present in test data point [True]  
64 Text feature [followed] present in test data point [True]  
65 Text feature [enhanced] present in test data point [True]  
66 Text feature [although] present in test data point [True]  
67 Text feature [interestingly] present in test data point [True]  
68 Text feature [phosphorylation] present in test data point [True]  
70 Text feature [new] present in test data point [True]  
71 Text feature [inhibited] present in test data point [True]  
72 Text feature [constitutively] present in test data point [True]  
75 Text feature [1b] present in test data point [True]  
76 Text feature [reported] present in test data point [True]  
77 Text feature [confirmed] present in test data point [True]  
78 Text feature [inhibitors] present in test data point [True]  
79 Text feature [proliferation] present in test data point [True]  
80 Text feature [report] present in test data point [True]  
81 Text feature [either] present in test data point [True]  
82 Text feature [molecular] present in test data point [True]  
83 Text feature [15] present in test data point [True]  
84 Text feature [thus] present in test data point [True]  
85 Text feature [recent] present in test data point [True]  
86 Text feature [3b] present in test data point [True]  
87 Text feature [fig] present in test data point [True]  
88 Text feature [results] present in test data point [True]

```
89 Text feature [occur] present in test data point [True]
90 Text feature [small] present in test data point [True]
91 Text feature [3a] present in test data point [True]
92 Text feature [approximately] present in test data point [True]
93 Text feature [hours] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [figure] present in test data point [True]
97 Text feature [suggests] present in test data point [True]
98 Text feature [absence] present in test data point [True]
99 Text feature [measured] present in test data point [True]
Out of the top 100 features 78 are present in query point
```

#### 4.1.1.4. Feature Importance, Incorrectly classified point

```

1 test_point_index = 100
2 no_feature = 100
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
8 print("-"*50)
9 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0876 0.0895 0.0136 0.1069 0.0322 0.0356 0.6267 0.0055 0.0024]]

Actual Class : 7

```

-----
17 Text feature [kinase] present in test data point [True]
18 Text feature [well] present in test data point [True]
19 Text feature [activating] present in test data point [True]
20 Text feature [downstream] present in test data point [True]
21 Text feature [cell] present in test data point [True]
23 Text feature [cells] present in test data point [True]
24 Text feature [independent] present in test data point [True]
25 Text feature [contrast] present in test data point [True]
26 Text feature [recently] present in test data point [True]
29 Text feature [shown] present in test data point [True]
30 Text feature [potential] present in test data point [True]
31 Text feature [also] present in test data point [True]
33 Text feature [growth] present in test data point [True]
34 Text feature [activation] present in test data point [True]
35 Text feature [suggest] present in test data point [True]
36 Text feature [showed] present in test data point [True]
37 Text feature [however] present in test data point [True]
39 Text feature [addition] present in test data point [True]
40 Text feature [found] present in test data point [True]
41 Text feature [10] present in test data point [True]
42 Text feature [previously] present in test data point [True]
44 Text feature [compared] present in test data point [True]
46 Text feature [inhibition] present in test data point [True]
47 Text feature [higher] present in test data point [True]
48 Text feature [observed] present in test data point [True]

```

```
50 Text feature [may] present in test data point [True]
51 Text feature [similar] present in test data point [True]
54 Text feature [studies] present in test data point [True]
55 Text feature [using] present in test data point [True]
56 Text feature [without] present in test data point [True]
58 Text feature [1a] present in test data point [True]
60 Text feature [including] present in test data point [True]
61 Text feature [mutations] present in test data point [True]
62 Text feature [respectively] present in test data point [True]
63 Text feature [12] present in test data point [True]
64 Text feature [followed] present in test data point [True]
65 Text feature [enhanced] present in test data point [True]
66 Text feature [although] present in test data point [True]
68 Text feature [phosphorylation] present in test data point [True]
69 Text feature [activated] present in test data point [True]
70 Text feature [new] present in test data point [True]
71 Text feature [inhibited] present in test data point [True]
72 Text feature [constitutively] present in test data point [True]
75 Text feature [1b] present in test data point [True]
78 Text feature [inhibitors] present in test data point [True]
79 Text feature [proliferation] present in test data point [True]
81 Text feature [either] present in test data point [True]
82 Text feature [molecular] present in test data point [True]
83 Text feature [15] present in test data point [True]
85 Text feature [recent] present in test data point [True]
86 Text feature [3b] present in test data point [True]
87 Text feature [fig] present in test data point [True]
88 Text feature [results] present in test data point [True]
89 Text feature [occur] present in test data point [True]
90 Text feature [small] present in test data point [True]
91 Text feature [3a] present in test data point [True]
94 Text feature [consistent] present in test data point [True]
95 Text feature [figure] present in test data point [True]
97 Text feature [suggests] present in test data point [True]
98 Text feature [absence] present in test data point [True]
Out of the top 100 features 60 are present in query point
```

## 4.2. K Nearest Neighbour Classification

### 4.2.1. Hyper parameter tuning

```

1  # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.kneighborsclassifier
2  # -----
3  # default parameter
4  # KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
5  # metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
6
7  # methods of
8  # fit(X, y) : Fit the model using X as training data and y as target values
9  # predict(X):Predict the class labels for the provided data
10 # predict_proba(X):Return probability estimates for the test data X.
11 #-----
12 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors
13 #-----
14
15
16 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.calibrated\_classifier\_cv
17 # -----
18 # default paramters
19 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
20 #
21 # some of the methods of CalibratedClassifierCV()
22 # fit(X, y[, sample_weight]) Fit the calibrated model
23 # get_params([deep]) Get parameters for this estimator.
24 # predict(X) Predict the target of new samples.
25 # predict_proba(X) Posterior probabilities of classification
26 #-----
27 # video link:
28 #-----
29
30
31 alpha = [5, 11, 15, 21, 31, 41, 51, 99]
32 cv_log_error_array = []
33 for i in alpha:
34     print("for alpha =", i)

```

```

35     clf = KNeighborsClassifier(n_neighbors=i)
36     clf.fit(train_x_responseCoding, train_y)
37     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
38     sig_clf.fit(train_x_responseCoding, train_y)
39     sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
40     cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
41     # to avoid rounding error while multiplying probabilities we use log-probability estimates
42     print("Log Loss :", log_loss(cv_y, sig_clf_probs))
43
44     fig, ax = plt.subplots()
45     ax.plot(alpha, cv_log_error_array, c='g')
46     for i, txt in enumerate(np.round(cv_log_error_array, 3)):
47         ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
48     plt.grid()
49     plt.title("Cross Validation Error for each alpha")
50     plt.xlabel("Alpha i's")
51     plt.ylabel("Error measure")
52     plt.show()
53
54
55     best_alpha = np.argmin(cv_log_error_array)
56     clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
57     clf.fit(train_x_responseCoding, train_y)
58     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
59     sig_clf.fit(train_x_responseCoding, train_y)
60
61     predict_y = sig_clf.predict_proba(train_x_responseCoding)
62     print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
63     predict_y = sig_clf.predict_proba(cv_x_responseCoding)
64     print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
65     predict_y = sig_clf.predict_proba(test_x_responseCoding)
66     print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))
67

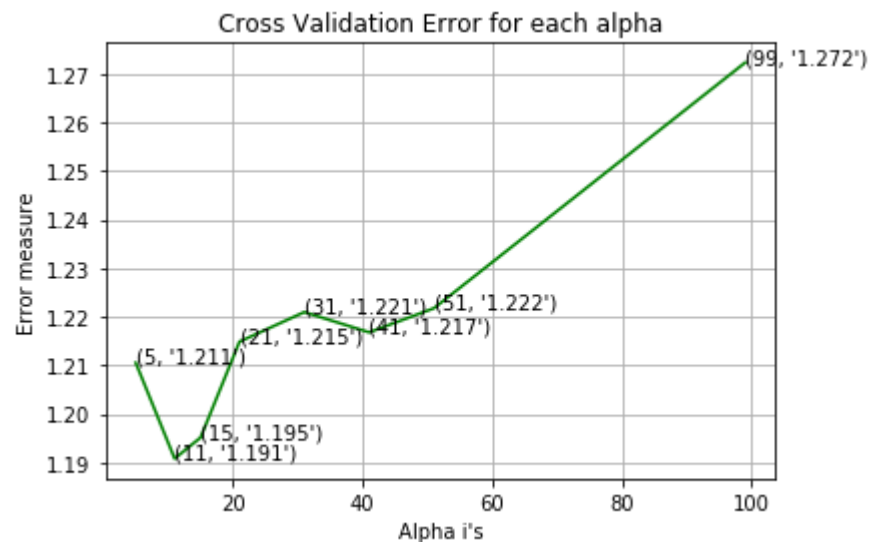
```

for alpha = 5

```

Log Loss : 1.2105157395018837
for alpha = 11
Log Loss : 1.1907654313575005
for alpha = 15
Log Loss : 1.195111998018821
for alpha = 21
Log Loss : 1.21485287042271
for alpha = 31
Log Loss : 1.220947945524939
for alpha = 41
Log Loss : 1.2167808909261397
for alpha = 51
Log Loss : 1.2217462776639945
for alpha = 99
Log Loss : 1.2723366930919473

```



```

For values of best alpha = 11 The train log loss is: 0.9863069808592798
For values of best alpha = 11 The cross validation log loss is: 1.1907654313575005
For values of best alpha = 11 The test log loss is: 1.1711855200838064

```

## 4.2.2. Testing the model with best hyper paramters



```

1 # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.
2 # -----
3 # default parameter
4 # KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
5 # metric='minkowski', metric_params=None, n_jobs=1, **kwargs)
6
7 # methods of
8 # fit(X, y) : Fit the model using X as training data and y as target values
9 # predict(X):Predict the class labels for the provided data
10 # predict_proba(X):Return probability estimates for the test data X.
11 #-----
12 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/Lessons/k-nearest-neighbor
13 #-----
14 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
15 predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)

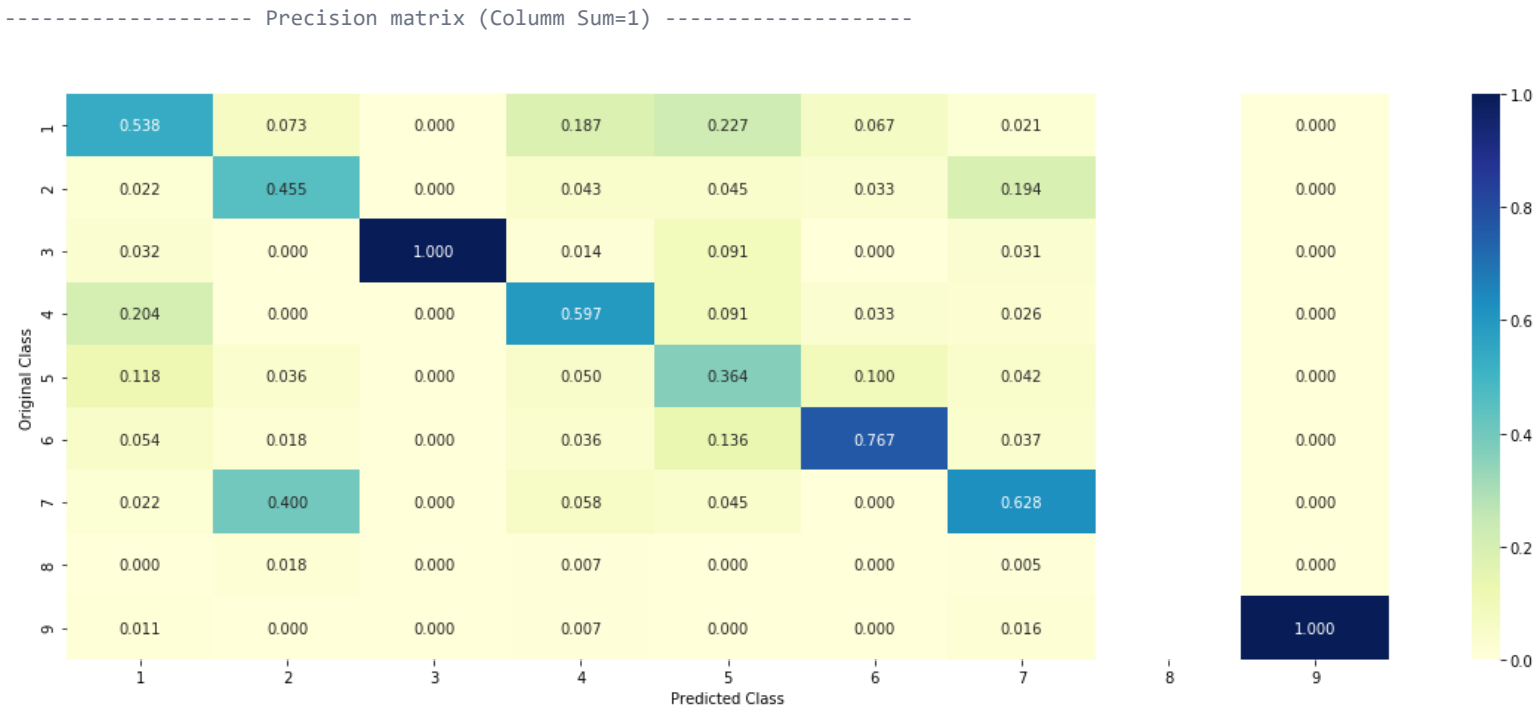
```

Log loss : 1.1907654313575005

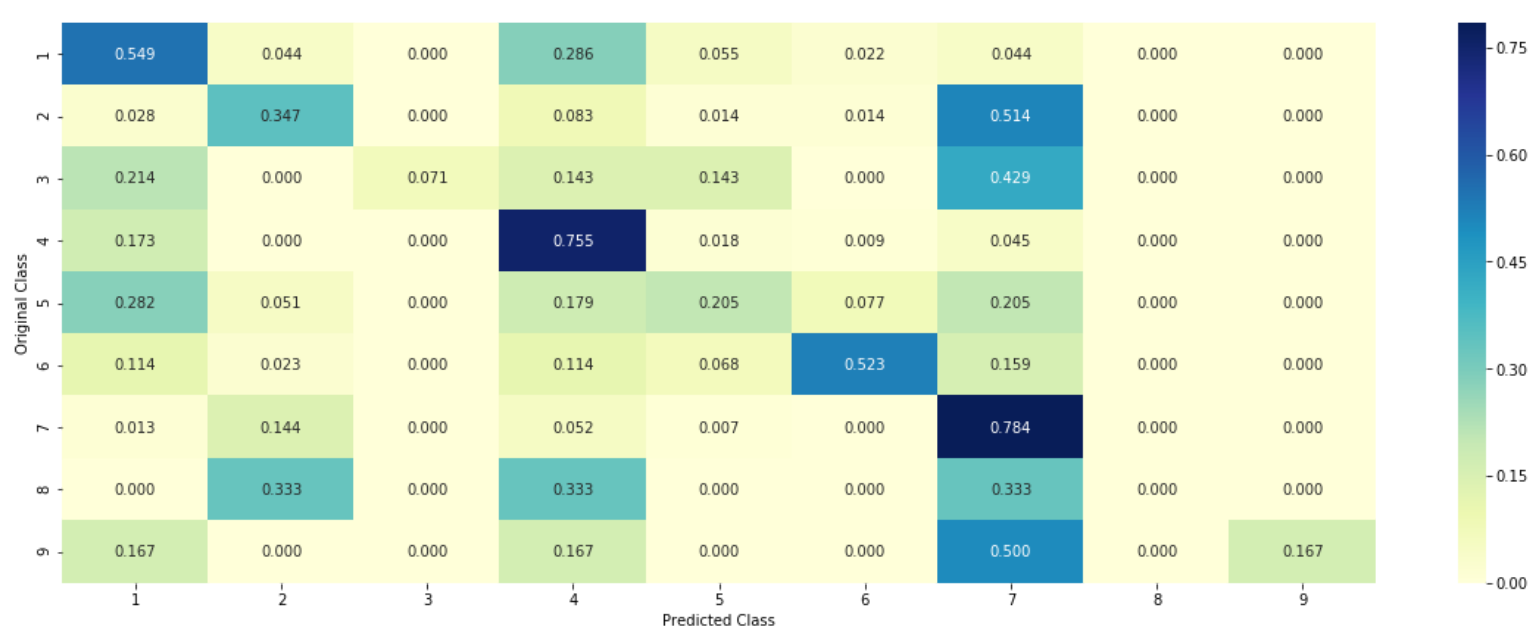
Number of mis-classified points : 0.41541353383458646

----- Confusion matrix -----





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



### 4.2.3. Sample Query point -1

```
1 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
2 clf.fit(train_x_responseCoding, train_y)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig_clf.fit(train_x_responseCoding, train_y)
5
6 test_point_index = 1
7 predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1,-1))
8 print("Predicted Class :", predicted_cls[0])
9 print("Actual Class :", test_y[test_point_index])
10 neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
11 print("The ",alpha[best_alpha]," nearest neighbours of the test points belongs to classes",train_y[neighbors[0][0]])
12 print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 7

Actual Class : 6

The 15 nearest neighbours of the test points belongs to classes [6 6 6 6 6 6 6 5 1 6 6 6 6 6 6]

Fequency of nearest points : Counter({6: 13, 5: 1, 1: 1})

#### 4.2.4. Sample Query Point-2

```
1 clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
2 clf.fit(train_x_responseCoding, train_y)
3 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4 sig_clf.fit(train_x_responseCoding, train_y)
5
6 test_point_index = 100
7
8 predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
9 print("Predicted Class :", predicted_cls[0])
10 print("Actual Class :", test_y[test_point_index])
11 neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), alpha[best_alpha])
12 print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test points belongs
13 print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 7

Actual Class : 5

the k value for knn is 15 and the nearest neighbours of the test points belongs to classes [7 7 7 7 7 7 7 7 7 7 7 7 3 3]

Fequency of nearest points : Counter({7: 13, 3: 2})

## 4.3. Logistic Regression

### 4.3.1. With Class balancing

#### 4.3.1.1. Hyper paramter tuning

```
1
2 # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
3 # -----
4 # default parameters
5 # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
6 # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
7 # class_weight=None, warm_start=False, average=False, n_iter=None)
8
9 # some of methods
10 # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
11 # predict(X) Predict class labels for samples in X.
12
13 #-----
14 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuiti
15 #-----
16
17
18 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
19 # -----
20 # default paramters
21 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
22 #
23 # some of the methods of CalibratedClassifierCV()
24 # fit(X, y[, sample_weight]) Fit the calibrated model
25 # get_params([deep]) Get parameters for this estimator.
26 # predict(X) Predict the target of new samples.
27 # predict_proba(X) Posterior probabilities of classification
28 #-----
29 # video link:
30 #-----
31
32 alpha = [10 ** x for x in range(-8, 3)]
33 cv_log_error_array = []
34 for i in alpha:
```

```

35     print("for alpha =", i)
36     clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2', loss='log', random_state=42)
37     clf.fit(train_x_onehotCoding, train_y)
38     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
39     sig_clf.fit(train_x_onehotCoding, train_y)
40     sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
41     cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
42     # to avoid rounding error while multiplying probabilities we use log-probability estimates
43     print("Log Loss :", log_loss(cv_y, sig_clf_probs))
44
45     fig, ax = plt.subplots()
46     ax.plot(alpha, cv_log_error_array, c='g')
47     for i, txt in enumerate(np.round(cv_log_error_array, 3)):
48         ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
49     plt.grid()
50     plt.title("Cross Validation Error for each alpha")
51     plt.xlabel("Alpha i's")
52     plt.ylabel("Error measure")
53     plt.show()
54
55
56     best_alpha = np.argmin(cv_log_error_array)
57     clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
58     clf.fit(train_x_onehotCoding, train_y)
59     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
60     sig_clf.fit(train_x_onehotCoding, train_y)
61
62     predict_y = sig_clf.predict_proba(train_x_onehotCoding)
63     print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predi
64     predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
65     print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_
66     predict_y = sig_clf.predict_proba(test_x_onehotCoding)
67     print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predic

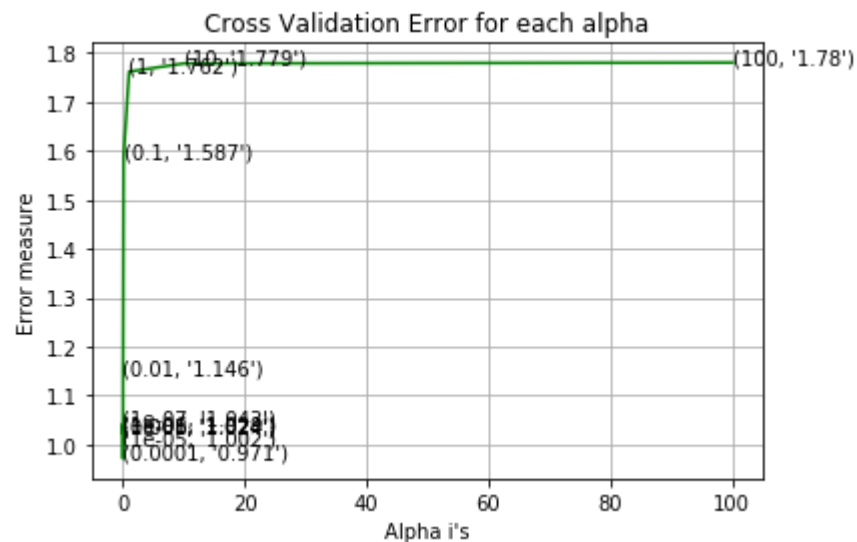
```

for alpha = 1e-08

```

Log Loss : 1.0235825979111233
for alpha = 1e-07
Log Loss : 1.042509321296799
for alpha = 1e-06
Log Loss : 1.0292628564725848
for alpha = 1e-05
Log Loss : 1.001599123255617
for alpha = 0.0001
Log Loss : 0.9708794113127591
for alpha = 0.001
Log Loss : 1.0239142786938178
for alpha = 0.01
Log Loss : 1.1457652222960732
for alpha = 0.1
Log Loss : 1.5873462623479377
for alpha = 1
Log Loss : 1.7618032536399892
for alpha = 10
Log Loss : 1.7785752978217295
for alpha = 100
Log Loss : 1.7802911854503285

```



For values of best alpha = 0.0001 The train log loss is: 0.4320099959828497

For values of best alpha = 0.0001 The cross validation log loss is: 0.9708794113127591

For values of best alpha = 0.0001 The test log loss is: 0.937647834831475



#### 4.3.1.2. Testing the model with best hyper paramters

```

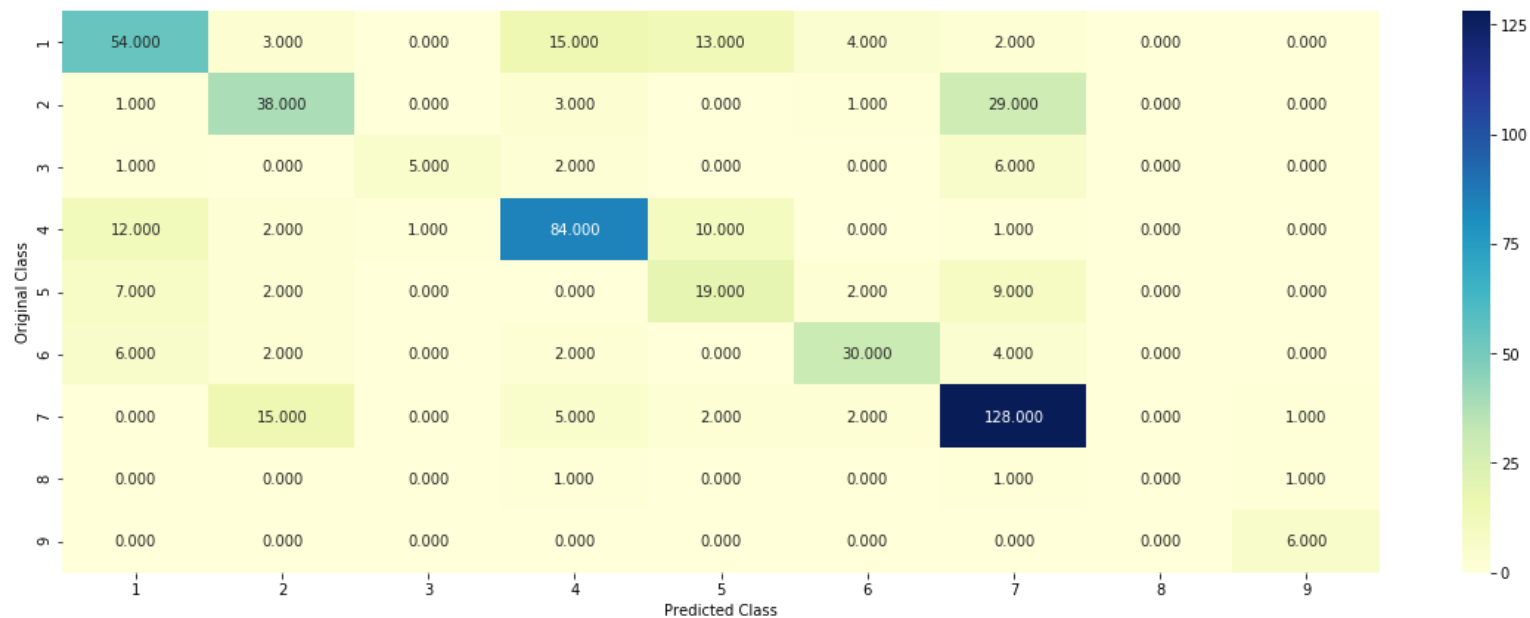
1  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model
2  # -----
3  # default parameters
4  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
5  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
6  # class_weight=None, warm_start=False, average=False, n_iter=None)
7
8  # some of methods
9  # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
10 # predict(X)    Predict class labels for samples in X.
11
12 #-----
13 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition
14 #-----
15 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=None)
16 predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

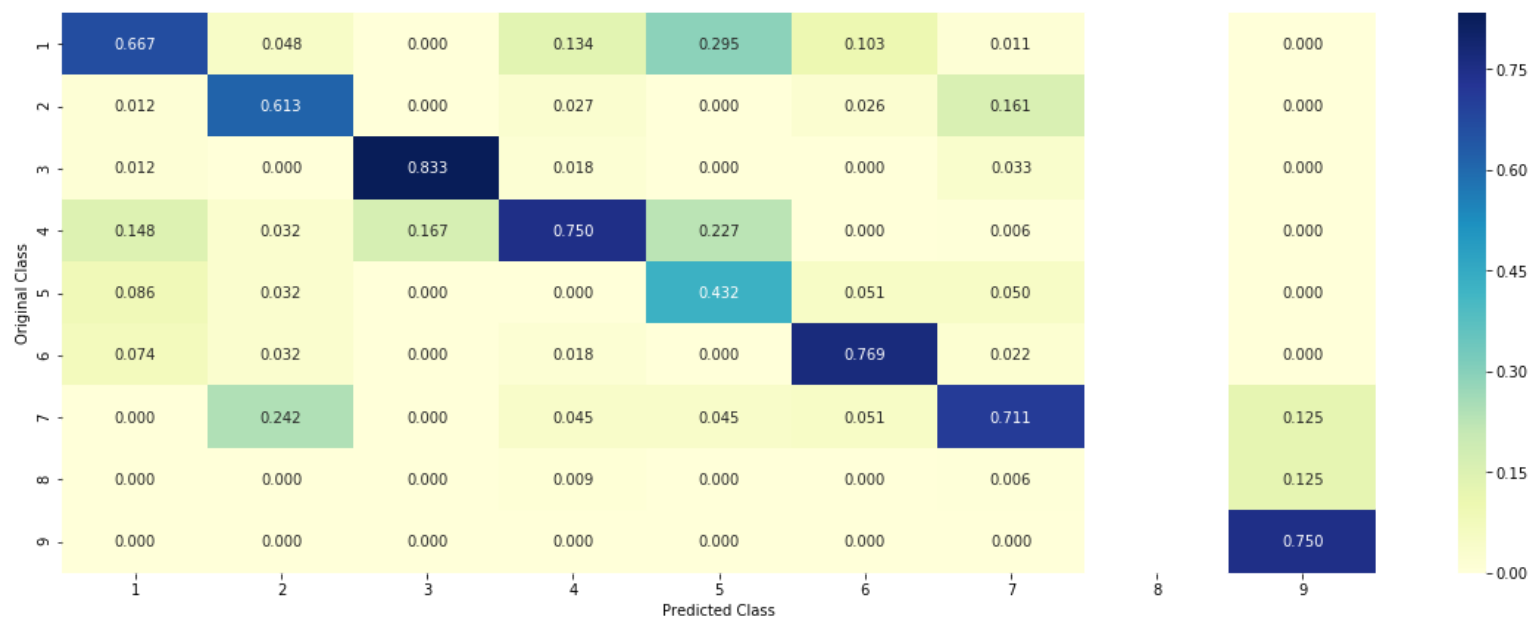
Log loss : 0.9708794113127591

Number of mis-classified points : 0.3157894736842105

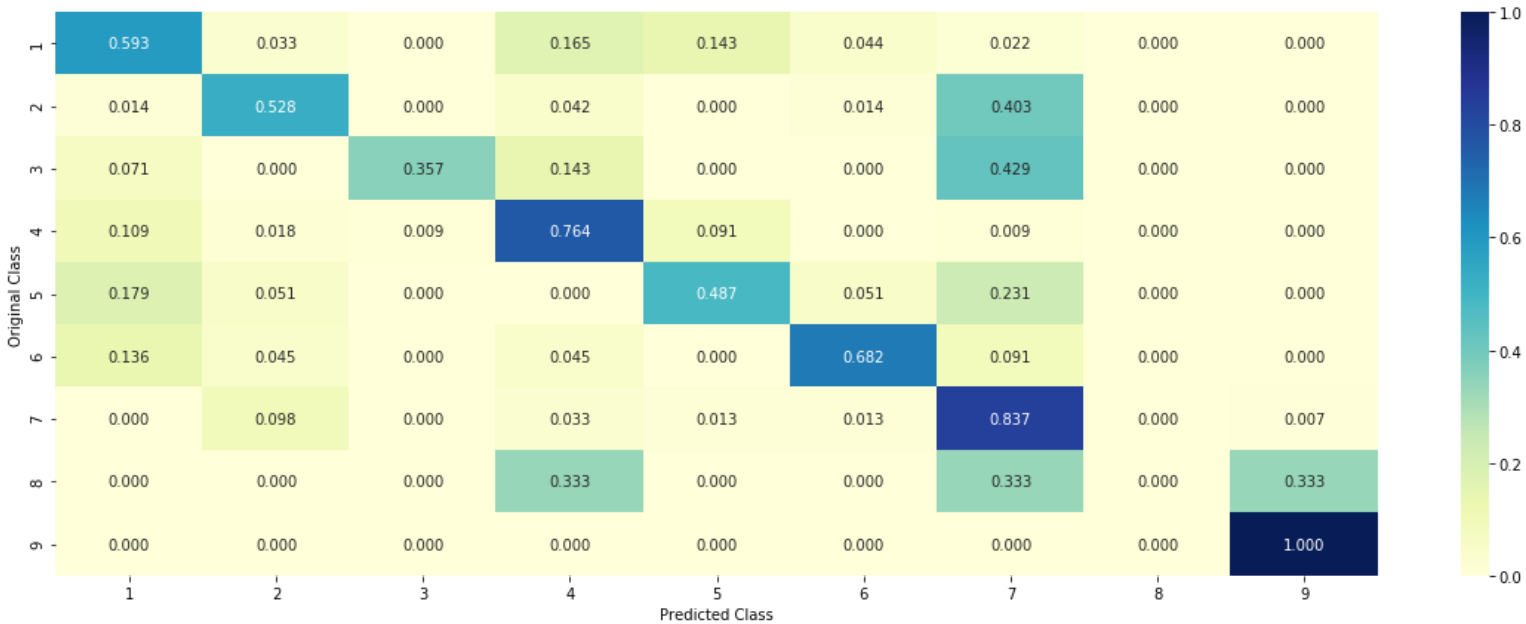
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



4.3.1.3. Feature Importance

```
1 def get_imp_feature_names(text, indices, removed_ind = []):
2     word_present = 0
3     tabulte_list = []
4     incresingorder_ind = 0
5     for i in indices:
6         if i < train_gene_feature_onehotCoding.shape[1]:
7             tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
8         elif i < 18:
9             tabulte_list.append([incresingorder_ind, "Variation", "Yes"])
10        if ((i > 17) & (i not in removed_ind)) :
11            word = train_text_features[i]
12            yes_no = True if word in text.split() else False
13            if yes_no:
14                word_present += 1
15                tabulte_list.append([incresingorder_ind, train_text_features[i], yes_no])
16            incresingorder_ind += 1
17        print(word_present, "most important features are present in our query point")
18        print("-"*50)
19        print("The features that are most important of the ", predicted_cls[0], " class:")
20        print (tabulate(tabulte_list, headers=["Index", 'Feature name', 'Present or Not']))
```

#### 4.3.1.3.1. Correctly Classified point

```

1  # from tabulate import tabulate
2  clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_
3  clf.fit(train_x_onehotCoding,train_y)
4  test_point_index = 1
5  no_feature = 500
6  predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
7  print("Predicted Class :", predicted_cls[0])
8  print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
9  print("Actual Class :", test_y[test_point_index])
10 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
11 print("-"*50)
12 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0045 0.1905 0.0012 0.0012 0.0047 0.0014 0.7872 0.0076 0.0017]]

Actual Class : 7

```

-----
23 Text feature [constitutively] present in test data point [True]
39 Text feature [flt1] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
80 Text feature [oncogenes] present in test data point [True]
84 Text feature [cysteine] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
137 Text feature [technology] present in test data point [True]
160 Text feature [dramatic] present in test data point [True]
162 Text feature [gaiix] present in test data point [True]
166 Text feature [ligand] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
181 Text feature [concentrations] present in test data point [True]
182 Text feature [thyroid] present in test data point [True]
187 Text feature [expressing] present in test data point [True]
217 Text feature [activating] present in test data point [True]
241 Text feature [cdnas] present in test data point [True]
250 Text feature [manageable] present in test data point [True]
265 Text feature [axilla] present in test data point [True]
302 Text feature [inhibitor] present in test data point [True]
311 Text feature [cot] present in test data point [True]
313 Text feature [viability] present in test data point [True]
334 Text feature [activation] present in test data point [True]

```

```
352 Text feature [forced] present in test data point [True]
368 Text feature [subcutaneous] present in test data point [True]
371 Text feature [melanocyte] present in test data point [True]
376 Text feature [erk1] present in test data point [True]
388 Text feature [hours] present in test data point [True]
446 Text feature [procure] present in test data point [True]
448 Text feature [doses] present in test data point [True]
480 Text feature [mapk] present in test data point [True]
Out of the top 500 features 30 are present in query point
```

#### 4.3.1.3.2. Incorrectly Classified point

```

1 test_point_index = 100
2 no_feature = 500
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]), 4))
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
8 print("-"*50)
9 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_index])

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0482 0.2032 0.0108 0.0446 0.071 0.0164 0.5932 0.0078 0.0046]]

Actual Class : 7

```

-----
23 Text feature [constitutively] present in test data point [True]
29 Text feature [constitutive] present in test data point [True]
47 Text feature [activated] present in test data point [True]
79 Text feature [oncogene] present in test data point [True]
89 Text feature [inhibited] present in test data point [True]
93 Text feature [transforming] present in test data point [True]
108 Text feature [transform] present in test data point [True]
148 Text feature [receptors] present in test data point [True]
177 Text feature [downstream] present in test data point [True]
210 Text feature [isozyme] present in test data point [True]
217 Text feature [activating] present in test data point [True]
232 Text feature [exchange] present in test data point [True]
326 Text feature [murine] present in test data point [True]
333 Text feature [agar] present in test data point [True]
334 Text feature [activation] present in test data point [True]
Out of the top 500 features 15 are present in query point

```

## 4.3.2. Without Class balancing

### 4.3.2.1. Hyper paramter tuning



```

1  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_mo
2  # -----
3  # default parameters
4  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=N
5  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
6  # class_weight=None, warm_start=False, average=False, n_iter=None)
7
8  # some of methods
9  # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
10 # predict(X)    Predict class labels for samples in X.
11
12 #-----
13 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuiti
14 #-----
15
16
17
18 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
19 # -----
20 # default paramters
21 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
22 #
23 # some of the methods of CalibratedClassifierCV()
24 # fit(X, y[, sample_weight])    Fit the calibrated model
25 # get_params([deep])    Get parameters for this estimator.
26 # predict(X)    Predict the target of new samples.
27 # predict_proba(X)    Posterior probabilities of classification
28 #-----
29 # video link:
30 #-----
31
32 alpha = [10 ** x for x in range(-8, 3)]
33 cv_log_error_array = []
34 for i in alpha:

```

```

35     print("for alpha =", i)
36     clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
37     clf.fit(train_x_onehotCoding, train_y)
38     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
39     sig_clf.fit(train_x_onehotCoding, train_y)
40     sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
41     cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
42     print("Log Loss :", log_loss(cv_y, sig_clf_probs))
43
44 fig, ax = plt.subplots()
45 ax.plot(alpha, cv_log_error_array, c='g')
46 for i, txt in enumerate(np.round(cv_log_error_array, 3)):
47     ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
48 plt.grid()
49 plt.title("Cross Validation Error for each alpha")
50 plt.xlabel("Alpha i's")
51 plt.ylabel("Error measure")
52 plt.show()
53
54
55 best_alpha = np.argmin(cv_log_error_array)
56 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
57 clf.fit(train_x_onehotCoding, train_y)
58 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
59 sig_clf.fit(train_x_onehotCoding, train_y)
60
61 predict_y = sig_clf.predict_proba(train_x_onehotCoding)
62 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
63 predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
64 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y))
65 predict_y = sig_clf.predict_proba(test_x_onehotCoding)
66 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))

```

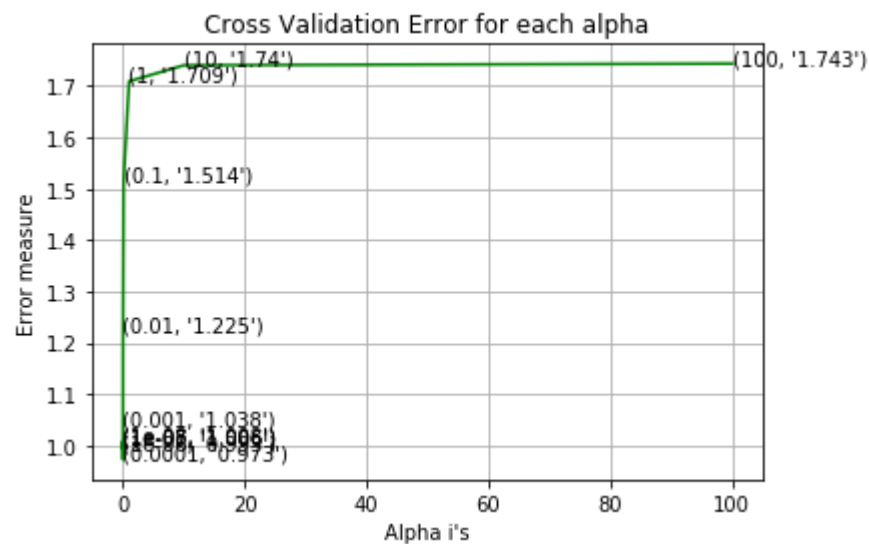
for alpha = 1e-08

Log Loss : 1.0077619488195633

```

for alpha = 1e-07
Log Loss : 1.0059137316873308
for alpha = 1e-06
Log Loss : 0.9933698852520187
for alpha = 1e-05
Log Loss : 1.005666682738722
for alpha = 0.0001
Log Loss : 0.9727390915718834
for alpha = 0.001
Log Loss : 1.0383018095431007
for alpha = 0.01
Log Loss : 1.2246582500935372
for alpha = 0.1
Log Loss : 1.5141250061352167
for alpha = 1
Log Loss : 1.708960638300253
for alpha = 10
Log Loss : 1.7401886457350255
for alpha = 100
Log Loss : 1.7434287209568087

```



For values of best alpha = 0.0001 The train log loss is: 0.4229145557036816

For values of best alpha = 0.0001 The cross validation log loss is: 0.9727390915718834

For values of best alpha = 0.0001 The test log loss is: 0.9333137008092869

#### 4.3.2.2. Testing model with best hyper parameters

```

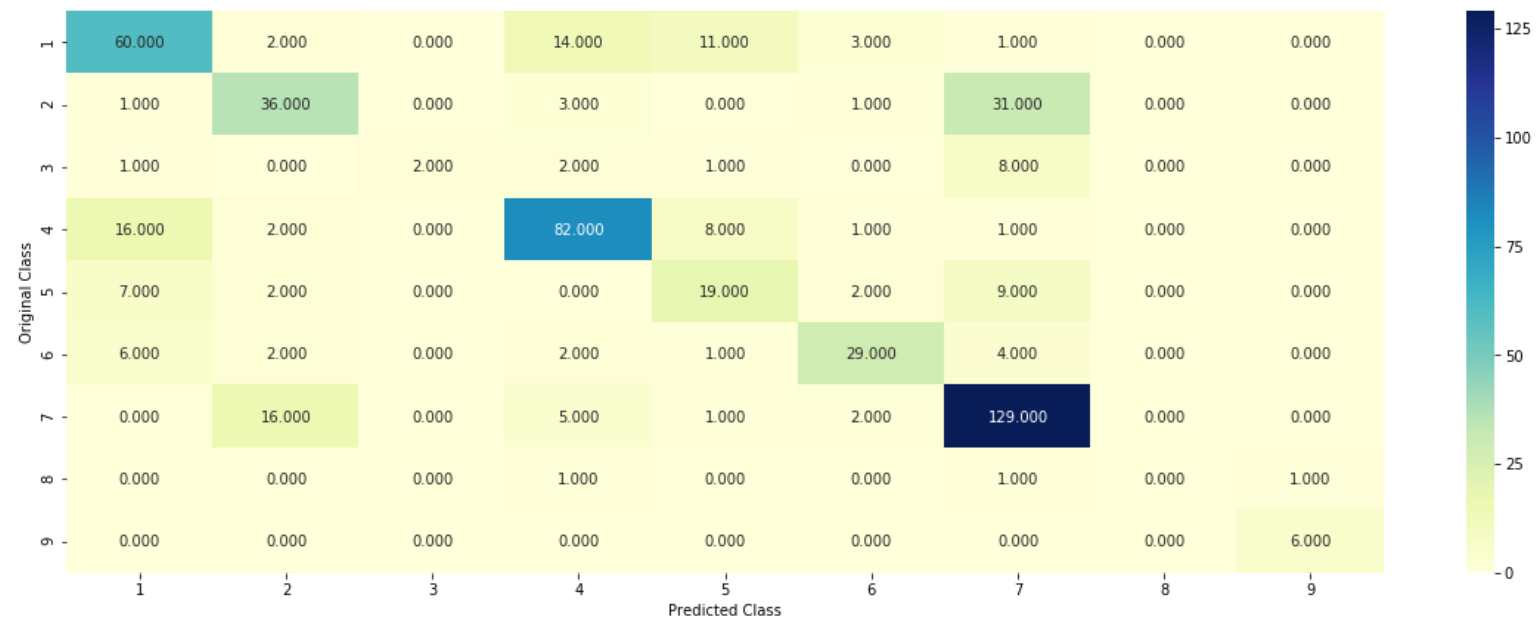
1  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model
2  # -----
3  # default parameters
4  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=Nu
5  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
6  # class_weight=None, warm_start=False, average=False, n_iter=None)
7
8  # some of methods
9  # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
10 # predict(X)    Predict class labels for samples in X.
11
12 #-----
13 # video link:
14 #-----
15
16 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
17 predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

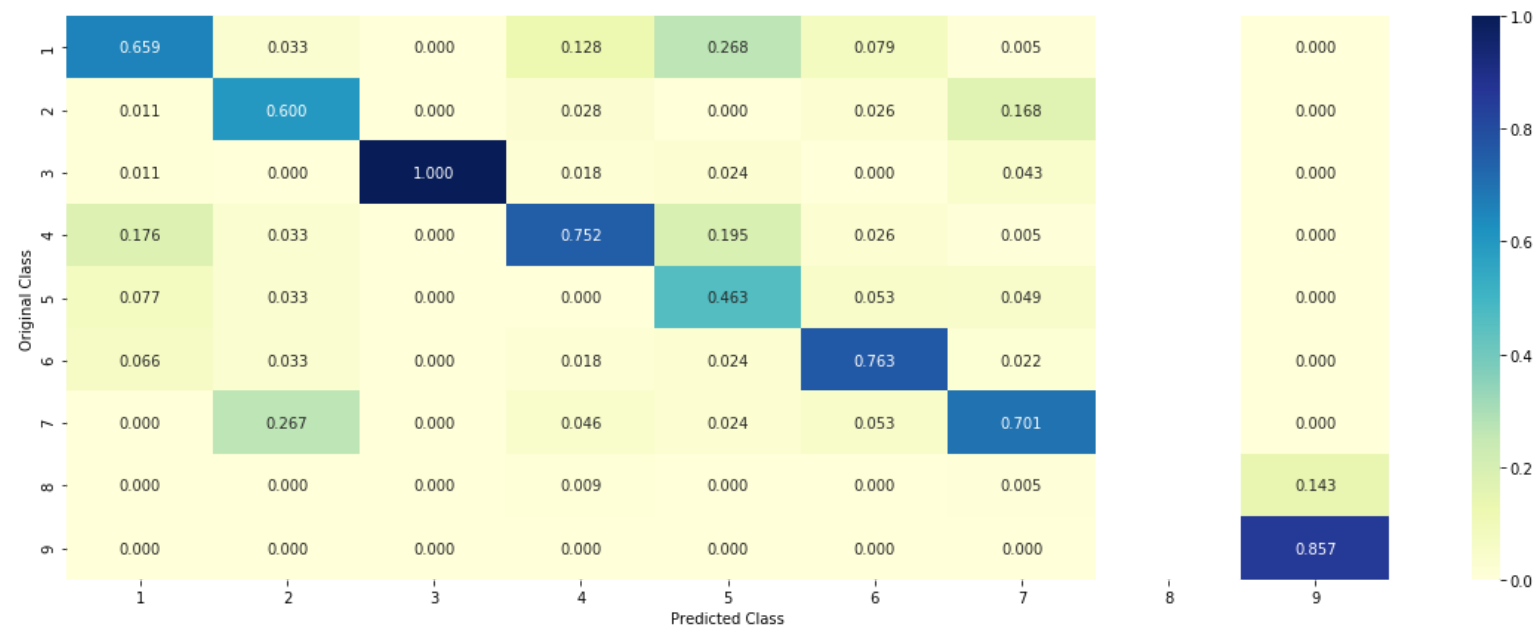
Log loss : 0.9727390915718834

Number of mis-classified points : 0.3176691729323308

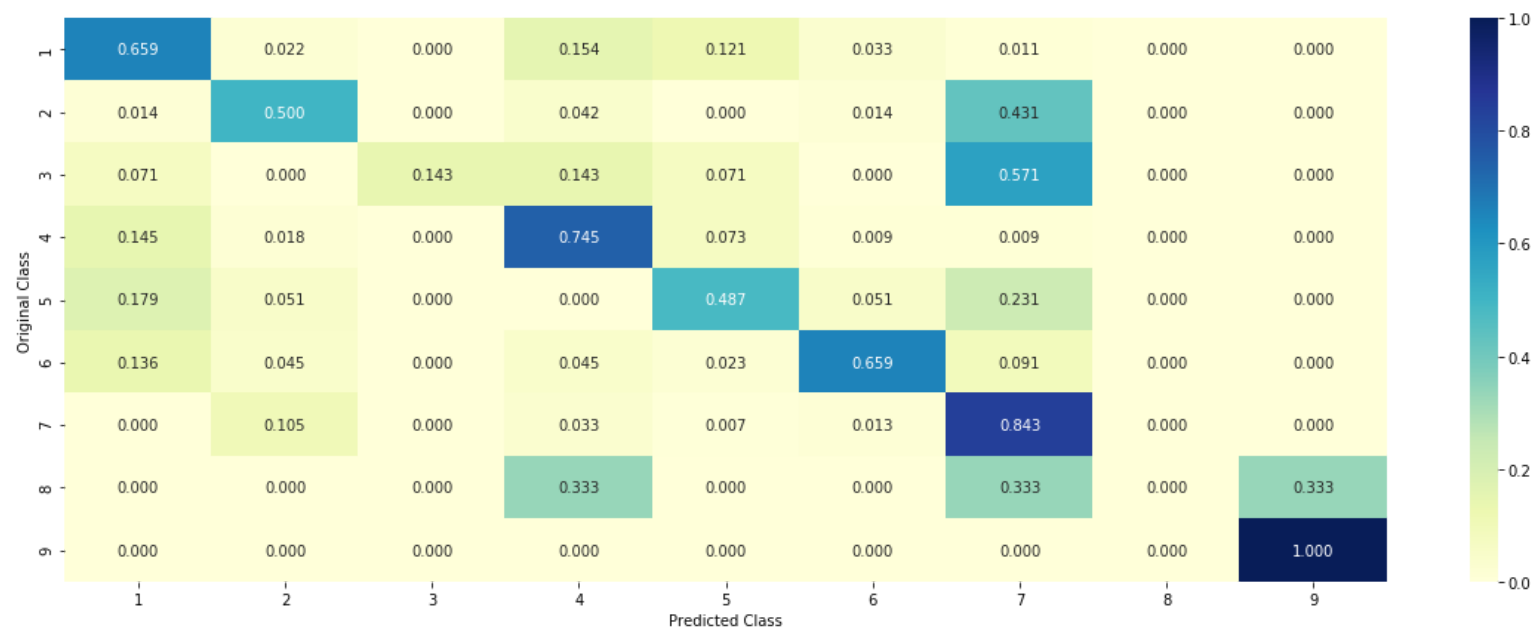
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



#### 4.3.2.3. Feature Importance, Correctly Classified point

```

1  clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
2  clf.fit(train_x_onehotCoding,train_y)
3  test_point_index = 1
4  no_feature = 500
5  predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
6  print("Predicted Class :", predicted_cls[0])
7  print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
8  print("Actual Class :", test_y[test_point_index])
9  indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
10 print("-"*50)
11 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[5.100e-03 1.255e-01 2.000e-04 1.300e-03 2.300e-03 1.400e-03 8.556e-01  
8.500e-03 1.000e-04]]

Actual Class : 7

```

-----
60 Text feature [constitutively] present in test data point [True]
107 Text feature [flt1] present in test data point [True]
124 Text feature [cysteine] present in test data point [True]
157 Text feature [oncogenes] present in test data point [True]
158 Text feature [inhibited] present in test data point [True]
195 Text feature [activating] present in test data point [True]
200 Text feature [ligand] present in test data point [True]
203 Text feature [oncogene] present in test data point [True]
204 Text feature [technology] present in test data point [True]
257 Text feature [gaiix] present in test data point [True]
260 Text feature [concentrations] present in test data point [True]
265 Text feature [downstream] present in test data point [True]
314 Text feature [hki] present in test data point [True]
316 Text feature [dramatic] present in test data point [True]
323 Text feature [expressing] present in test data point [True]
371 Text feature [cdnas] present in test data point [True]
380 Text feature [viability] present in test data point [True]
412 Text feature [thyroid] present in test data point [True]
459 Text feature [activation] present in test data point [True]
461 Text feature [manageable] present in test data point [True]
462 Text feature [ser473] present in test data point [True]
468 Text feature [axilla] present in test data point [True]

```



495 Text feature [extracellular] present in test data point [True]  
 Out of the top 500 features 23 are present in query point

#### 4.3.2.4. Feature Importance, Inorrectly Classified point

```

1 test_point_index = 100
2 no_feature = 500
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.coef_)[predicted_cls-1][:, :no_feature]
8 print("-"*50)
9 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0485 0.1851 0.0052 0.0442 0.0617 0.0143 0.6317 0.0072 0.0022]]

Actual Class : 7

-----

60 Text feature [constitutively] present in test data point [True]  
 89 Text feature [constitutive] present in test data point [True]  
 116 Text feature [activated] present in test data point [True]  
 158 Text feature [inhibited] present in test data point [True]  
 159 Text feature [transforming] present in test data point [True]  
 193 Text feature [receptors] present in test data point [True]  
 195 Text feature [activating] present in test data point [True]  
 203 Text feature [oncogene] present in test data point [True]  
 226 Text feature [transform] present in test data point [True]  
 241 Text feature [isozyme] present in test data point [True]  
 265 Text feature [downstream] present in test data point [True]  
 377 Text feature [agar] present in test data point [True]  
 442 Text feature [interatomic] present in test data point [True]  
 459 Text feature [activation] present in test data point [True]  
 Out of the top 500 features 14 are present in query point

## 4.4. Linear Support Vector Machines

### 4.4.1. Hyper paramter tuning



```

1  # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
2
3  # -----
4  # default parameters
5  # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
6  # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
7
8  # Some of methods of SVM()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # -----
12 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivatives-of-svm
13 # -----
14
15
16
17 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
18 # -----
19 # default parameters
20 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
21 #
22 # some of the methods of CalibratedClassifierCV()
23 # fit(X, y[, sample_weight])    Fit the calibrated model
24 # get_params([deep])    Get parameters for this estimator.
25 # predict(X)    Predict the target of new samples.
26 # predict_proba(X)    Posterior probabilities of classification
27 # -----
28 # video link:
29 # -----
30
31 alpha = [10 ** x for x in range(-8, 3)]
32 cv_log_error_array = []
33 for i in alpha:
34     print("for C =", i)

```

```

35 #     clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
36     clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
37     clf.fit(train_x_onehotCoding, train_y)
38     sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
39     sig_clf.fit(train_x_onehotCoding, train_y)
40     sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
41     cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
42     print("Log Loss :", log_loss(cv_y, sig_clf_probs))
43
44 fig, ax = plt.subplots()
45 ax.plot(alpha, cv_log_error_array, c='g')
46 for i, txt in enumerate(np.round(cv_log_error_array, 3)):
47     ax.annotate((alpha[i], str(txt)), (alpha[i], cv_log_error_array[i]))
48 plt.grid()
49 plt.title("Cross Validation Error for each alpha")
50 plt.xlabel("Alpha i's")
51 plt.ylabel("Error measure")
52 plt.show()
53
54
55 best_alpha = np.argmin(cv_log_error_array)
56 # clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
57 clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
58 clf.fit(train_x_onehotCoding, train_y)
59 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
60 sig_clf.fit(train_x_onehotCoding, train_y)
61
62 predict_y = sig_clf.predict_proba(train_x_onehotCoding)
63 print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y))
64 predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
65 print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_train, predict_y))
66 predict_y = sig_clf.predict_proba(test_x_onehotCoding)
67 print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y))

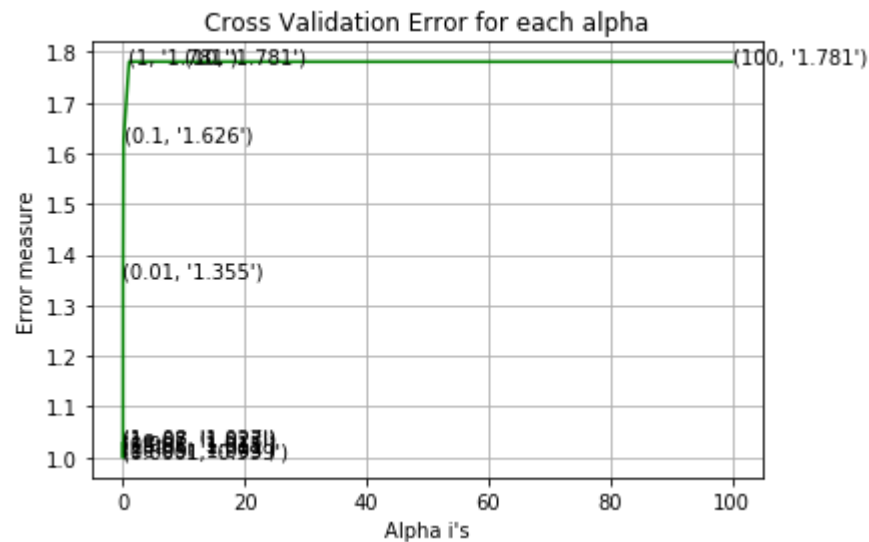
```

for C = 1e-08

```

Log Loss : 1.0274812639356432
for C = 1e-07
Log Loss : 1.0234226694876858
for C = 1e-06
Log Loss : 1.0108477818459018
for C = 1e-05
Log Loss : 1.0026246990710774
for C = 0.0001
Log Loss : 0.9993499412634308
for C = 0.001
Log Loss : 1.0151339586211054
for C = 0.01
Log Loss : 1.3554727775339737
for C = 0.1
Log Loss : 1.625997059089221
for C = 1
Log Loss : 1.7808303595587045
for C = 10
Log Loss : 1.7805701916057137
for C = 100
Log Loss : 1.7805701892766186

```



```

For values of best alpha = 0.0001 The train log loss is: 0.48469065401759154
For values of best alpha = 0.0001 The cross validation log loss is: 0.9993499412634308
For values of best alpha = 0.0001 The test log loss is: 0.9869218828762779

```

#### 4.4.2. Testing model with best hyper parameters

```

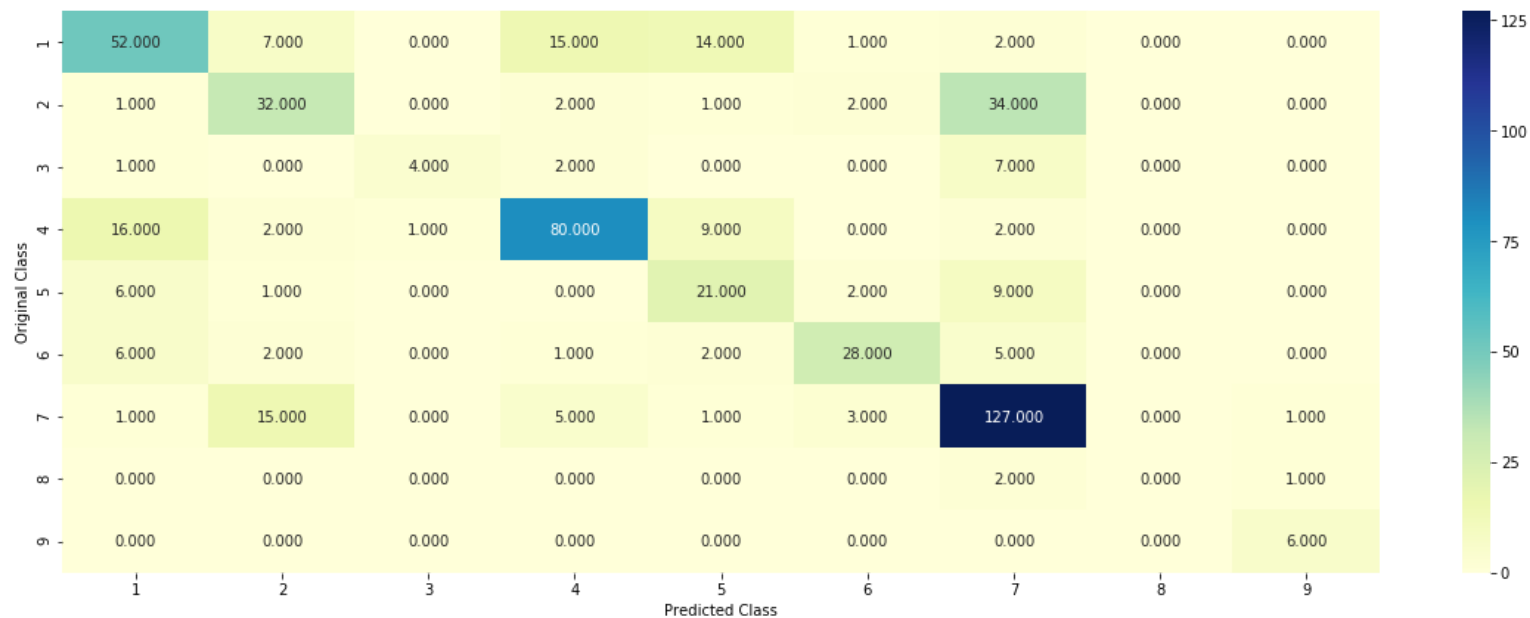
1  # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/svm.html
2
3  # -----
4  # default parameters
5  # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
6  # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
7
8  # Some of methods of SVM()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # -----
12 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivatives-of-svm
13 # -----
14
15
16 # clf = SVC(C=alpha[best_alpha], kernel='linear', probability=True, class_weight='balanced')
17 clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42, class_weight='balanced')
18 predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, cv_y, clf)

```

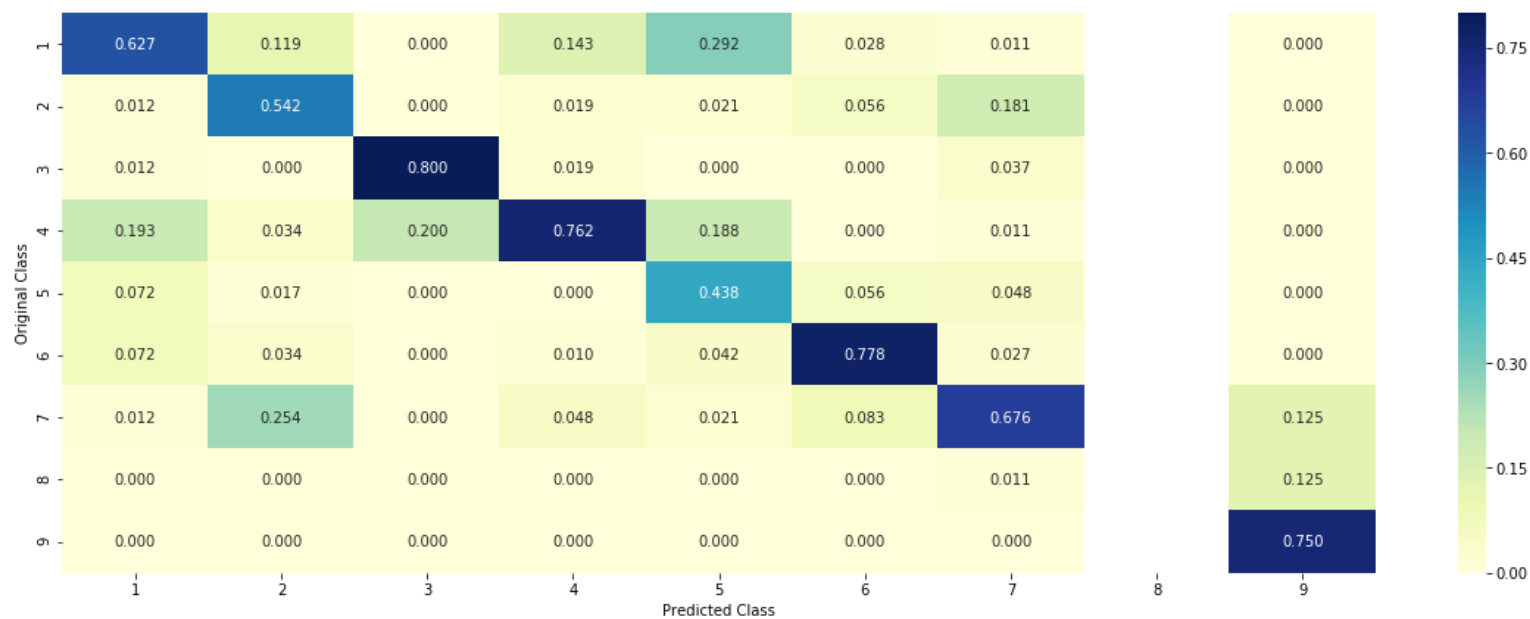
Log loss : 0.9993499412634308

Number of mis-classified points : 0.34210526315789475

----- Confusion matrix -----

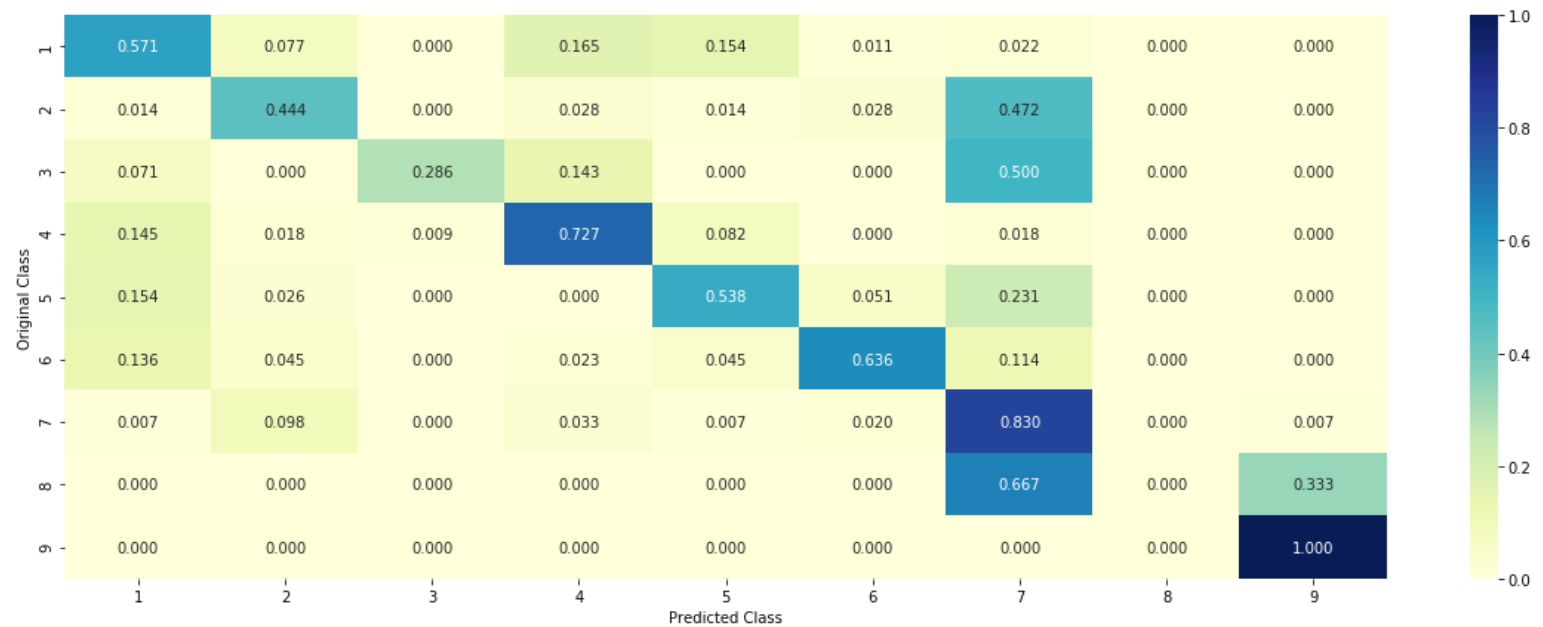


----- Precision matrix (Column Sum=1) -----



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





### 4.3.3. Feature Importance

#### 4.3.3.1. For Correctly classified point

```

1  clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
2  clf.fit(train_x_onehotCoding,train_y)
3  test_point_index = 1
4  # test_point_index = 100
5  no_feature = 500
6  predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
7  print("Predicted Class :", predicted_cls[0])
8  print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
9  print("Actual Class :", test_y[test_point_index])
10 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
11 print("-"*50)
12 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0153 0.1199 0.0029 0.0151 0.0121 0.0075 0.8104 0.0129 0.0039]]

Actual Class : 7

```

-----
28 Text feature [constitutively] present in test data point [True]
29 Text feature [cysteine] present in test data point [True]
49 Text feature [cdnas] present in test data point [True]
76 Text feature [flt1] present in test data point [True]
79 Text feature [concentrations] present in test data point [True]
82 Text feature [gaiix] present in test data point [True]
96 Text feature [technology] present in test data point [True]
101 Text feature [inhibited] present in test data point [True]
104 Text feature [activating] present in test data point [True]
114 Text feature [oncogenes] present in test data point [True]
147 Text feature [expressing] present in test data point [True]
150 Text feature [mapk] present in test data point [True]
151 Text feature [oncogene] present in test data point [True]
169 Text feature [thyroid] present in test data point [True]
171 Text feature [inhibitor] present in test data point [True]
205 Text feature [transduced] present in test data point [True]
211 Text feature [seeded] present in test data point [True]
230 Text feature [ligand] present in test data point [True]
255 Text feature [activation] present in test data point [True]
279 Text feature [downstream] present in test data point [True]
314 Text feature [doses] present in test data point [True]
351 Text feature [subcutaneous] present in test data point [True]

```

```
366 Text feature [atcc] present in test data point [True]
405 Text feature [melanocyte] present in test data point [True]
436 Text feature [hours] present in test data point [True]
445 Text feature [selleck] present in test data point [True]
446 Text feature [dramatic] present in test data point [True]
454 Text feature [chemiluminescence] present in test data point [True]
487 Text feature [viability] present in test data point [True]
489 Text feature [ser473] present in test data point [True]
Out of the top 500 features 30 are present in query point
```

#### 4.3.3.2. For Incorrectly classified point

```

1 test_point_index = 100
2 no_feature = 500
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
8 print("-"*50)
9 get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],test_df['Gene'].iloc[test_point_

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0786 0.1516 0.0146 0.1064 0.1105 0.0323 0.4839 0.0128 0.0094]]

Actual Class : 7

```

-----
28 Text feature [constitutively] present in test data point [True]
40 Text feature [constitutive] present in test data point [True]
73 Text feature [activated] present in test data point [True]
75 Text feature [transforming] present in test data point [True]
94 Text feature [receptors] present in test data point [True]
97 Text feature [exchange] present in test data point [True]
101 Text feature [inhibited] present in test data point [True]
104 Text feature [activating] present in test data point [True]
151 Text feature [oncogene] present in test data point [True]
231 Text feature [transform] present in test data point [True]
255 Text feature [activation] present in test data point [True]
279 Text feature [downstream] present in test data point [True]
440 Text feature [doubled] present in test data point [True]
470 Text feature [substituting] present in test data point [True]
Out of the top 500 features 14 are present in query point

```

## 4.5 Random Forest Classifier

### 4.5.1. Hyper paramter tuning (With One hot Encoding)

```

1  # -----
2  # default parameters
3  # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample:
4  # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
5  # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, wari
6  # class_weight=None)
7
8  # Some of methods of RandomForestClassifier()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # predict_proba (X) Perform classification on samples in X.
12
13 # some of attributes of RandomForestClassifier()
14 # feature_importances_ : array of shape = [n_features]
15 # The feature importances (the higher, the more important the feature).
16
17 # -----
18 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
19 # -----
20
21
22 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklea
23 # -----
24 # default paramters
25 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
26 #
27 # some of the methods of CalibratedClassifierCV()
28 # fit(X, y[, sample_weight])    Fit the calibrated model
29 # get_params([deep])    Get parameters for this estimator.
30 # predict(X)    Predict the target of new samples.
31 # predict_proba(X) Posterior probabilities of classification
32 #-----
33 # video link:
34 #-----

```

```
35
36 alpha = [100,200,500,1000,2000]
37 max_depth = [5, 10]
38 cv_log_error_array = []
39 for i in alpha:
40     for j in max_depth:
41         print("for n_estimators =", i,"and max depth = ", j)
42         clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_
43         clf.fit(train_x_onehotCoding, train_y)
44         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
45         sig_clf.fit(train_x_onehotCoding, train_y)
46         sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
47         cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
48         print("Log Loss :",log_loss(cv_y, sig_clf_probs))
49
50 '''fig, ax = plt.subplots()
51 features = np.dot(np.array(alpha)[: ,None],np.array(max_depth)[None]).ravel()
52 ax.plot(features, cv_log_error_array,c='g')
53 for i, txt in enumerate(np.round(cv_log_error_array,3)):
54     ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (features[i],cv_log_error_array[i]))
55 plt.grid()
56 plt.title("Cross Validation Error for each alpha")
57 plt.xlabel("Alpha i's")
58 plt.ylabel("Error measure")
59 plt.show()
60 '''
61
62 best_alpha = np.argmin(cv_log_error_array)
63 clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_dept
64 clf.fit(train_x_onehotCoding, train_y)
65 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
66 sig_clf.fit(train_x_onehotCoding, train_y)
67
68 predict_y = sig_clf.predict_proba(train_x_onehotCoding)
69 print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:",log_loss(y_
```

```

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

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.2122164395282773
for n_estimators = 100 and max depth = 10
Log Loss : 1.2727761783679366
for n_estimators = 200 and max depth = 5
Log Loss : 1.1930324598373374
for n_estimators = 200 and max depth = 10
Log Loss : 1.257686233418453
for n_estimators = 500 and max depth = 5
Log Loss : 1.1837485653657103
for n_estimators = 500 and max depth = 10
Log Loss : 1.245519161210513
for n_estimators = 1000 and max depth = 5
Log Loss : 1.1789928962594791
for n_estimators = 1000 and max depth = 10
Log Loss : 1.2441267586106597
for n_estimators = 2000 and max depth = 5
Log Loss : 1.179122510770064
for n_estimators = 2000 and max depth = 10
Log Loss : 1.2433154816817893
For values of best estimator = 1000 The train log loss is: 0.862607994394021
For values of best estimator = 1000 The cross validation log loss is: 1.1789928962594791
For values of best estimator = 1000 The test log loss is: 1.1581753252838762

```

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

```

1  # -----
2  # default parameters
3  # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample:
4  # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
5  # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, wari
6  # class_weight=None)
7
8  # Some of methods of RandomForestClassifier()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # predict_proba (X) Perform classification on samples in X.
12
13 # some of attributes of RandomForestClassifier()
14 # feature_importances_ : array of shape = [n_features]
15 # The feature importances (the higher, the more important the feature).
16
17 # -----
18 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
19 # -----
20
21 clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_dep
22 predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_y, clf)

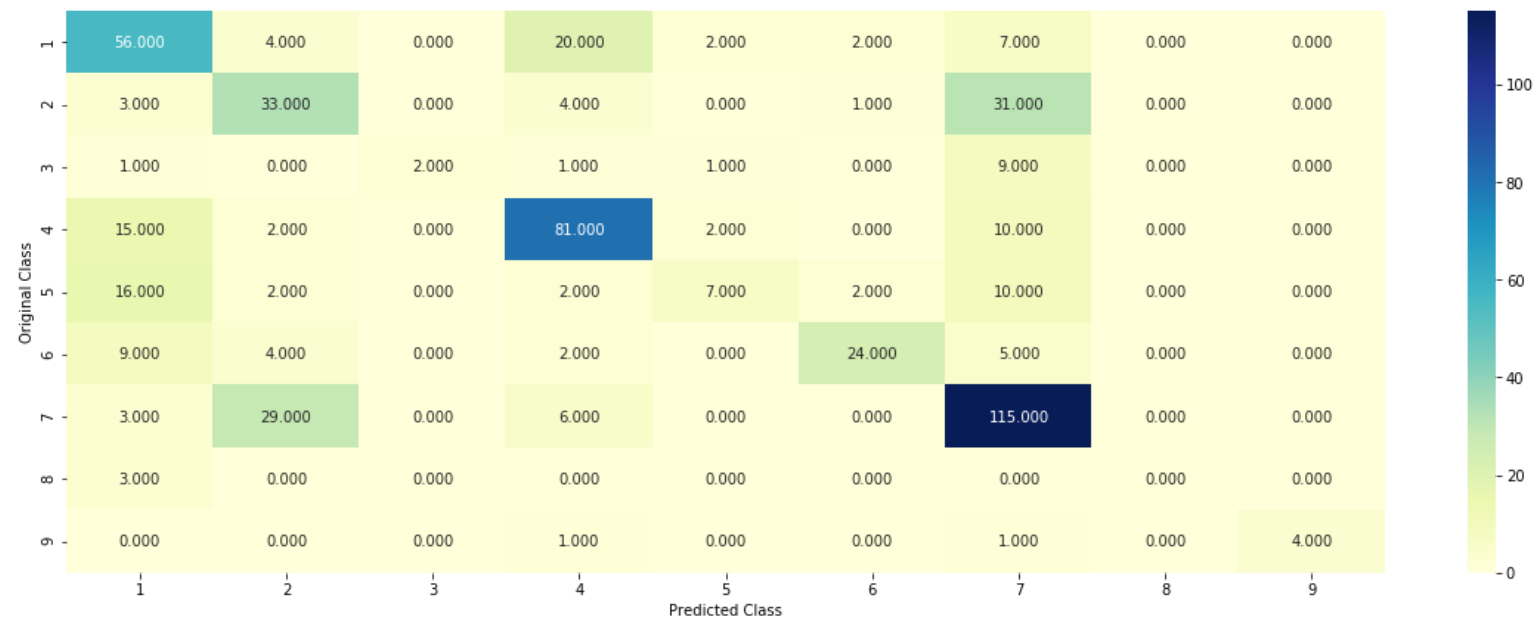
```

Log loss : 1.1789928962594791

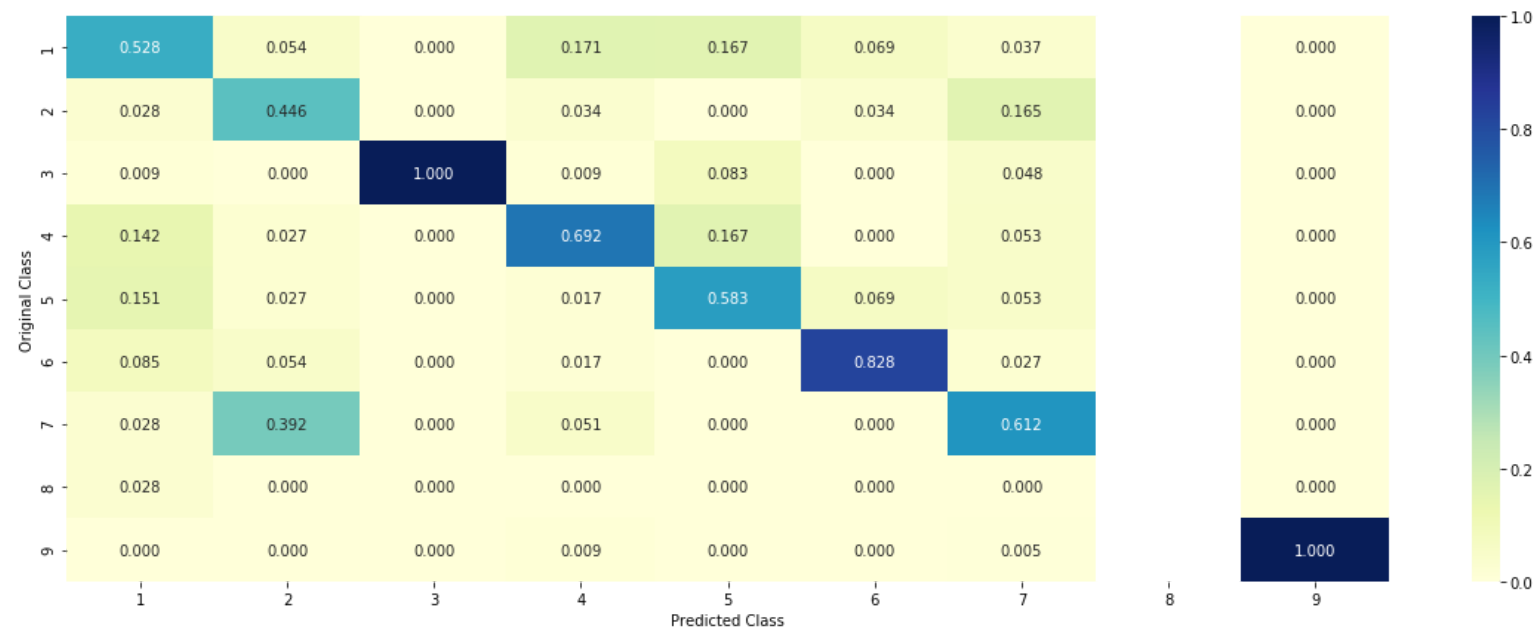
Number of mis-classified points : 0.39473684210526316

----- Confusion matrix -----





----- Precision matrix (Column Sum=1) -----



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



### 4.5.3. Feature Importance

#### 4.5.3.1. Correctly Classified point

```

1 # test_point_index = 10
2 clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_dep
3 clf.fit(train_x_onehotCoding, train_y)
4 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
5 sig_clf.fit(train_x_onehotCoding, train_y)
6
7 test_point_index = 1
8 no_feature = 100
9 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
10 print("Predicted Class :", predicted_cls[0])
11 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
12 print("Actual Class :", test_y[test_point_index])
13 indices = np.argsort(-clf.feature_importances_)
14 print("-"*50)
15 get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0454 0.1404 0.0133 0.029 0.036 0.0294 0.6977 0.005 0.004 ]]

Actual Class : 7

```

-----
0 Text feature [inhibitors] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
4 Text feature [missense] present in test data point [True]
5 Text feature [inhibitor] present in test data point [True]
7 Text feature [treatment] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
9 Text feature [suppressor] present in test data point [True]
10 Text feature [activation] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [kinases] present in test data point [True]
13 Text feature [nonsense] present in test data point [True]
14 Text feature [akt] present in test data point [True]
15 Text feature [function] present in test data point [True]
17 Text feature [erk] present in test data point [True]
19 Text feature [growth] present in test data point [True]
20 Text feature [variants] present in test data point [True]

```

22 Text feature [frameshift] present in test data point [True]  
24 Text feature [therapeutic] present in test data point [True]  
25 Text feature [functional] present in test data point [True]  
28 Text feature [signaling] present in test data point [True]  
30 Text feature [patients] present in test data point [True]  
31 Text feature [cells] present in test data point [True]  
32 Text feature [constitutively] present in test data point [True]  
34 Text feature [trials] present in test data point [True]  
35 Text feature [therapy] present in test data point [True]  
37 Text feature [erk1] present in test data point [True]  
38 Text feature [activate] present in test data point [True]  
39 Text feature [downstream] present in test data point [True]  
41 Text feature [efficacy] present in test data point [True]  
42 Text feature [protein] present in test data point [True]  
43 Text feature [loss] present in test data point [True]  
44 Text feature [inhibited] present in test data point [True]  
45 Text feature [expressing] present in test data point [True]  
46 Text feature [pten] present in test data point [True]  
48 Text feature [lines] present in test data point [True]  
49 Text feature [treated] present in test data point [True]  
50 Text feature [proliferation] present in test data point [True]  
51 Text feature [drug] present in test data point [True]  
57 Text feature [mek] present in test data point [True]  
59 Text feature [inhibition] present in test data point [True]  
61 Text feature [repair] present in test data point [True]  
62 Text feature [sensitivity] present in test data point [True]  
64 Text feature [receptor] present in test data point [True]  
66 Text feature [assays] present in test data point [True]  
68 Text feature [survival] present in test data point [True]  
69 Text feature [cell] present in test data point [True]  
71 Text feature [ligand] present in test data point [True]  
73 Text feature [expression] present in test data point [True]  
74 Text feature [variant] present in test data point [True]  
75 Text feature [oncogene] present in test data point [True]  
78 Text feature [extracellular] present in test data point [True]  
79 Text feature [doses] present in test data point [True]  
80 Text feature [mapk] present in test data point [True]  
81 Text feature [hours] present in test data point [True]  
84 Text feature [information] present in test data point [True]  
86 Text feature [harboring] present in test data point [True]  
90 Text feature [dna] present in test data point [True]  
91 Text feature [concentrations] present in test data point [True]  
92 Text feature [likelihood] present in test data point [True]

```
93 Text feature [months] present in test data point [True]
94 Text feature [binding] present in test data point [True]
96 Text feature [imatinib] present in test data point [True]
98 Text feature [preclinical] present in test data point [True]
Out of the top 100 features 65 are present in query point
```

#### 4.5.3.2. Inorrectly Classified point

```

1 test_point_index = 100
2 no_feature = 100
3 predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
4 print("Predicted Class :", predicted_cls[0])
5 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_i
6 print("Actual Class :", test_y[test_point_index])
7 indices = np.argsort(-clf.feature_importances_)
8 print("-"*50)
9 get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index], test_df['Gene'].iloc[

```

Predicted Class : 7

Predicted Class Probabilities: [[0.1337 0.116 0.0224 0.1773 0.0674 0.0545 0.4156 0.0071 0.0059]]

Actual Class : 7

```

-----
0 Text feature [inhibitors] present in test data point [True]
1 Text feature [kinase] present in test data point [True]
2 Text feature [activating] present in test data point [True]
3 Text feature [tyrosine] present in test data point [True]
6 Text feature [activated] present in test data point [True]
8 Text feature [oncogenic] present in test data point [True]
10 Text feature [activation] present in test data point [True]
11 Text feature [phosphorylation] present in test data point [True]
12 Text feature [kinases] present in test data point [True]
14 Text feature [akt] present in test data point [True]
15 Text feature [function] present in test data point [True]
19 Text feature [growth] present in test data point [True]
21 Text feature [constitutive] present in test data point [True]
25 Text feature [functional] present in test data point [True]
28 Text feature [signaling] present in test data point [True]
31 Text feature [cells] present in test data point [True]
32 Text feature [constitutively] present in test data point [True]
38 Text feature [activate] present in test data point [True]
39 Text feature [downstream] present in test data point [True]
42 Text feature [protein] present in test data point [True]
43 Text feature [loss] present in test data point [True]
44 Text feature [inhibited] present in test data point [True]
46 Text feature [pten] present in test data point [True]
47 Text feature [transforming] present in test data point [True]
48 Text feature [lines] present in test data point [True]

```

```
50 Text feature [proliferation] present in test data point [True]
53 Text feature [neutral] present in test data point [True]
55 Text feature [transform] present in test data point [True]
56 Text feature [stability] present in test data point [True]
58 Text feature [transformation] present in test data point [True]
59 Text feature [inhibition] present in test data point [True]
62 Text feature [sensitivity] present in test data point [True]
64 Text feature [receptor] present in test data point [True]
66 Text feature [assays] present in test data point [True]
69 Text feature [cell] present in test data point [True]
75 Text feature [oncogene] present in test data point [True]
84 Text feature [information] present in test data point [True]
90 Text feature [dna] present in test data point [True]
94 Text feature [binding] present in test data point [True]
Out of the top 100 features 39 are present in query point
```

### 4.5.3. Hyper paramter tuning (With Response Coding)

```

1  # -----
2  # default parameters
3  # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample:
4  # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
5  # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, wari
6  # class_weight=None)
7
8  # Some of methods of RandomForestClassifier()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # predict_proba (X) Perform classification on samples in X.
12
13 # some of attributes of RandomForestClassifier()
14 # feature_importances_ : array of shape = [n_features]
15 # The feature importances (the higher, the more important the feature).
16
17 # -----
18 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
19 # -----
20
21
22 # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn
23 # -----
24 # default paramters
25 # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
26 #
27 # some of the methods of CalibratedClassifierCV()
28 # fit(X, y[, sample_weight])    Fit the calibrated model
29 # get_params([deep])    Get parameters for this estimator.
30 # predict(X)    Predict the target of new samples.
31 # predict_proba(X) Posterior probabilities of classification
32 #-----
33 # video link:
34 #-----

```



```

35
36 alpha = [10,50,100,200,500,1000]
37 max_depth = [2,3,5,10]
38 cv_log_error_array = []
39 for i in alpha:
40     for j in max_depth:
41         print("for n_estimators =", i,"and max depth = ", j)
42         clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_state=42, n_
43         clf.fit(train_x_responseCoding, train_y)
44         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
45         sig_clf.fit(train_x_responseCoding, train_y)
46         sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
47         cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
48         print("Log Loss :",log_loss(cv_y, sig_clf_probs))
49     ''
50 fig, ax = plt.subplots()
51 features = np.dot(np.array(alpha)[: ,None],np.array(max_depth)[None]).ravel()
52 ax.plot(features, cv_log_error_array,c='g')
53 for i, txt in enumerate(np.round(cv_log_error_array,3)):
54     ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_array[i]))
55 plt.grid()
56 plt.title("Cross Validation Error for each alpha")
57 plt.xlabel("Alpha i's")
58 plt.ylabel("Error measure")
59 plt.show()
60 ''
61
62 best_alpha = np.argmin(cv_log_error_array)
63 clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_dept
64 clf.fit(train_x_responseCoding, train_y)
65 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
66 sig_clf.fit(train_x_responseCoding, train_y)
67
68 predict_y = sig_clf.predict_proba(train_x_responseCoding)
69 print('For values of best alpha = ', alpha[int(best_alpha/4)], "The train log loss is:",log_loss(y_train

```

```
70 predict_y = sig_clf.predict_proba(cv_x_responseCoding)
71 print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation log loss is:", log_
72 predict_y = sig_clf.predict_proba(test_x_responseCoding)
73 print('For values of best alpha = ', alpha[int(best_alpha/4)], "The test log loss is:", log_loss(y_test,
```

```
for n_estimators = 10 and max depth = 2
Log Loss : 2.118930028859171
for n_estimators = 10 and max depth = 3
Log Loss : 1.519797481402448
for n_estimators = 10 and max depth = 5
Log Loss : 1.4412657373720836
for n_estimators = 10 and max depth = 10
Log Loss : 1.5420579985274587
for n_estimators = 50 and max depth = 2
Log Loss : 1.8673316601419094
for n_estimators = 50 and max depth = 3
Log Loss : 1.3792135049794187
for n_estimators = 50 and max depth = 5
Log Loss : 1.2089200148091979
for n_estimators = 50 and max depth = 10
Log Loss : 1.405652487089799
for n_estimators = 100 and max depth = 2
Log Loss : 1.8315025008452632
for n_estimators = 100 and max depth = 3
Log Loss : 1.3902247140947537
for n_estimators = 100 and max depth = 5
Log Loss : 1.1776582944530516
for n_estimators = 100 and max depth = 10
Log Loss : 1.3767945798702477
for n_estimators = 200 and max depth = 2
Log Loss : 1.7424385014220518
for n_estimators = 200 and max depth = 3
Log Loss : 1.4321584504562903
for n_estimators = 200 and max depth = 5
Log Loss : 1.1817673871718715
for n_estimators = 200 and max depth = 10
Log Loss : 1.3790254662509127
for n_estimators = 500 and max depth = 2
Log Loss : 1.5668300633238137
for n_estimators = 500 and max depth = 3
Log Loss : 1.41500248517593
for n_estimators = 500 and max depth = 5
```

```
Log Loss : 1.1830541377782244
for n_estimators = 500 and max depth = 10
Log Loss : 1.3484808093685314
for n_estimators = 1000 and max depth = 2
Log Loss : 1.529955488349422
for n_estimators = 1000 and max depth = 3
Log Loss : 1.3701025070903536
for n_estimators = 1000 and max depth = 5
Log Loss : 1.1608286154615162
for n_estimators = 1000 and max depth = 10
Log Loss : 1.3605354325550236
For values of best alpha = 1000 The train log loss is: 0.05350747269088699
For values of best alpha = 1000 The cross validation log loss is: 1.1608286154615164
For values of best alpha = 1000 The test log loss is: 1.1065684015049262
```

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

```

1  # -----
2  # default parameters
3  # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample:
4  # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
5  # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, wari
6  # class_weight=None)
7
8  # Some of methods of RandomForestClassifier()
9  # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
10 # predict(X)    Perform classification on samples in X.
11 # predict_proba (X) Perform classification on samples in X.
12
13 # some of attributes of RandomForestClassifier()
14 # feature_importances_ : array of shape = [n_features]
15 # The feature importances (the higher, the more important the feature).
16
17 # -----
18 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
19 # -----
20
21 clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=alpha[int(best_alpha/4)],
22 predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCoding,cv_y, clf)

```

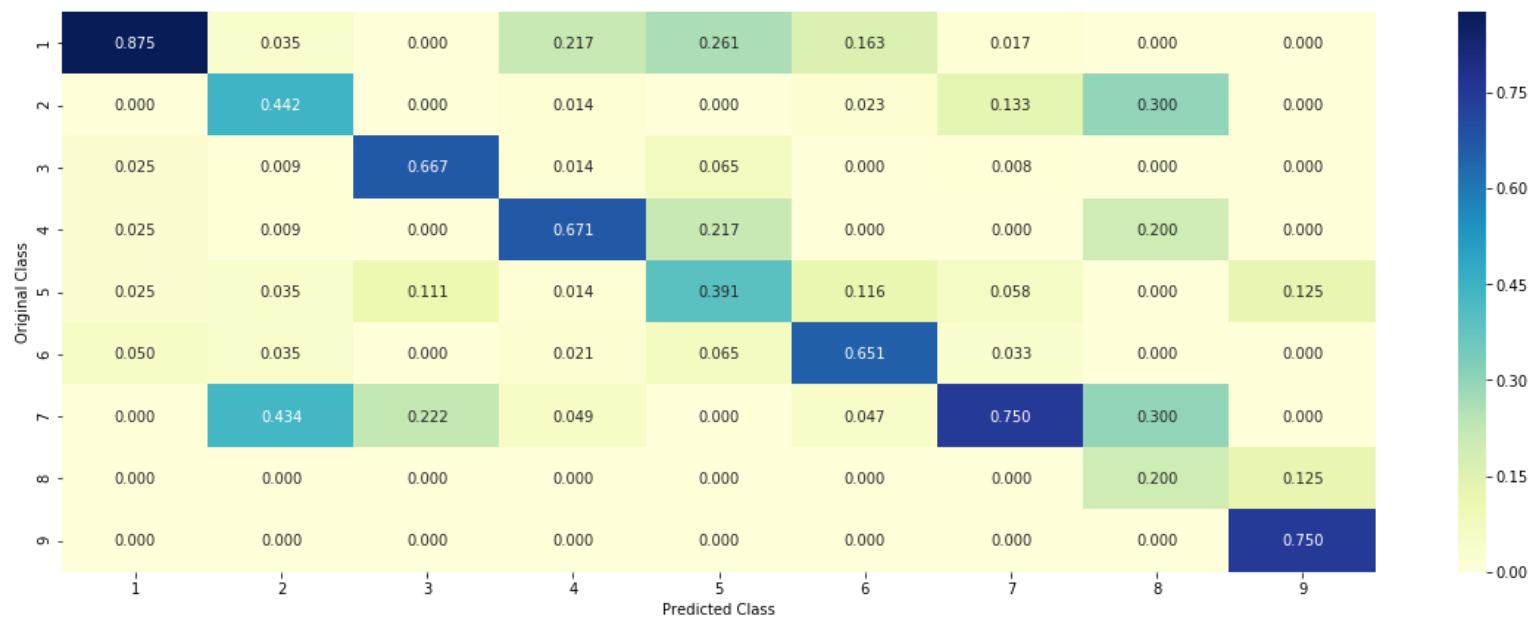
Log loss : 1.1608286154615164

Number of mis-classified points : 0.37781954887218044

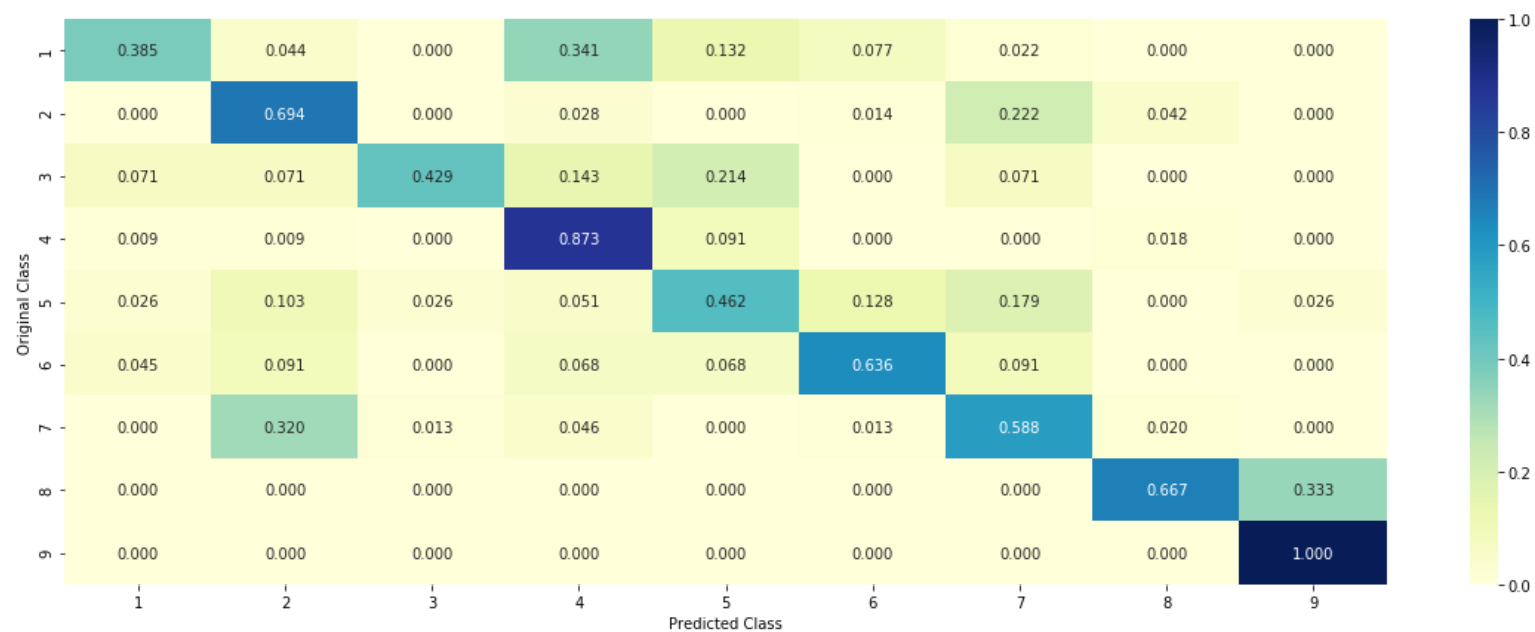
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



## 4.5.5. Feature Importance

### 4.5.5.1. Correctly Classified point



```

1  clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_dep
2  clf.fit(train_x_responseCoding, train_y)
3  sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
4  sig_clf.fit(train_x_responseCoding, train_y)
5
6
7  test_point_index = 1
8  no_feature = 27
9  predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
10 print("Predicted Class :", predicted_cls[0])
11 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point
12 print("Actual Class :", test_y[test_point_index])
13 indices = np.argsort(-clf.feature_importances_)
14 print("-"*50)
15 for i in indices:
16     if i<9:
17         print("Gene is important feature")
18     elif i<18:
19         print("Variation is important feature")
20     else:
21         print("Text is important feature")

```

Predicted Class : 2

Predicted Class Probabilities: [[0.0143 0.5044 0.1471 0.0191 0.0245 0.065 0.1724 0.039 0.0142]]

Actual Class : 7

```

-----
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Text is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Text 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  
Text is important feature  
Gene is important feature  
Gene 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  
Gene is important feature  
Gene is important feature

#### 4.5.5.2. Incorrectly Classified point

```

1 test_point_index = 100
2 predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
3 print("Predicted Class :", predicted_cls[0])
4 print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_
5 print("Actual Class :", test_y[test_point_index])
6 indices = np.argsort(-clf.feature_importances_)
7 print("-"*50)
8 for i in indices:
9     if i<9:
10         print("Gene is important feature")
11     elif i<18:
12         print("Variation is important feature")
13     else:
14         print("Text is important feature")

```

Predicted Class : 7

Predicted Class Probabilities: [[0.0281 0.2006 0.203 0.0857 0.0626 0.0906 0.2249 0.0676 0.0369]]

Actual Class : 7

-----

Variation is important feature  
 Variation is important feature  
 Variation is important feature  
 Variation is important feature  
 Text is important feature  
 Variation is important feature  
 Gene is important feature  
 Variation is important feature  
 Text 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  
 Text is important feature  
 Gene is important feature  
 Gene 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  
Gene is important feature  
Gene is important feature
```

## 4.7 Stack the models

### 4.7.1 testing with hyper parameter tuning

```

1  # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
2  # -----
3  # default parameters
4  # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000,
5  # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0,
6  # class_weight=None, warm_start=False, average=False, n_iter=None)
7
8  # some of methods
9  # fit(X, y[, coef_init, intercept_init, ...]) Fit linear model with Stochastic Gradient Descent.
10 # predict(X)    Predict class labels for samples in X.
11
12 #-----
13 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-for-linear-models
14 #-----
15
16
17 # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
18 # -----
19 # default parameters
20 # SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking=True, probability=False, tol=0.001,
21 # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None)
22
23 # Some of methods of SVM()
24 # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
25 # predict(X)    Perform classification on samples in X.
26 # -----
27 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivatives-of-svm
28 # -----
29
30
31 # read more about support vector machines with linear kernels here http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
32 # -----
33 # default parameters
34 # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None, min_sample_size=10,

```

```

35 # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impur
36 # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, wari
37 # class_weight=None)
38
39 # Some of methods of RandomForestClassifier()
40 # fit(X, y, [sample_weight])    Fit the SVM model according to the given training data.
41 # predict(X)    Perform classification on samples in X.
42 # predict_proba (X) Perform classification on samples in X.
43
44 # some of attributes of RandomForestClassifier()
45 # feature_importances_ : array of shape = [n_features]
46 # The feature importances (the higher, the more important the feature).
47
48 # -----
49 # video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and
50 # -----
51
52
53 clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=0)
54 clf1.fit(train_x_onehotCoding, train_y)
55 sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")
56
57 clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=0)
58 clf2.fit(train_x_onehotCoding, train_y)
59 sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
60
61
62 clf3 = MultinomialNB(alpha=0.001)
63 clf3.fit(train_x_onehotCoding, train_y)
64 sig_clf3 = CalibratedClassifierCV(clf3, method="sigmoid")
65
66 sig_clf1.fit(train_x_onehotCoding, train_y)
67 print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCodi
68 sig_clf2.fit(train_x_onehotCoding, train_y)
69 print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotC

```

```

70 sig_clf3.fit(train_x_onehotCoding, train_y)
71 print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
72 print("-"*50)
73 alpha = [0.0001,0.001,0.01,0.1,1,10]
74 best_alpha = 999
75 for i in alpha:
76     lr = LogisticRegression(C=i)
77     sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_proba:
78     sclf.fit(train_x_onehotCoding, train_y)
79     print("Stacking Classifier : for the value of alpha: %f Log Loss: %0.3f" % (i, log_loss(cv_y, sclf.pr
80     log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
81     if best_alpha > log_error:
82         best_alpha = log_error

```

Logistic Regression : Log Loss: 1.02

Support vector machines : Log Loss: 1.78

Naive Bayes : Log Loss: 1.18

-----

Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178

Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.033

Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.503

Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.145

Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.306

Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.684

## 4.7.2 testing the model with the best hyper parameters

```
1 lr = LogisticRegression(C=0.1)
2 sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifier=lr, use_probabilities=True)
3 sclf.fit(train_x_onehotCoding, train_y)
4
5 log_error = log_loss(train_y, sclf.predict_proba(train_x_onehotCoding))
6 print("Log loss (train) on the stacking classifier :", log_error)
7
8 log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
9 print("Log loss (CV) on the stacking classifier :", log_error)
10
11 log_error = log_loss(test_y, sclf.predict_proba(test_x_onehotCoding))
12 print("Log loss (test) on the stacking classifier :", log_error)
13
14 print("Number of misclassified points :", np.count_nonzero((sclf.predict(test_x_onehotCoding) - test_y))),
15 plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_onehotCoding))
```

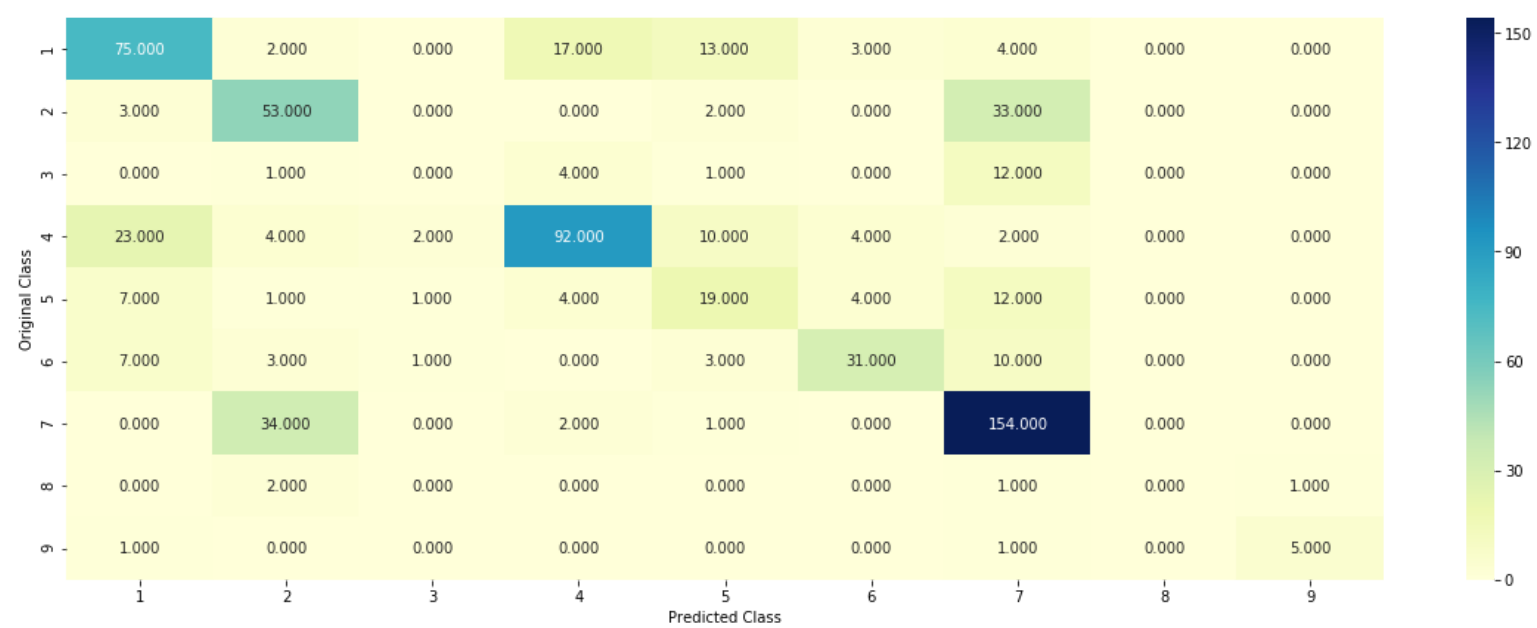
Log loss (train) on the stacking classifier : 0.5913619788986474

Log loss (CV) on the stacking classifier : 1.1453012283175468

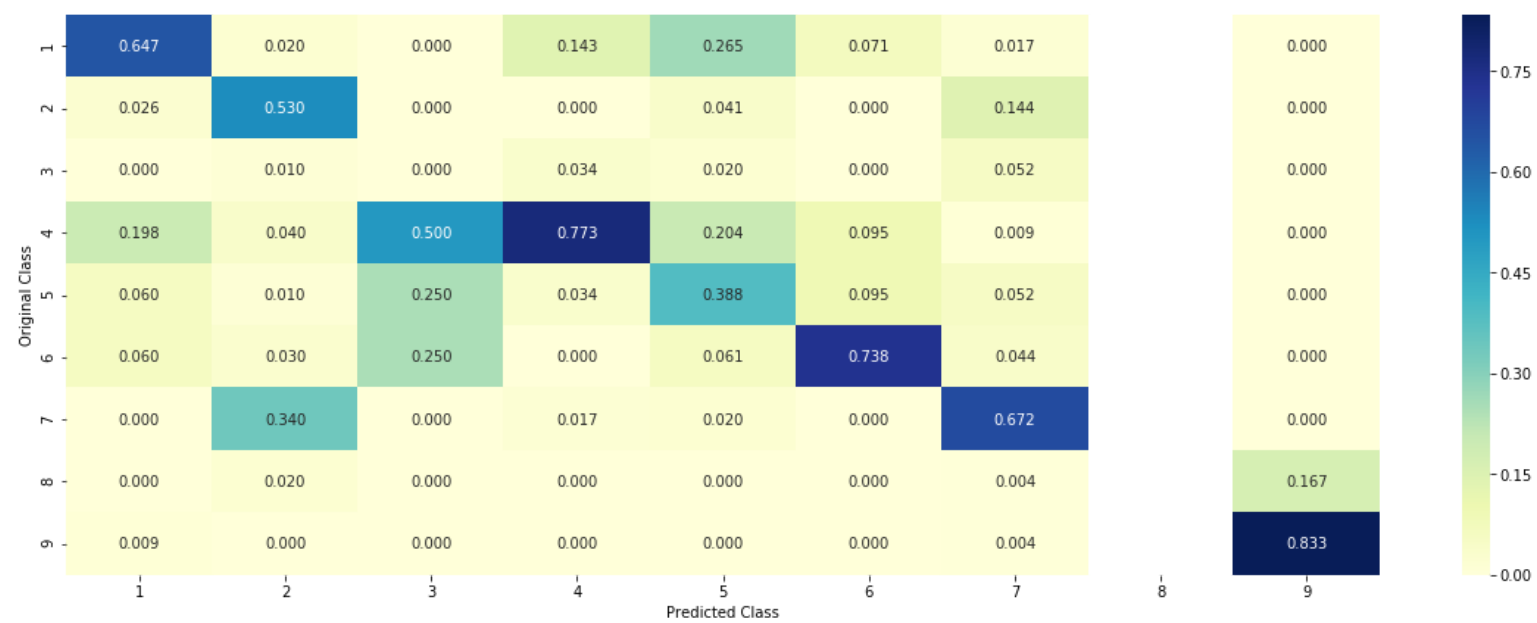
Log loss (test) on the stacking classifier : 1.0937282801519348

Number of misclassified points : 0.3548872180451128

----- Confusion matrix -----

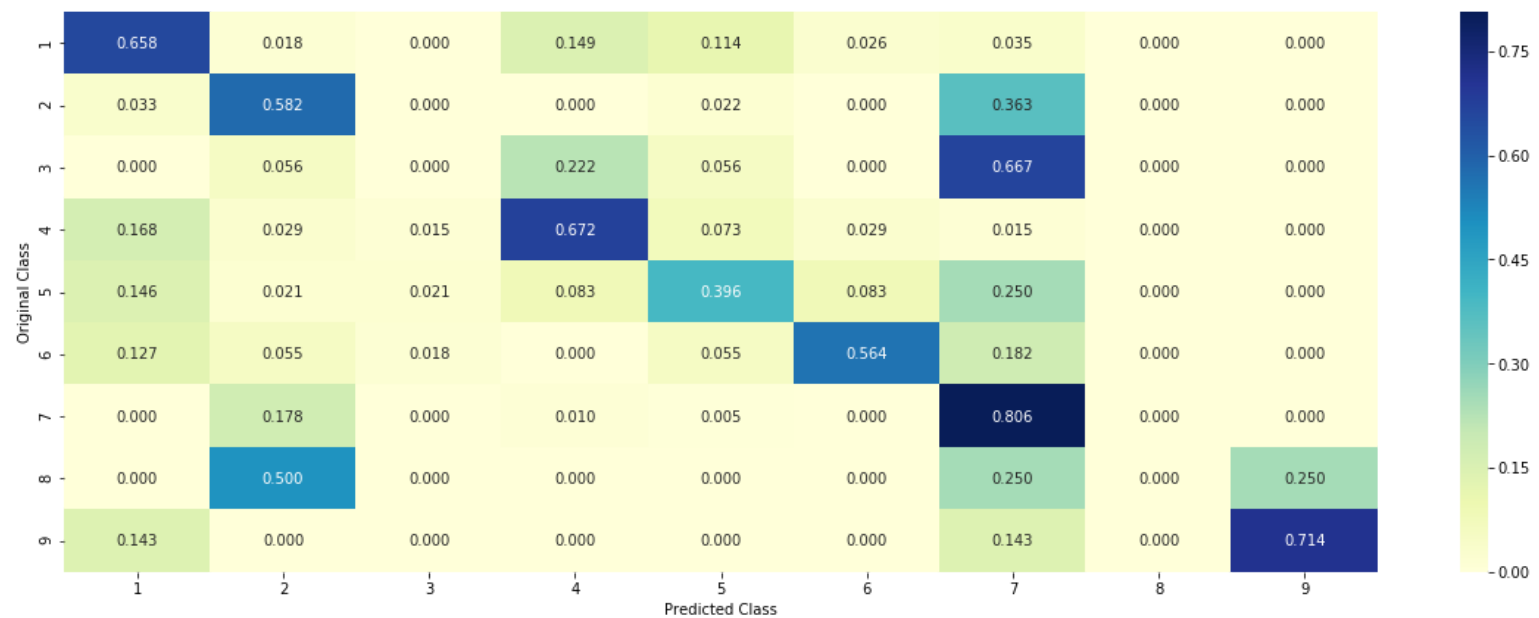


----- Precision matrix (Column Sum=1) -----



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





### 4.7.3 Maximum Voting classifier

```

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

```

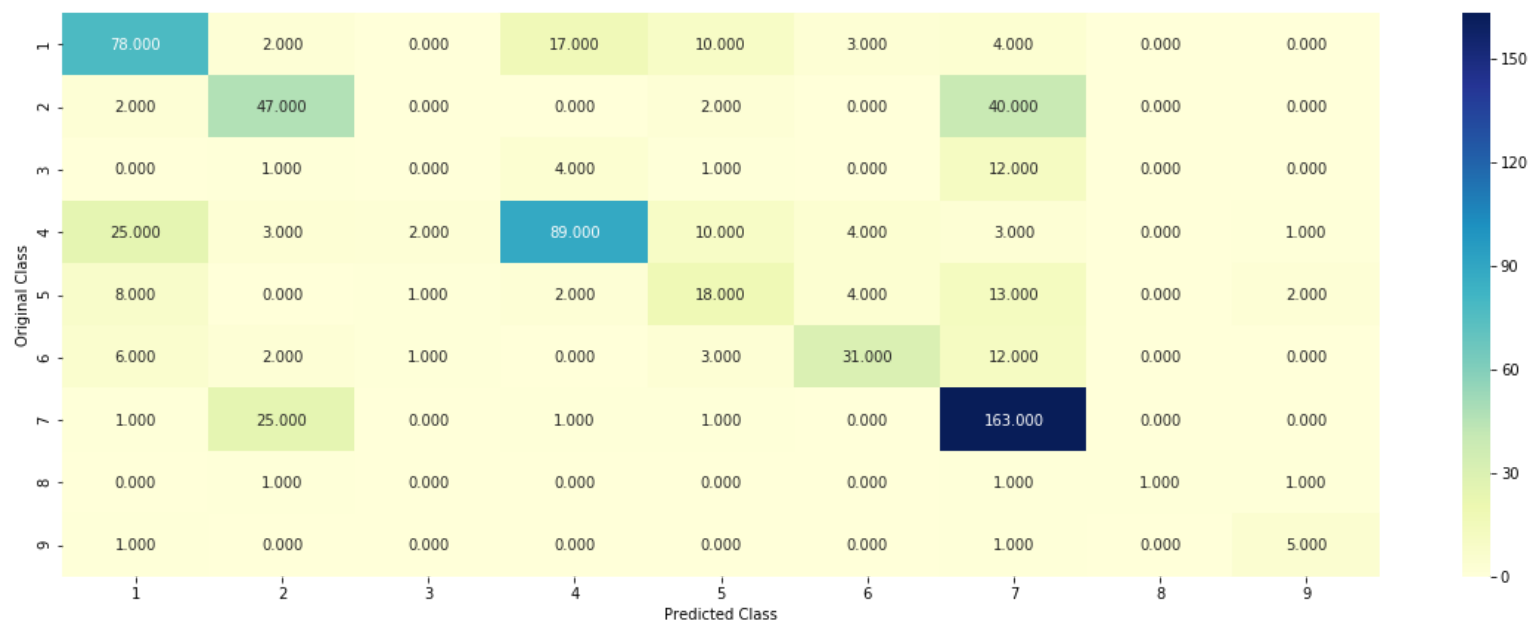
Log loss (train) on the VotingClassifier : 0.8525530164599718

Log loss (CV) on the VotingClassifier : 1.1478941474355593

Log loss (test) on the VotingClassifier : 1.147009573669618

Number of missclassified point : 0.35037593984962406

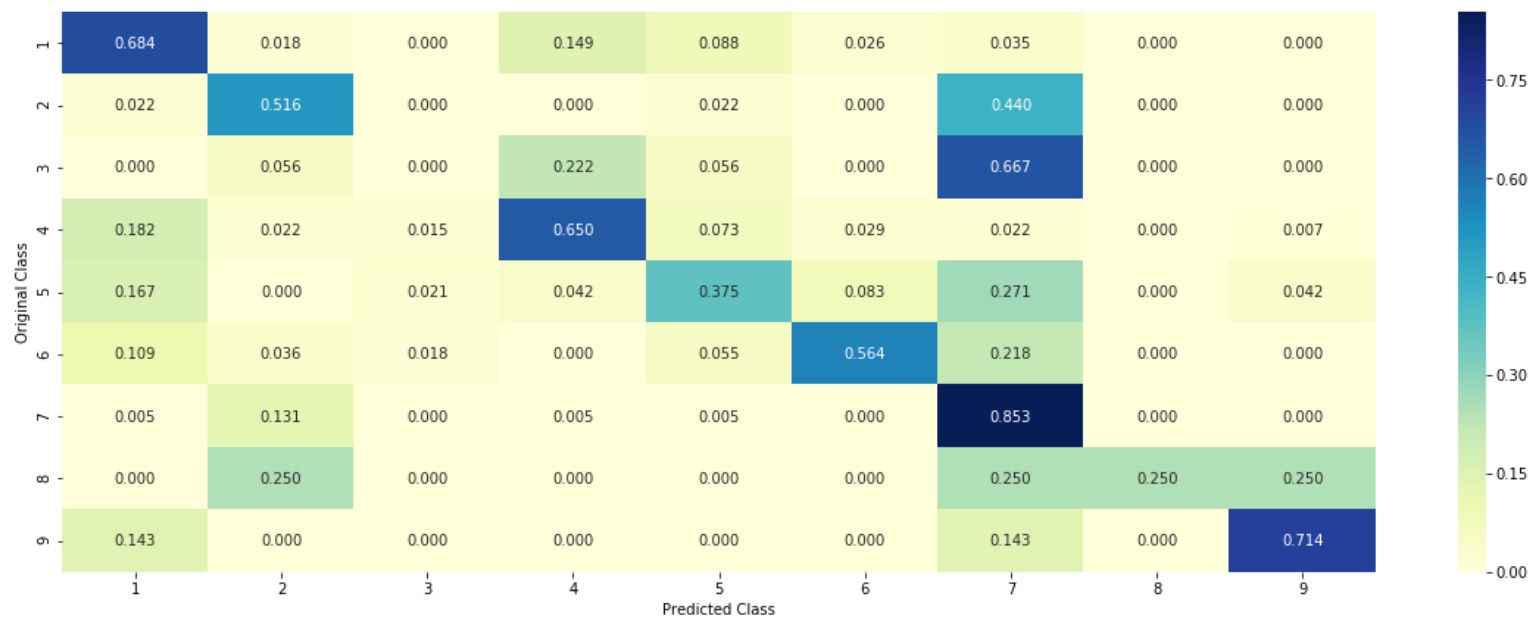
----- Confusion matrix -----



----- Precision matrix (Column Sum=1) -----



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



## Summary

```

1  from prettytable import PrettyTable
2  x=PrettyTable()
3  x.field_names = ["Model","Train loss", "CV loss", "Test Loss", "Mis-classified pts"]
4  x.add_row(["NB with one-hot encoding",0.5465,1.1694,1.1550,0.3872])
5  x.add_row(["KNN with response coding",0.9863,1.1907,1.1711,0.4154])
6  x.add_row(["LR(class balancing) with one hot encoding",0.4320,0.9708,0.9376,0.3157])
7  x.add_row(["LR(without class balancing) with one hot encoding",0.4229,0.9727,0.9331,0.3176])
8  x.add_row(["SVM with one hot encoding",0.4846,0.9493,0.4869,0.3421])
9  x.add_row(["RF with one hot encoding",0.8626,1.1789,1.1581,0.3947])
10 x.add_row(["RF with response coding",0.5350,1.1608,1.1065,0.3778])
11 x.add_row(["Stacked classifier(LR+NB+SVM)",0.5913,1.1453,1.0937,0.3548])
12 x.add_row(["Maximum Voting classifier(LR+SVC+RF)",0.8525,1.1478,1.1470,0.3503])
13
14 print(x)

```

Model	Train loss	CV loss	Test Loss	Mis-classified pts
NB with one-hot encoding	0.5465	1.1694	1.155	0.3872
KNN with response coding	0.9863	1.1907	1.1711	0.4154
LR(class balancing) with one hot encoding	0.432	0.9708	0.9376	0.3157
LR(without class balancing) with one hot encoding	0.4229	0.9727	0.9331	0.3176
SVM with one hot encoding	0.4846	0.9493	0.4869	0.3421
RF with one hot encoding	0.8626	1.1789	1.1581	0.3947
RF with response coding	0.535	1.1608	1.1065	0.3778
Stacked classifier(LR+NB+SVM)	0.5913	1.1453	1.0937	0.3548
Maximum Voting classifier(LR+SVC+RF)	0.8525	1.1478	1.147	0.3503

Our Best model is LR(class balancing) with one hot encoding with 0.9376 log loss and 31.57% Mis-classification rate.

