

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>

2. <https://www.youtube.com/watch?v=UwbuW7oK8rk>
3. <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>
- 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>

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

```
In [7]: !pip install nltk
!pip install mlxtend
!pip install seaborn
!pip install imblearn
```

```
Requirement already satisfied: nltk in /opt/conda/envs/py3.6/lib/python
3.6/site-packages (3.3)
```

```
Requirement already satisfied: six in /opt/conda/envs/py3.6/lib/python
3.6/site-packages (from nltk) (1.11.0)
```

```
jupyter 1.0.0 requires qtconsole, which is not installed.
```

```
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
have widgetsnbextension 3.2.1 which is incompatible.
```

```
You are using pip version 10.0.1, however version 18.0 is available.
```

```
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
```

```
Requirement already satisfied: mlxtend in /opt/conda/envs/py3.6/lib/pyt
hon3.6/site-packages (0.13.0)
```

```
Requirement already satisfied: pandas>=0.17.1 in /opt/conda/envs/py3.6/
lib/python3.6/site-packages (from mlxtend) (0.20.3)
```

```
Requirement already satisfied: scikit-learn>=0.18 in /opt/conda/envs/py
```

```
3.6/lib/python3.6/site-packages (from mlxtend) (0.19.0)
Requirement already satisfied: scipy>=0.17 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from mlxtend) (0.19.1)
Requirement already satisfied: matplotlib>=1.5.1 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from mlxtend) (2.1.2)
Requirement already satisfied: numpy>=1.10.4 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from mlxtend) (1.12.1)
Requirement already satisfied: setuptools in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from mlxtend) (36.4.0)
Requirement already satisfied: python-dateutil>=2 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from pandas>=0.17.1->mlxtend) (2.7.3)
Requirement already satisfied: pytz>=2011k in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from pandas>=0.17.1->mlxtend) (2018.4)
Requirement already satisfied: six>=1.10 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (1.11.0)
Requirement already satisfied: cyclur>=0.10 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>=1.5.1->mlxtend) (2.2.0)
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll have widgetsnbextension 3.2.1 which is incompatible.
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Requirement already satisfied: seaborn in /opt/conda/envs/py3.6/lib/python3.6/site-packages (0.9.0)
Requirement already satisfied: numpy>=1.9.3 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from seaborn) (1.12.1)
Requirement already satisfied: pandas>=0.15.2 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from seaborn) (0.20.3)
Requirement already satisfied: matplotlib>=1.4.3 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from seaborn) (2.1.2)
Requirement already satisfied: scipy>=0.14.0 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from seaborn) (0.19.1)
Requirement already satisfied: python-dateutil>=2 in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from pandas>=0.15.2->seaborn) (2.7.3)
Requirement already satisfied: pytz>=2011k in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from pandas>=0.15.2->seaborn) (2018.4)
```

```

b/python3.6/site-packages (from pandas>=0.15.2->seaborn) (2018.4)
Requirement already satisfied: six>=1.10 in /opt/conda/envs/py3.6/lib/p
ython3.6/site-packages (from matplotlib>=1.4.3->seaborn) (1.11.0)
Requirement already satisfied: cyclor>=0.10 in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from matplotlib>=1.4.3->seaborn) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1
in /opt/conda/envs/py3.6/lib/python3.6/site-packages (from matplotlib>
=1.4.3->seaborn) (2.2.0)
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
have widgetsnbextension 3.2.1 which is incompatible.
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.
Requirement already satisfied: imblearn in /opt/conda/envs/py3.6/lib/py
thon3.6/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /opt/conda/envs/py3.
6/lib/python3.6/site-packages (from imblearn) (0.3.3)
Requirement already satisfied: numpy in /opt/conda/envs/py3.6/lib/pytho
n3.6/site-packages (from imbalanced-learn->imblearn) (1.12.1)
Requirement already satisfied: scipy in /opt/conda/envs/py3.6/lib/pytho
n3.6/site-packages (from imbalanced-learn->imblearn) (0.19.1)
Requirement already satisfied: scikit-learn in /opt/conda/envs/py3.6/li
b/python3.6/site-packages (from imbalanced-learn->imblearn) (0.19.0)
jupyter 1.0.0 requires qtconsole, which is not installed.
ipywidgets 7.0.3 has requirement widgetsnbextension~=3.0.0, but you'll
have widgetsnbextension 3.2.1 which is incompatible.
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' comma
nd.

```

```

In [8]: import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import numpy as np
import nltk
from nltk.corpus import stopwords

```

```

from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from imblearn.over_sampling import SMOTE
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier
warnings.filterwarnings("ignore")

from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
nltk.download('stopwords')

```

```

[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.

```

Out[8]: True

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
In [3]: data = pd.read_csv('training_variants')
print('Number of data points : ', data.shape[0])
print('Number of features : ', data.shape[1])
print('Features : ', data.columns.values)
data.head()
```

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

Out[3]:

	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

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

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

Out[4]:

	ID	TEXT
0	0	Cyclin-dependent kinases (CDKs) regulate a var...
1	1	Abstract Background Non-small cell lung canc...
2	2	Abstract Background Non-small cell lung canc...
3	3	Recent evidence has demonstrated that acquired...
4	4	Oncogenic mutations in the monomeric Casitas B...

3.1.3. Preprocessing of text

```
In [9]: # loading stop words from nltk library
stop_words = set(stopwords.words('english'))

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

```

total_text = re.sub('\s+', ' ', total_text)
# converting all the chars into lower-case.
total_text = total_text.lower()

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

data_text[column][index] = string

```

```

In [10]: #text processing stage.
start_time = time.clock()
for index, row in data_text.iterrows():
    nlp_preprocessing(row['TEXT'], index, 'TEXT')
print('Time took for preprocessing the text :', time.clock() - start_time, "seconds")

```

Time took for preprocessing the text : 153.651573 seconds

```

In [11]: #merging both gene_variations and text data based on ID
result = pd.merge(data, data_text, on='ID', how='left')
result.head()

```

Out[11]:

	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...
4	4	CBL	L399V	4	oncogenic mutations monomeric casitas b lineag...

	ID	Gene	Variation	Class	TEXT
--	----	------	-----------	-------	------

3.1.4. Test, Train and Cross Validation Split

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

```
In [12]: y_true = result['Class'].values
result.Gene      = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')

# split the data into test and train by maintaining same distribution o
f output variable 'y_true' [stratify=y_true]
X_train, test_df, y_train, y_test = train_test_split(result, y_true, st
ratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining s
ame distribution of output variable 'y_train' [stratify=y_train]
train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, str
atify=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

```
In [13]: print('Number of data points in train data:', train_df.shape[0])
print('Number of data points in test data:', test_df.shape[0])
print('Number of data points in cross validation data:', cv_df.shape[0])
```

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

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

```
In [14]: # it returns a dict, keys as class labels and values as the number of d
ata points in that class
```

```

train_class_distribution = train_df['Class'].value_counts().sortlevel()
test_class_distribution = test_df['Class'].value_counts().sortlevel()
cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', train_class_distribution.values[i], '(', np.round((train_class_distribution.values[i]/train_df.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
test_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in test data')
plt.grid()
plt.show()

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_yi = np.argsort(-test_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ':', test_class_distrib

```

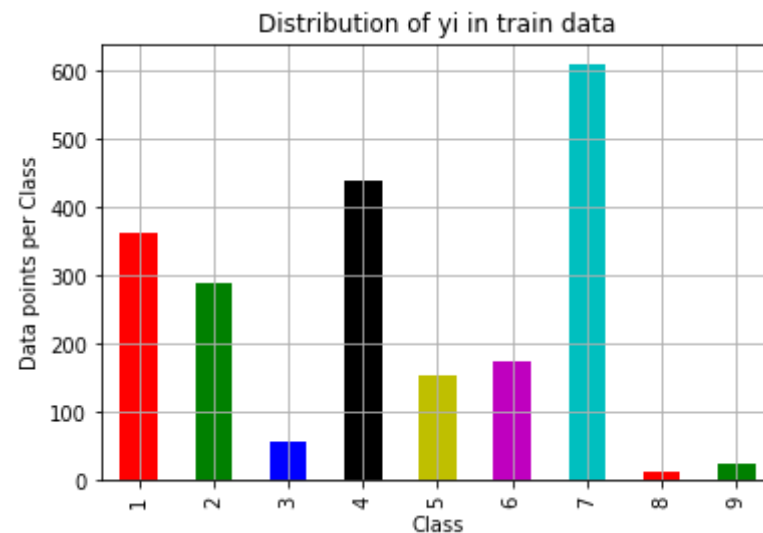
```

ution.values[i], '(', np.round((test_class_distribution.values[i]/test_
df.shape[0]*100), 3), '%)')

print('-'*80)
my_colors = 'rgbkymc'
cv_class_distribution.plot(kind='bar', color=my_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in cross validation data')
plt.grid()
plt.show()

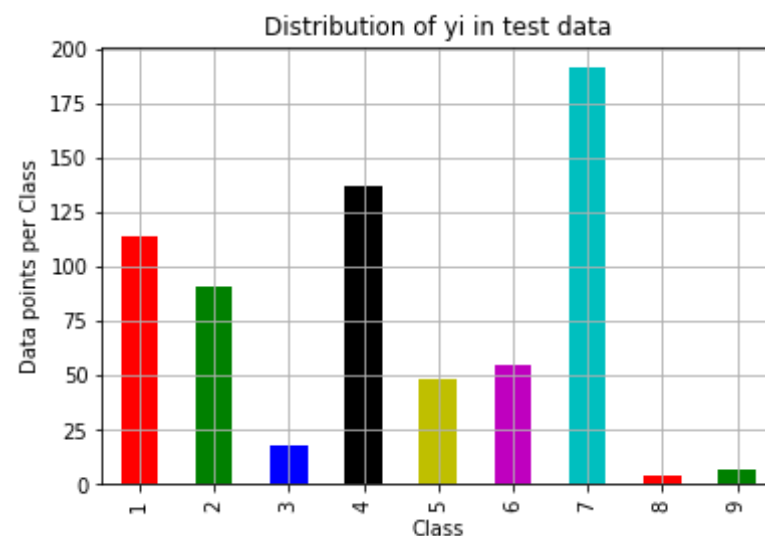
# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in de
creasing order
sorted_yi = np.argsort(-train_class_distribution.values)
for i in sorted_yi:
    print('Number of data points in class', i+1, ': ', cv_class_distribut
ion.values[i], '(', np.round((cv_class_distribution.values[i]/cv_df.sha
pe[0]*100), 3), '%)')

```

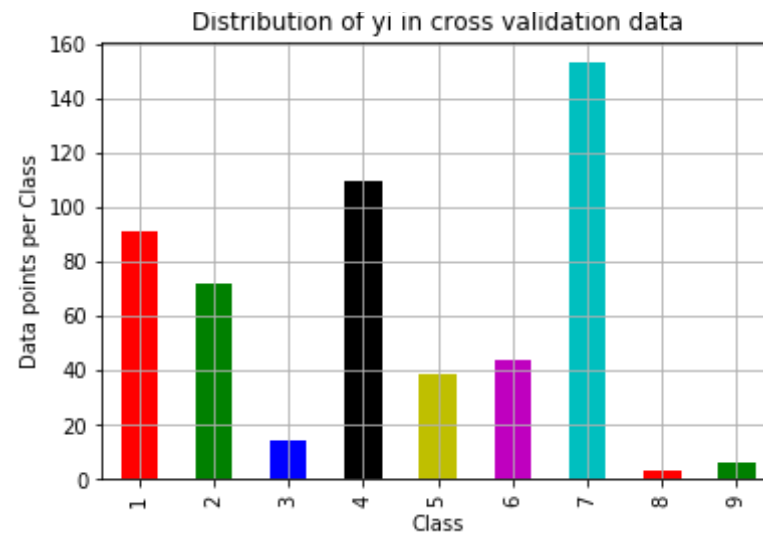


Number of data points in class 7 : 609 (28.672 %)

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.

```

In [15]: # This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i
    # are predicted class j
  
```



```

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

# C = [[1, 2],
#      [3, 4]]
# C.T = [[1, 3],
#        [2, 4]]
# C.sum(axis = 1) axis=0 corresonds to columns and axis=1 correspo
nds to rows in two dimensional array
# C.sum(axix =1) = [[3, 7]]
# ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
#                           [2/3, 4/7]]

# ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
#                               [3/7, 4/7]]
# sum of row elements = 1

B = (C/C.sum(axis=0))
#divid each element of the confusion matrix with the sum of element
s in that row
# C = [[1, 2],
#      [3, 4]]
# C.sum(axis = 0) axis=0 corresonds to columns and axis=1 correspo
nds to rows in two dimensional array
# C.sum(axix =0) = [[4, 6]]
# (C/C.sum(axis=0)) = [[1/4, 2/6],
#                      [3/4, 4/6]]

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

```

```

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

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

```

```

In [16]: # we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 9 numbers and divide each of the numbers
# by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

# we create a output array that has exactly same size as the CV data
cv_predicted_y = np.zeros((cv_data_len,9))
for i in range(cv_data_len):
    rand_probs = np.random.rand(1,9)
    cv_predicted_y[i] = ((rand_probs/sum(sum(rand_probs))))[0]
print("Log loss on Cross Validation Data using Random Model", log_loss(y_cv, cv_predicted_y, eps=1e-15))

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)

```

```

test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_p
redicted_y, eps=1e-15))

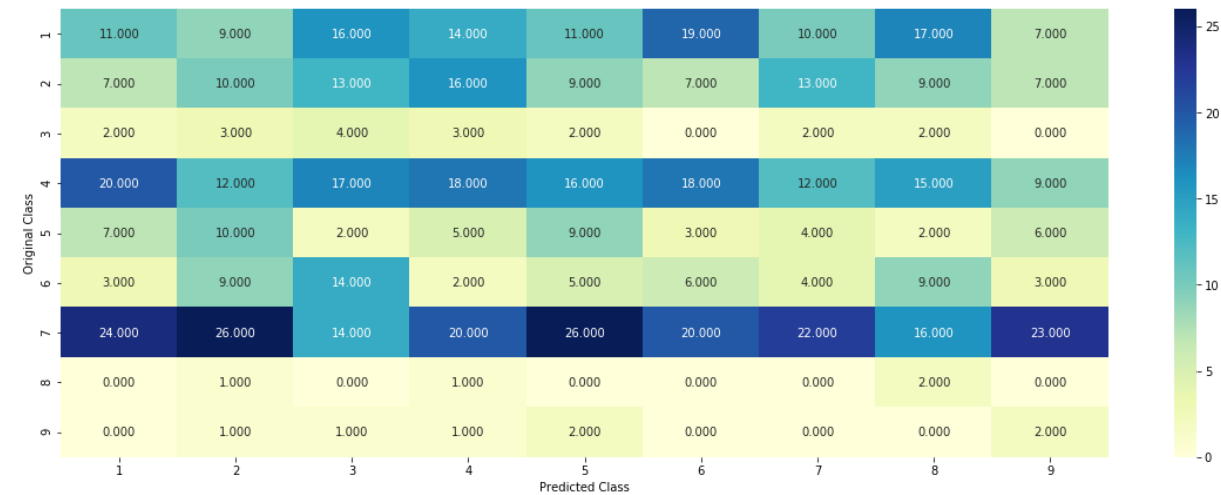
predicted_y =np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

Log loss on Cross Validation Data using Random Model 2.43267400611

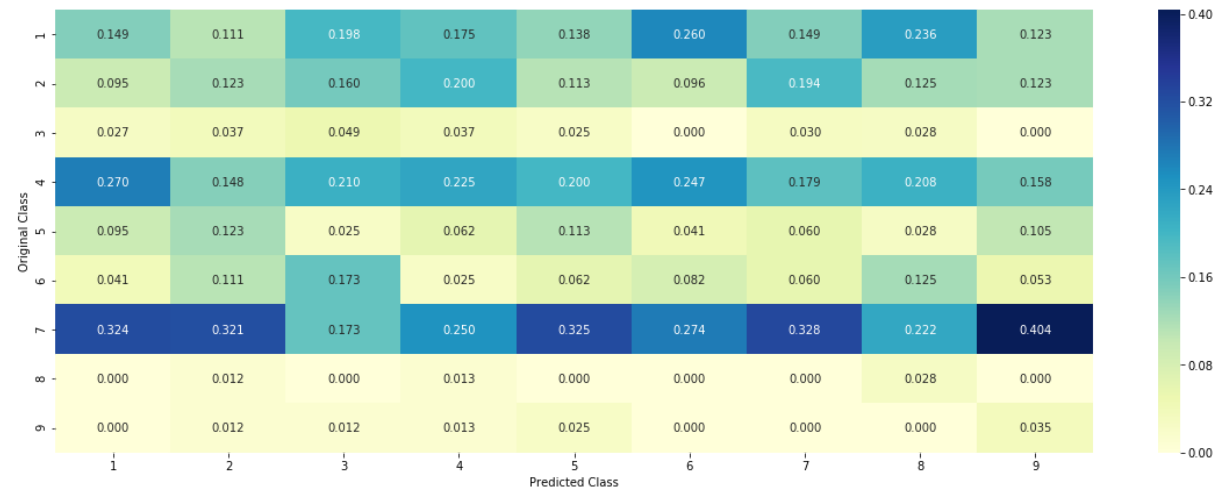
Log loss on Test Data using Random Model 2.44314697085

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

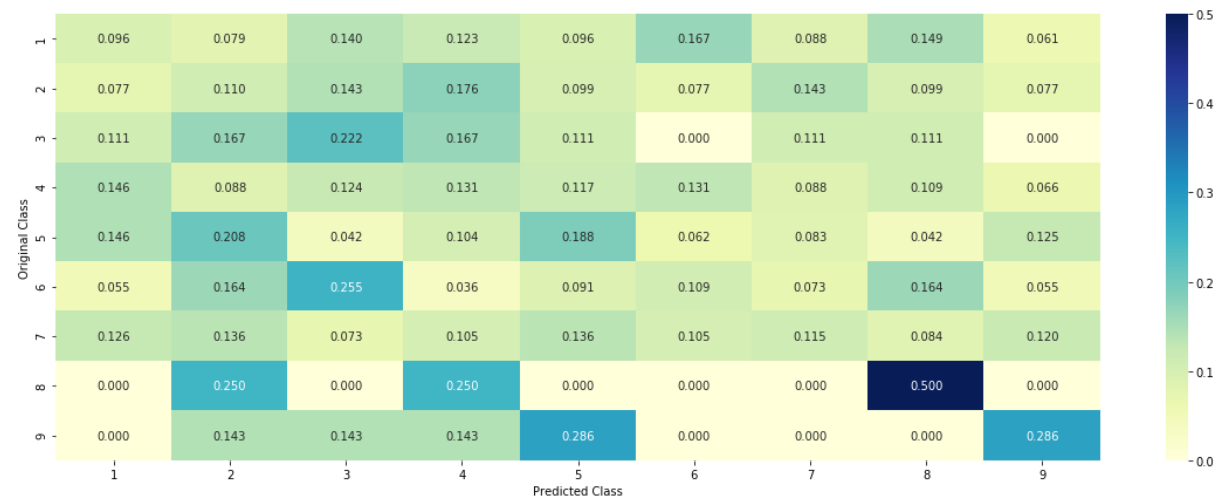


----- Precision matrix (Column Sum=1) -----

--



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



3.3 Univariate Analysis

```
In [17]: # code for response coding with Laplace smoothing.
          # alpha : used for laplace smoothing
```

```

# feature: ['gene', 'variation']
# df: ['train_df', 'test_df', 'cv_df']
# algorithm
# -----
# Consider all unique values and the number of occurrences of given feature in train data dataframe
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alpha / number of times it occurred in total data+90*alpha)
# gv_dict is like a look up table, for every gene it stores a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----

# get_gv_fea_dict: Get Gene variation Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #      {BRCA1      174
    #       TP53      106
    #       EGFR       86
    #       BRCA2       75
    #       PTEN       69
    #       KIT        61
    #       BRAF        60
    #       ERBB2       47
    #       PDGFRA      46
    #       ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations

```

63

```

# Deletion 43
# Amplification 43
# Fusions 22
# Overexpression 3
# E17K 3
# Q61L 3
# S222D 2
# P130S 2
# ...
# }
value_count = train_df[feature].value_counts()

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

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

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

```

```

        # cls_cnt.shape[0](numerator) will contain the number of ti
me that particular feature occurred in whole data
        vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90
*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181
8181818177, 0.13636363636363635, 0.25, 0.19318181818181818, 0.0378787
8787878788, 0.03787878787878788, 0.03787878787878788],
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224
489795918366, 0.27040816326530615, 0.061224489795918366, 0.066326530612
244902, 0.051020408163265307, 0.051020408163265307, 0.05612244897959183
7],
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625,
0.068181818181818177, 0.068181818181818177, 0.0625, 0.3465909090909091
2, 0.0625, 0.056818181818181816],
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.06060
60606060608, 0.078787878787878782, 0.1393939393939394, 0.345454545454
54546, 0.060606060606060608, 0.060606060606060608, 0.060606060606060
8],
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.06918
2389937106917, 0.46540880503144655, 0.075471698113207544, 0.06289308176
1006289, 0.069182389937106917, 0.062893081761006289, 0.0628930817610062
89],
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.0728476
82119205295, 0.072847682119205295, 0.066225165562913912, 0.066225165562
913912, 0.27152317880794702, 0.066225165562913912, 0.06622516556291391
2],
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333
333333333334, 0.073333333333333334, 0.093333333333333338, 0.08000000000
0000002, 0.29999999999999999, 0.066666666666666666, 0.0666666666666666
6],

```

```

#     ...
#     }
gv_dict = get_gv_fea_dict(alpha, feature, df)
# value_count is similar in get_gv_fea_dict
value_count = train_df[feature].value_counts()

# gv_fea: Gene_variation feature, it will contain the feature for e
ach feature value in the data
gv_fea = []
# for every feature values in the given data frame we will check if
it is there in the train data then we will add the feature to gv_fea
# 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_fe
a
for index, row in df.iterrows():
    if row[feature] in dict(value_count).keys():
        gv_fea.append(gv_dict[row[feature]])
    else:
        gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
#     gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
return gv_fea

```

when we 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?

```

In [18]: unique_genes = train_df['Gene'].value_counts()
print('Number of Unique Genes :', unique_genes.shape[0])

```



```
# the top 10 genes that occurred most  
print(unique_genes.head(10))
```

Number of Unique Genes : 228

BRCA1 178

EGFR 101

TP53 99

BRCA2 79

PTEN 74

KIT 68

BRAF 55

ERBB2 47

FLT3 35

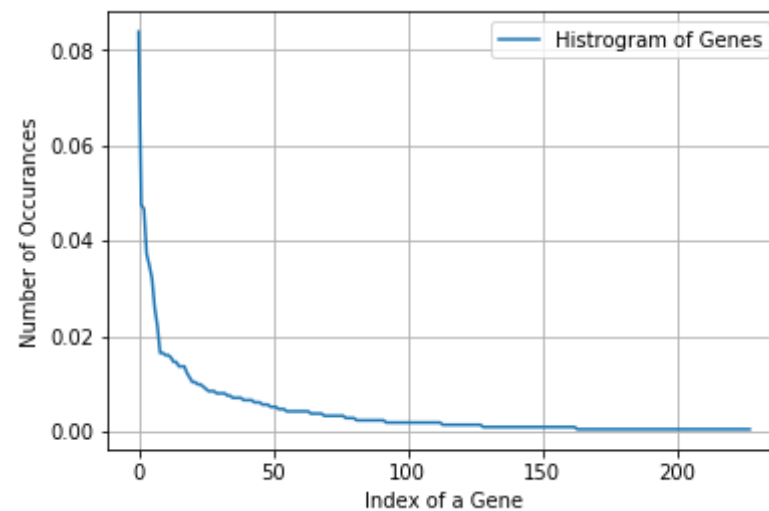
PDGFRA 35

Name: Gene, dtype: int64

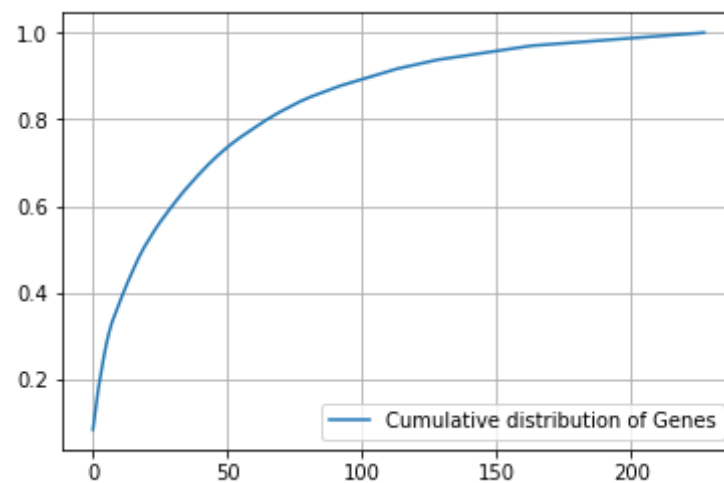
```
In [19]: print("Ans: There are", unique_genes.shape[0], "different categories of  
genes in the train data, and they are distributed as follows",)
```

Ans: There are 228 different categories of genes in the train data, and they are distributed as follows

```
In [20]: s = sum(unique_genes.values);  
h = unique_genes.values/s;  
plt.plot(h, label="Histogram of Genes")  
plt.xlabel('Index of a Gene')  
plt.ylabel('Number of Occurances')  
plt.legend()  
plt.grid()  
plt.show()
```



```
In [21]: c = np.cumsum(h)
plt.plot(c, label='Cumulative distribution of Genes')
plt.grid()
plt.legend()
plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video:
<https://www.appliedaicourse.com/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.

```
In [22]: #response-coding of the Gene feature
# alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", train_df))
# test gene feature
test_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", test_df))
# cross validation gene feature
cv_gene_feature_responseCoding = np.array(get_gv_feature(alpha, "Gene", cv_df))
```

```
In [23]: print("train_gene_feature_responseCoding is converted feature using response coding method. The shape of gene feature:", train_gene_feature_responseCoding.shape)
```

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

```
In [24]: # one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_gene_feature_onehotCoding = gene_vectorizer.fit_transform(train_df['Gene'])
```

```
test_gene_feature_onehotCoding = gene_vectorizer.transform(test_df['Gene'])
cv_gene_feature_onehotCoding = gene_vectorizer.transform(cv_df['Gene'])
```

```
In [25]: train_df['Gene'].head()
```

```
Out[25]: 2686    BRAF
         2692    BRAF
         1739    MSH2
         2512   BRCA1
         1892    MTOR
         Name: Gene, dtype: object
```

```
In [26]: gene_vectorizer.get_feature_names()
```

```
Out[26]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1b',
          'arid5b',
          'asxl2',
          'atm',
          'atrx',
          'aurka',
          'axin1',
          'b2m',
          'bap1',
          'bard1',
          'bcl10',
          'bcl2l11',
          'bcor',
          'braf',
```

```
'brca1',  
'brca2',  
'brd4',  
'brip1',  
'btk',  
'card11',  
'carm1',  
'casp8',  
'cbl',  
'ccnd1',  
'ccnd2',  
'ccnd3',  
'ccne1',  
'cdh1',  
'cdk12',  
'cdk4',  
'cdk6',  
'cdkn1a',  
'cdkn1b',  
'cdkn2a',  
'cdkn2b',  
'cdkn2c',  
'cebpa',  
'chek2',  
'cic',  
'crebbp',  
'ctcf',  
'ctnnb1',  
'ddr2',  
'dicer1',  
'dnmt3a',  
'dnmt3b',  
'dusp4',  
'egfr',  
'eif1ax',  
'elf3',  
'ep300',  
'epas1',  
'erbb2',
```

'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'errfi1',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fam58a',
'fanca',
'fat1',
'fbxw7',
'fgf3',
'fgf4',
'fgfr1',
'fgfr2',
'fgfr3',
'fgfr4',
'flt1',
'flt3',
'foxa1',
'foxl2',
'foxo1',
'foxp1',
'fubp1',
'gata3',
'gnaq',
'gnas',
'hist1h1c',
'hla',
'hnf1a',
'hras',
'idh1',
'idh2',
'igf1r',

'ikbke',
'il7r',
'inpp4b',
'jak1',
'jak2',
'jun',
'kdm5a',
'kdm5c',
'kdm6a',
'kdr',
'keap1',
'kit',
'klf4',
'kmt2a',
'kmt2c',
'kmt2d',
'knstrn',
'kras',
'lats1',
'lats2',
'map2k1',
'map2k2',
'map2k4',
'map3k1',
'mapk1',
'mdm2',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'mycn',
'myd88',
'ncor1',

```
'nf1',  
'nf2',  
'nfe2l2',  
'nfkb1a',  
'nkx2',  
'notch1',  
'notch2',  
'nras',  
'nsd1',  
'ntrk1',  
'ntrk2',  
'ntrk3',  
'nup93',  
'pbrm1',  
'pdgfra',  
'pdgfrb',  
'pik3ca',  
'pik3cb',  
'pik3cd',  
'pik3r1',  
'pik3r2',  
'pik3r3',  
'pim1',  
'pms2',  
'pole',  
'ppp2r1a',  
'ppp6c',  
'prdm1',  
'pten',  
'ptpn11',  
'ptprd',  
'ptprt',  
'rab35',  
'rac1',  
'rad21',  
'rad50',  
'rad51b',  
'rad51c',  
'rad51d',
```


'rad54l',
'raf1',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'runx1',
'rxra',
'rybp',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shoc2',
'smad2',
'smad3',
'smad4',
'smarca4',
'smo',
'sos1',
'sox9',
'spop',
'srsf2',
'stat3',
'stk11',
'tcf3',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
'tp53',

```
'tp53bp1',  
'tsc1',  
'tsc2',  
'u2af1',  
'vegfa',  
'vhl',  
'whsc1',  
'xpo1',  
'yap1']
```

```
In [27]: print("train_gene_feature_onehotCoding is converted feature using one-hot  
encoding method. The shape of gene feature:", train_gene_feature_one  
hotCoding.shape)
```

train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 228)

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.

```
In [28]: alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.  
  
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html  
# -----  
# default parameters  
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15,  
# fit_intercept=True, max_iter=None, tol=None,  
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',  
# eta0=0.0, power_t=0.5,  
# class_weight=None, warm_start=False, average=False, n_iter=None)  
  
# some of methods
```

```

# fit(X, y[, coef_init, intercept_init, ...])    Fit linear model with S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

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

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

```

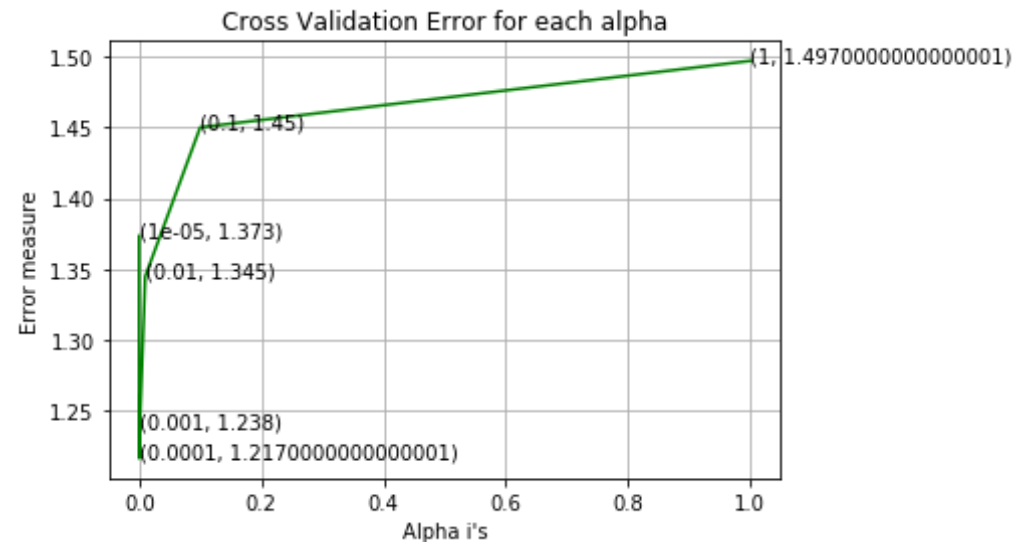
```

sig_clf.fit(train_gene_feature_onehotCoding, y_train)

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
      loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
      ))
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
      dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
      =1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
      oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.37334837881
 For values of alpha = 0.0001 The log loss is: 1.21671946976
 For values of alpha = 0.001 The log loss is: 1.23780768181
 For values of alpha = 0.01 The log loss is: 1.3448127857
 For values of alpha = 0.1 The log loss is: 1.44987690497
 For values of alpha = 1 The log loss is: 1.49675355779



For values of best alpha = 0.0001 The train log loss is: 1.03958597069
 For values of best alpha = 0.0001 The cross validation log loss is: 1.

21671946976

For values of best alpha = 0.0001 The test log loss is: 1.24941773589

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.

```
In [29]: print("Q6. How many data points in Test and CV datasets are covered by  
         the ", unique_genes.shape[0], " genes in train dataset?")  
  
         test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene']  
         ])))]  
         cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))]  
         e[0]  
  
         print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0],  
         ":",(test_coverage/test_df.shape[0])*100)  
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[  
         0],":" ,(cv_coverage/cv_df.shape[0])*100)
```

Q6. How many data points in Test and CV datasets are covered by the 228 genes in train dataset?

Ans

1. In test data 645 out of 665 : 96.99248120300751

2. In cross validation data 507 out of 532 : 95.30075187969925

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?

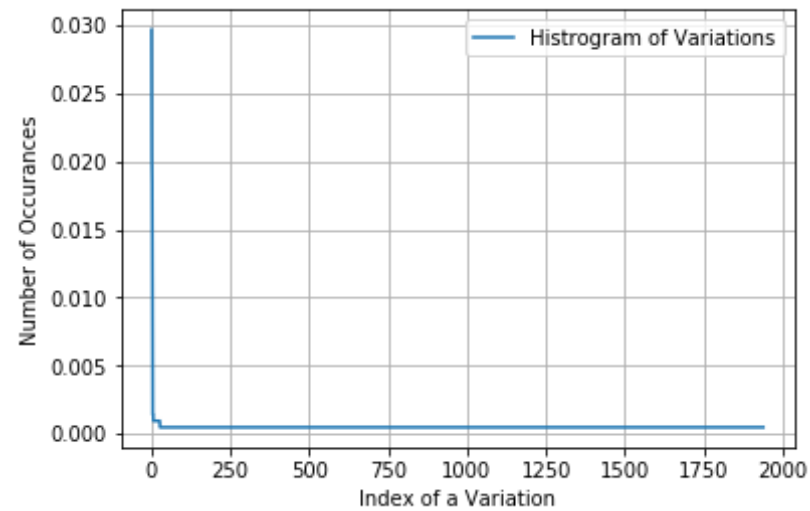
```
In [30]: unique_variations = train_df['Variation'].value_counts()
print('Number of Unique Variations :', unique_variations.shape[0])
# the top 10 variations that occurred most
print(unique_variations.head(10))
```

```
Number of Unique Variations : 1940
Truncating_Mutations      63
Amplification              43
Deletion                  39
Fusions                   18
Q61L                      3
Overexpression            3
R173C                     2
G12C                      2
E17K                      2
E330K                     2
Name: Variation, dtype: int64
```

```
In [31]: print("Ans: There are", unique_variations.shape[0] , "different categories of variations in the train data, and they are distributed as follows", )
```

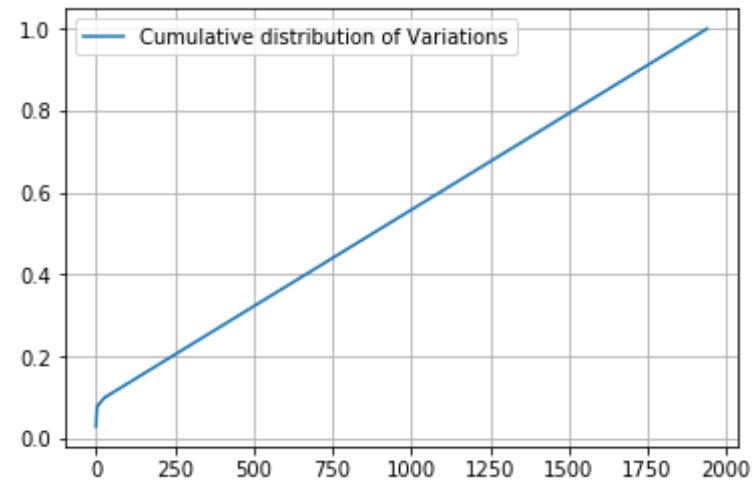
Ans: There are 1940 different categories of variations in the train data, and they are distributed as follows

```
In [32]: s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurances')
plt.legend()
plt.grid()
plt.show()
```



```
In [33]: c = np.cumsum(h)
print(c)
plt.plot(c, label='Cumulative distribution of Variations')
plt.grid()
plt.legend()
plt.show()

[ 0.02966102  0.04990584  0.06826742 ...,  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

```
In [34]: # alpha is used for laplace smoothing
alpha = 1
# train gene feature
train_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", train_df))
# test gene feature
test_variation_feature_responseCoding = np.array(get_gv_feature(alpha,
"Variation", test_df))
# cross validation gene feature
cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "V
ariation", cv_df))
```



```
In [35]: print("train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feature:", train_variation_feature_responseCoding.shape)
```

train_variation_feature_responseCoding is a converted feature using the response coding method. The shape of Variation feature: (2124, 9)

```
In [36]: # one-hot encoding of variation feature.
variation_vectorizer = CountVectorizer(ngram_range=(1, 2))
train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Variation'])
test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variation'])
cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [37]: print("train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature:", train_variation_feature_onehotCoding.shape)
```

train_variation_feature_onehotEncoded is converted feature using the one-hot encoding method. The shape of Variation feature: (2124, 2074)

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

Let's build a model just like the earlier!

```
In [38]: alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
```

```

arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])    Fit linear model with S
tochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.

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

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

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

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

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

```

```

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

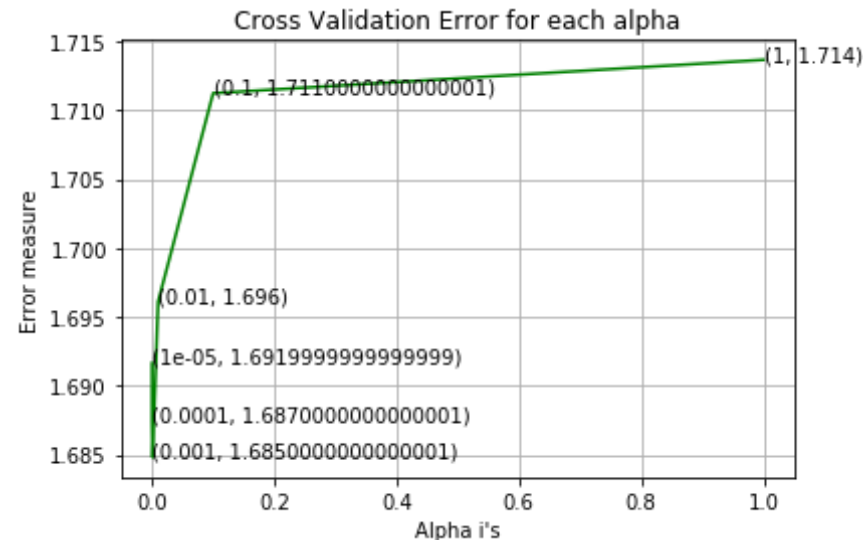
predict_y = sig_clf.predict_proba(train_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

```

For values of alpha = 1e-05 The log loss is: 1.69164208602
For values of alpha = 0.0001 The log loss is: 1.6874326956
For values of alpha = 0.001 The log loss is: 1.68480809229
For values of alpha = 0.01 The log loss is: 1.69609616654
For values of alpha = 0.1 The log loss is: 1.71122133236
For values of alpha = 1 The log loss is: 1.71364535976

```



For values of best alpha = 0.001 The train log loss is: 1.10500011603
 For values of best alpha = 0.001 The cross validation log loss is: 1.68480809229
 For values of best alpha = 0.001 The test log loss is: 1.70196908685

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.

```
In [39]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?")
test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
cv_coverage=cv_df[cv_df['Variation'].isin(list(set(train_df['Variation'])))].shape[0]
print('Ans\n1. In test data',test_coverage, 'out of ',test_df.shape[0], ':',(test_coverage/test_df.shape[0])*100)
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage/cv_df.shape[0])*100)
```

Q12. How many data points are covered by total 1940 genes in test and cross validation data sets?

Ans

1. In test data 71 out of 665 : 10.676691729323307
2. In cross validation data 66 out of 532 : 12.406015037593985

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?

```
In [40]: # cls_text is a data frame
# for every row in data fram consider the 'TEXT'
# split the words by space
# make a dict with those words
# increment its count whenever we see that word

def extract_dictionary_paddle(cls_text):
    dictionary = defaultdict(int)
    for index, row in cls_text.iterrows():
        for word in row['TEXT'].split():
            dictionary[word] +=1
    return dictionary
```

```
In [41]: import math
#https://stackoverflow.com/a/1602964
def get_text_responsecoding(df):
    text_feature_responseCoding = np.zeros((df.shape[0],9))
    for i in range(0,9):
        row_index = 0
        for index, row in df.iterrows():
            sum_prob = 0
            for word in row['TEXT'].split():
```

```

        sum_prob += math.log(((dict_list[i].get(word,0)+10)/(total_dict.get(word,0)+90)))
        text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TEXT'].split()))
        row_index += 1
    return text_feature_responseCoding

```

In [42]:

```

# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3, ngram_range=(1, 2))
train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
# getting all the feature names (words)
train_text_features= text_vectorizer.get_feature_names()

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

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

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

```

Total number of unique words in train data : 766209

In [43]:

```

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

# dict_list[i] is build on i'th class text data

```

```
# total_dict is build on whole training text data
total_dict = extract_dictionary_paddle(train_df)

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

In [44]: *#response coding of text features*

```
train_text_feature_responseCoding = get_text_responsecoding(train_df)
test_text_feature_responseCoding = get_text_responsecoding(test_df)
cv_text_feature_responseCoding = get_text_responsecoding(cv_df)
```

In [45]: *# https://stackoverflow.com/a/16202486*
we convert each row values such that they sum to 1

```
train_text_feature_responseCoding = (train_text_feature_responseCoding.T/
train_text_feature_responseCoding.sum(axis=1)).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/
test_text_feature_responseCoding.sum(axis=1)).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text
ext_feature_responseCoding.sum(axis=1)).T
```

In [46]: *# don't forget to normalize every feature*

```
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
In [47]: #https://stackoverflow.com/a/2258273/4084039
sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x:
x[1] , reverse=True))
sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

```
In [48]: # Number of words for a given frequency.
print(Counter(sorted_text_occur))
```

```
Counter({3: 142295, 4: 100545, 5: 68525, 6: 60785, 8: 47979, 7: 37195,
9: 30807, 10: 30601, 11: 20820, 12: 20146, 14: 17272, 13: 15538, 16: 11
848, 15: 10851, 18: 8121, 17: 8090, 19: 6743, 20: 6678, 22: 5112, 21: 5
035, 28: 4744, 24: 4433, 30: 4410, 27: 4051, 23: 3620, 45: 3266, 25: 32
07, 26: 3069, 29: 2752, 32: 2389, 31: 2377, 54: 2245, 43: 2055, 33: 205
2, 34: 1894, 36: 1847, 35: 1786, 37: 1505, 55: 1460, 38: 1455, 40: 142
1, 39: 1369, 69: 1342, 42: 1270, 44: 1265, 46: 1189, 41: 1146, 48: 114
3, 47: 1057, 56: 1028, 49: 969, 52: 841, 60: 839, 50: 831, 51: 810, 57:
799, 53: 775, 58: 736, 59: 697, 62: 620, 61: 597, 63: 573, 70: 557, 66:
555, 64: 554, 72: 546, 65: 532, 71: 517, 86: 509, 67: 486, 68: 474, 75:
442, 74: 432, 73: 425, 76: 412, 81: 407, 77: 403, 80: 375, 78: 371, 84:
357, 90: 356, 79: 346, 82: 329, 87: 306, 85: 306, 83: 300, 88: 299, 92:
295, 91: 279, 89: 269, 108: 262, 94: 250, 93: 247, 95: 237, 97: 232, 9
6: 231, 98: 221, 110: 219, 101: 218, 99: 213, 104: 210, 102: 207, 100:
199, 111: 196, 105: 193, 103: 193, 109: 189, 112: 185, 107: 185, 106: 1
75, 116: 168, 117: 163, 129: 159, 118: 157, 114: 157, 120: 154, 115: 15
3, 119: 149, 126: 143, 135: 139, 128: 139, 125: 137, 130: 136, 122: 13
5, 113: 133, 133: 129, 132: 129, 124: 128, 121: 127, 138: 126, 123: 12
6, 140: 125, 127: 119, 143: 117, 136: 117, 162: 116, 131: 116, 142: 11
4, 139: 113, 144: 112, 141: 111, 145: 109, 137: 106, 147: 102, 134: 10
2, 172: 101, 153: 100, 148: 100, 146: 100, 149: 99, 154: 98, 168: 94, 1
55: 94, 150: 94, 165: 91, 158: 88, 151: 88, 159: 85, 180: 84, 175: 83,
163: 80, 152: 79, 193: 78, 174: 77, 167: 77, 164: 77, 161: 76, 160: 76,
216: 74, 156: 72, 173: 71, 183: 70, 171: 70, 178: 68, 157: 68, 192: 67,
187: 67, 169: 67, 170: 66, 189: 65, 176: 64, 179: 62, 199: 61, 188: 59,
```


181: 59, 210: 58, 190: 58, 166: 58, 196: 57, 186: 57, 197: 56, 198: 55,
191: 54, 212: 53, 209: 53, 208: 53, 207: 53, 213: 52, 201: 52, 185: 52,
182: 52, 200: 51, 184: 51, 195: 50, 224: 49, 215: 49, 220: 48, 194: 48,
177: 48, 218: 47, 202: 46, 270: 45, 221: 44, 211: 44, 227: 43, 225: 43,
217: 43, 259: 42, 243: 42, 260: 41, 237: 41, 222: 41, 229: 40, 223: 39,
219: 39, 272: 38, 233: 38, 228: 38, 250: 37, 230: 37, 276: 36, 263: 36,
253: 36, 251: 36, 231: 36, 206: 36, 205: 36, 252: 35, 232: 35, 273: 34,
262: 34, 245: 34, 244: 34, 234: 34, 265: 33, 248: 33, 246: 33, 235: 33,
226: 33, 204: 33, 289: 32, 258: 32, 255: 32, 241: 32, 277: 31, 268: 31,
267: 31, 254: 31, 239: 31, 214: 31, 288: 30, 323: 29, 319: 29, 307: 29,
305: 29, 304: 29, 286: 29, 281: 29, 274: 29, 269: 29, 261: 29, 249: 29,
242: 29, 203: 29, 293: 28, 280: 28, 238: 28, 236: 28, 329: 27, 322: 27,
301: 27, 285: 27, 282: 27, 247: 27, 240: 27, 284: 26, 344: 25, 332: 25,
296: 25, 290: 25, 279: 25, 257: 25, 396: 24, 333: 24, 292: 24, 275: 24,
271: 24, 256: 24, 360: 23, 328: 23, 283: 23, 264: 23, 375: 22, 339: 22,
327: 22, 324: 22, 316: 22, 313: 22, 311: 22, 310: 22, 291: 22, 287: 22,
266: 22, 347: 21, 321: 21, 320: 21, 306: 21, 299: 21, 408: 20, 378: 20,
351: 20, 346: 20, 330: 20, 315: 20, 302: 20, 300: 20, 295: 20, 278: 20,
432: 19, 388: 19, 377: 19, 364: 19, 363: 19, 355: 19, 352: 19, 342: 19,
326: 19, 314: 19, 309: 19, 298: 19, 297: 19, 439: 18, 426: 18, 421: 18,
373: 18, 367: 18, 336: 18, 335: 18, 318: 18, 303: 18, 447: 17, 385: 17,
379: 17, 371: 17, 338: 17, 504: 16, 463: 16, 405: 16, 374: 16, 369: 16,
359: 16, 349: 16, 343: 16, 325: 16, 518: 15, 511: 15, 451: 15, 443: 15,
425: 15, 416: 15, 401: 15, 394: 15, 391: 15, 384: 15, 370: 15, 358: 15,
357: 15, 348: 15, 341: 15, 340: 15, 337: 15, 334: 15, 619: 14, 490: 14,
471: 14, 456: 14, 450: 14, 435: 14, 390: 14, 372: 14, 368: 14, 366: 14,
365: 14, 361: 14, 356: 14, 353: 14, 350: 14, 331: 14, 312: 14, 558: 13,
498: 13, 480: 13, 475: 13, 442: 13, 431: 13, 430: 13, 427: 13, 422: 13,
419: 13, 413: 13, 409: 13, 404: 13, 400: 13, 389: 13, 386: 13, 354: 13,
308: 13, 725: 12, 520: 12, 458: 12, 449: 12, 437: 12, 418: 12, 411: 12,
406: 12, 403: 12, 387: 12, 376: 12, 345: 12, 317: 12, 679: 11, 571: 11,
569: 11, 561: 11, 554: 11, 540: 11, 536: 11, 534: 11, 528: 11, 515: 11,
493: 11, 486: 11, 481: 11, 470: 11, 467: 11, 459: 11, 445: 11, 444: 11,
441: 11, 440: 11, 429: 11, 399: 11, 393: 11, 381: 11, 294: 11, 820: 10,
723: 10, 712: 10, 646: 10, 586: 10, 581: 10, 578: 10, 559: 10, 549: 10,
537: 10, 529: 10, 503: 10, 484: 10, 483: 10, 474: 10, 465: 10, 455: 10,
420: 10, 417: 10, 415: 10, 407: 10, 398: 10, 397: 10, 395: 10, 392: 10,
1004: 9, 701: 9, 639: 9, 638: 9, 567: 9, 530: 9, 522: 9, 521: 9, 516:
9, 500: 9, 497: 9, 492: 9, 487: 9, 477: 9, 476: 9, 473: 9, 468: 9, 453:

9, 452: 9, 424: 9, 414: 9, 410: 9, 382: 9, 362: 9, 1238: 8, 1197: 8, 79
9: 8, 766: 8, 762: 8, 751: 8, 680: 8, 672: 8, 659: 8, 657: 8, 633: 8, 6
11: 8, 603: 8, 595: 8, 594: 8, 570: 8, 552: 8, 550: 8, 543: 8, 541: 8,
538: 8, 533: 8, 524: 8, 512: 8, 510: 8, 507: 8, 499: 8, 482: 8, 466: 8,
464: 8, 461: 8, 448: 8, 446: 8, 436: 8, 412: 8, 947: 7, 945: 7, 785: 7,
752: 7, 745: 7, 704: 7, 699: 7, 698: 7, 693: 7, 691: 7, 689: 7, 683: 7,
678: 7, 675: 7, 668: 7, 664: 7, 658: 7, 656: 7, 654: 7, 647: 7, 645: 7,
643: 7, 641: 7, 632: 7, 631: 7, 626: 7, 624: 7, 613: 7, 598: 7, 596: 7,
583: 7, 572: 7, 565: 7, 563: 7, 557: 7, 551: 7, 544: 7, 531: 7, 527: 7,
513: 7, 495: 7, 491: 7, 489: 7, 488: 7, 485: 7, 479: 7, 478: 7, 472: 7,
460: 7, 454: 7, 438: 7, 433: 7, 383: 7, 1902: 6, 1179: 6, 1012: 6, 990:
6, 983: 6, 970: 6, 966: 6, 890: 6, 855: 6, 837: 6, 825: 6, 806: 6, 805:
6, 804: 6, 800: 6, 777: 6, 744: 6, 743: 6, 740: 6, 738: 6, 731: 6, 730:
6, 721: 6, 709: 6, 682: 6, 677: 6, 673: 6, 653: 6, 651: 6, 650: 6, 648:
6, 637: 6, 625: 6, 614: 6, 605: 6, 601: 6, 599: 6, 590: 6, 588: 6, 585:
6, 584: 6, 566: 6, 562: 6, 560: 6, 555: 6, 546: 6, 526: 6, 523: 6, 514:
6, 496: 6, 494: 6, 469: 6, 462: 6, 434: 6, 428: 6, 380: 6, 2028: 5, 168
9: 5, 1603: 5, 1567: 5, 1366: 5, 1359: 5, 1231: 5, 1203: 5, 1180: 5, 11
48: 5, 1108: 5, 1096: 5, 1061: 5, 1032: 5, 1001: 5, 981: 5, 978: 5, 95
7: 5, 943: 5, 928: 5, 923: 5, 902: 5, 893: 5, 892: 5, 871: 5, 858: 5, 8
48: 5, 844: 5, 841: 5, 834: 5, 832: 5, 817: 5, 814: 5, 798: 5, 793: 5,
792: 5, 790: 5, 783: 5, 782: 5, 779: 5, 776: 5, 772: 5, 763: 5, 759: 5,
748: 5, 739: 5, 729: 5, 728: 5, 727: 5, 716: 5, 702: 5, 697: 5, 696: 5,
695: 5, 688: 5, 687: 5, 685: 5, 670: 5, 667: 5, 665: 5, 640: 5, 630: 5,
627: 5, 622: 5, 621: 5, 612: 5, 610: 5, 609: 5, 608: 5, 607: 5, 606: 5,
597: 5, 582: 5, 577: 5, 575: 5, 574: 5, 573: 5, 568: 5, 556: 5, 553: 5,
545: 5, 539: 5, 535: 5, 532: 5, 519: 5, 517: 5, 509: 5, 506: 5, 505: 5,
402: 5, 3998: 4, 2081: 4, 1750: 4, 1718: 4, 1584: 4, 1489: 4, 1472: 4,
1455: 4, 1447: 4, 1381: 4, 1360: 4, 1281: 4, 1271: 4, 1265: 4, 1253: 4,
1251: 4, 1243: 4, 1221: 4, 1216: 4, 1184: 4, 1182: 4, 1171: 4, 1170: 4,
1121: 4, 1104: 4, 1082: 4, 1080: 4, 1074: 4, 1066: 4, 1055: 4, 1042: 4,
1038: 4, 1037: 4, 1036: 4, 1035: 4, 1030: 4, 1019: 4, 1014: 4, 1011: 4,
996: 4, 989: 4, 972: 4, 968: 4, 956: 4, 951: 4, 933: 4, 931: 4, 930: 4,
929: 4, 921: 4, 918: 4, 917: 4, 910: 4, 908: 4, 898: 4, 894: 4, 891: 4,
889: 4, 880: 4, 870: 4, 862: 4, 859: 4, 854: 4, 845: 4, 843: 4, 831: 4,
813: 4, 811: 4, 809: 4, 807: 4, 802: 4, 801: 4, 794: 4, 789: 4, 788: 4,
786: 4, 784: 4, 774: 4, 769: 4, 749: 4, 746: 4, 742: 4, 741: 4, 734: 4,
733: 4, 720: 4, 717: 4, 713: 4, 711: 4, 710: 4, 705: 4, 690: 4, 686: 4,
663: 4, 662: 4, 661: 4, 649: 4, 634: 4, 628: 4, 623: 4, 617: 4, 616: 4,

604: 4, 600: 4, 589: 4, 587: 4, 580: 4, 579: 4, 576: 4, 564: 4, 548: 4,
547: 4, 525: 4, 457: 4, 423: 4, 3678: 3, 3583: 3, 3255: 3, 3066: 3, 238
6: 3, 2369: 3, 2209: 3, 2205: 3, 2152: 3, 2120: 3, 2096: 3, 2088: 3, 20
13: 3, 1991: 3, 1971: 3, 1961: 3, 1953: 3, 1933: 3, 1917: 3, 1883: 3, 1
859: 3, 1842: 3, 1836: 3, 1816: 3, 1768: 3, 1767: 3, 1740: 3, 1729: 3,
1709: 3, 1686: 3, 1675: 3, 1659: 3, 1621: 3, 1600: 3, 1585: 3, 1536: 3,
1527: 3, 1525: 3, 1521: 3, 1491: 3, 1456: 3, 1450: 3, 1443: 3, 1436: 3,
1435: 3, 1420: 3, 1417: 3, 1404: 3, 1392: 3, 1387: 3, 1382: 3, 1372: 3,
1367: 3, 1362: 3, 1361: 3, 1352: 3, 1335: 3, 1333: 3, 1321: 3, 1318: 3,
1305: 3, 1304: 3, 1294: 3, 1293: 3, 1282: 3, 1280: 3, 1255: 3, 1235: 3,
1224: 3, 1219: 3, 1214: 3, 1207: 3, 1198: 3, 1192: 3, 1187: 3, 1185: 3,
1175: 3, 1169: 3, 1165: 3, 1163: 3, 1161: 3, 1158: 3, 1157: 3, 1143: 3,
1138: 3, 1128: 3, 1127: 3, 1119: 3, 1116: 3, 1110: 3, 1102: 3, 1101: 3,
1097: 3, 1094: 3, 1086: 3, 1083: 3, 1070: 3, 1067: 3, 1062: 3, 1060: 3,
1059: 3, 1056: 3, 1050: 3, 1046: 3, 1044: 3, 1039: 3, 1027: 3, 1020: 3,
1015: 3, 1009: 3, 1005: 3, 1000: 3, 998: 3, 994: 3, 982: 3, 980: 3, 97
7: 3, 974: 3, 971: 3, 969: 3, 967: 3, 965: 3, 962: 3, 961: 3, 960: 3, 9
58: 3, 955: 3, 950: 3, 938: 3, 922: 3, 920: 3, 913: 3, 912: 3, 907: 3,
905: 3, 904: 3, 895: 3, 887: 3, 885: 3, 882: 3, 878: 3, 877: 3, 875: 3,
874: 3, 873: 3, 860: 3, 856: 3, 851: 3, 850: 3, 849: 3, 847: 3, 842: 3,
839: 3, 830: 3, 827: 3, 818: 3, 816: 3, 815: 3, 810: 3, 797: 3, 796: 3,
780: 3, 778: 3, 773: 3, 770: 3, 767: 3, 764: 3, 761: 3, 760: 3, 757: 3,
755: 3, 753: 3, 737: 3, 736: 3, 724: 3, 718: 3, 715: 3, 714: 3, 708: 3,
707: 3, 706: 3, 703: 3, 676: 3, 674: 3, 671: 3, 669: 3, 644: 3, 635: 3,
620: 3, 615: 3, 593: 3, 542: 3, 508: 3, 502: 3, 501: 3, 18743: 2, 7834:
2, 7800: 2, 7268: 2, 6992: 2, 6667: 2, 6578: 2, 6561: 2, 6302: 2, 5926:
2, 5923: 2, 5552: 2, 4900: 2, 4687: 2, 4595: 2, 4418: 2, 4379: 2, 4206:
2, 4144: 2, 4068: 2, 4041: 2, 4035: 2, 3969: 2, 3868: 2, 3702: 2, 3659:
2, 3658: 2, 3645: 2, 3523: 2, 3469: 2, 3466: 2, 3464: 2, 3397: 2, 3366:
2, 3358: 2, 3333: 2, 3308: 2, 3250: 2, 3239: 2, 3220: 2, 3187: 2, 3178:
2, 3118: 2, 3034: 2, 3032: 2, 2975: 2, 2931: 2, 2923: 2, 2919: 2, 2917:
2, 2900: 2, 2856: 2, 2850: 2, 2837: 2, 2820: 2, 2789: 2, 2755: 2, 2743:
2, 2730: 2, 2726: 2, 2720: 2, 2708: 2, 2697: 2, 2682: 2, 2677: 2, 2661:
2, 2648: 2, 2611: 2, 2602: 2, 2571: 2, 2562: 2, 2559: 2, 2543: 2, 2534:
2, 2529: 2, 2506: 2, 2499: 2, 2490: 2, 2477: 2, 2471: 2, 2468: 2, 2436:
2, 2410: 2, 2399: 2, 2395: 2, 2389: 2, 2379: 2, 2336: 2, 2334: 2, 2333:
2, 2330: 2, 2329: 2, 2328: 2, 2326: 2, 2324: 2, 2320: 2, 2318: 2, 2312:
2, 2295: 2, 2248: 2, 2247: 2, 2245: 2, 2241: 2, 2222: 2, 2220: 2, 2200:
2, 2199: 2, 2195: 2, 2191: 2, 2184: 2, 2177: 2, 2172: 2, 2168: 2, 2114:

2, 2107: 2, 2104: 2, 2093: 2, 2082: 2, 2075: 2, 2071: 2, 2057: 2, 2026:
2, 2017: 2, 2007: 2, 2002: 2, 1998: 2, 1997: 2, 1992: 2, 1988: 2, 1975:
2, 1954: 2, 1951: 2, 1948: 2, 1943: 2, 1940: 2, 1903: 2, 1901: 2, 1895:
2, 1892: 2, 1881: 2, 1872: 2, 1851: 2, 1850: 2, 1834: 2, 1833: 2, 1831:
2, 1830: 2, 1824: 2, 1822: 2, 1817: 2, 1812: 2, 1810: 2, 1804: 2, 1800:
2, 1797: 2, 1788: 2, 1786: 2, 1778: 2, 1775: 2, 1772: 2, 1764: 2, 1762:
2, 1754: 2, 1747: 2, 1745: 2, 1738: 2, 1730: 2, 1721: 2, 1714: 2, 1712:
2, 1696: 2, 1692: 2, 1682: 2, 1681: 2, 1680: 2, 1678: 2, 1674: 2, 1666:
2, 1664: 2, 1656: 2, 1644: 2, 1643: 2, 1641: 2, 1632: 2, 1629: 2, 1627:
2, 1626: 2, 1622: 2, 1615: 2, 1614: 2, 1610: 2, 1608: 2, 1604: 2, 1599:
2, 1594: 2, 1586: 2, 1578: 2, 1575: 2, 1559: 2, 1558: 2, 1557: 2, 1554:
2, 1549: 2, 1547: 2, 1545: 2, 1542: 2, 1534: 2, 1528: 2, 1522: 2, 1519:
2, 1508: 2, 1505: 2, 1504: 2, 1497: 2, 1488: 2, 1485: 2, 1480: 2, 1477:
2, 1476: 2, 1468: 2, 1466: 2, 1460: 2, 1453: 2, 1449: 2, 1448: 2, 1431:
2, 1401: 2, 1399: 2, 1391: 2, 1388: 2, 1378: 2, 1377: 2, 1374: 2, 1373:
2, 1371: 2, 1358: 2, 1354: 2, 1342: 2, 1340: 2, 1336: 2, 1327: 2, 1325:
2, 1323: 2, 1319: 2, 1317: 2, 1313: 2, 1312: 2, 1309: 2, 1306: 2, 1303:
2, 1301: 2, 1299: 2, 1296: 2, 1291: 2, 1289: 2, 1288: 2, 1287: 2, 1286:
2, 1279: 2, 1277: 2, 1264: 2, 1262: 2, 1261: 2, 1260: 2, 1256: 2, 1252:
2, 1250: 2, 1247: 2, 1246: 2, 1242: 2, 1241: 2, 1237: 2, 1233: 2, 1230:
2, 1228: 2, 1226: 2, 1223: 2, 1222: 2, 1218: 2, 1206: 2, 1201: 2, 1200:
2, 1194: 2, 1193: 2, 1191: 2, 1188: 2, 1181: 2, 1173: 2, 1164: 2, 1159:
2, 1155: 2, 1147: 2, 1146: 2, 1145: 2, 1142: 2, 1136: 2, 1135: 2, 1131:
2, 1122: 2, 1115: 2, 1114: 2, 1112: 2, 1111: 2, 1107: 2, 1095: 2, 1091:
2, 1089: 2, 1088: 2, 1087: 2, 1084: 2, 1072: 2, 1071: 2, 1058: 2, 1057:
2, 1054: 2, 1052: 2, 1051: 2, 1049: 2, 1048: 2, 1047: 2, 1043: 2, 1040:
2, 1034: 2, 1033: 2, 1022: 2, 1017: 2, 1016: 2, 1013: 2, 1008: 2, 1007:
2, 997: 2, 995: 2, 993: 2, 992: 2, 986: 2, 985: 2, 984: 2, 979: 2, 976:
2, 964: 2, 963: 2, 954: 2, 953: 2, 952: 2, 949: 2, 948: 2, 942: 2, 941:
2, 936: 2, 932: 2, 925: 2, 919: 2, 914: 2, 911: 2, 903: 2, 901: 2, 899:
2, 888: 2, 886: 2, 883: 2, 881: 2, 879: 2, 876: 2, 872: 2, 869: 2, 867:
2, 866: 2, 865: 2, 864: 2, 853: 2, 852: 2, 840: 2, 838: 2, 836: 2, 833:
2, 828: 2, 824: 2, 822: 2, 812: 2, 803: 2, 791: 2, 787: 2, 781: 2, 754:
2, 747: 2, 735: 2, 732: 2, 726: 2, 719: 2, 700: 2, 694: 2, 692: 2, 681:
2, 666: 2, 660: 2, 655: 2, 652: 2, 636: 2, 592: 2, 591: 2, 152599: 1, 1
19436: 1, 80827: 1, 69004: 1, 67324: 1, 65137: 1, 65064: 1, 64435: 1, 6
2832: 1, 62020: 1, 56843: 1, 55001: 1, 49290: 1, 48242: 1, 47591: 1, 46
615: 1, 44539: 1, 42876: 1, 42761: 1, 42510: 1, 41976: 1, 41303: 1, 410
71: 1, 40824: 1, 39957: 1, 38625: 1, 38197: 1, 37814: 1, 37510: 1, 3705

1: 1, 36740: 1, 36440: 1, 34597: 1, 33696: 1, 33108: 1, 33039: 1, 3216
2: 1, 31510: 1, 29407: 1, 28404: 1, 27401: 1, 26750: 1, 26277: 1, 2615
3: 1, 26056: 1, 25606: 1, 24832: 1, 24727: 1, 24708: 1, 24476: 1, 2444
8: 1, 24270: 1, 23901: 1, 23708: 1, 22838: 1, 22555: 1, 22317: 1, 2204
2: 1, 21948: 1, 21859: 1, 21330: 1, 21059: 1, 21019: 1, 20636: 1, 2033
3: 1, 19860: 1, 19845: 1, 19544: 1, 19481: 1, 19388: 1, 19333: 1, 1929
6: 1, 18757: 1, 18520: 1, 18422: 1, 18364: 1, 18295: 1, 18241: 1, 1823
6: 1, 18230: 1, 18105: 1, 17834: 1, 17751: 1, 17749: 1, 17671: 1, 1757
8: 1, 17374: 1, 17333: 1, 17276: 1, 17079: 1, 16968: 1, 16923: 1, 1691
6: 1, 16899: 1, 16776: 1, 16769: 1, 16747: 1, 16681: 1, 16083: 1, 1602
5: 1, 16013: 1, 15972: 1, 15847: 1, 15724: 1, 15559: 1, 15448: 1, 1543
4: 1, 15272: 1, 15237: 1, 15129: 1, 14819: 1, 14738: 1, 14721: 1, 1463
0: 1, 14618: 1, 14548: 1, 14468: 1, 14460: 1, 14458: 1, 14288: 1, 1428
2: 1, 14053: 1, 13970: 1, 13921: 1, 13776: 1, 13694: 1, 13620: 1, 1350
4: 1, 13488: 1, 13414: 1, 13362: 1, 13204: 1, 13167: 1, 13152: 1, 1312
2: 1, 13087: 1, 13068: 1, 13061: 1, 13051: 1, 12991: 1, 12907: 1, 1277
3: 1, 12715: 1, 12714: 1, 12706: 1, 12705: 1, 12687: 1, 12625: 1, 1262
3: 1, 12614: 1, 12602: 1, 12547: 1, 12528: 1, 12474: 1, 12436: 1, 1242
6: 1, 12406: 1, 12365: 1, 12232: 1, 12207: 1, 12192: 1, 12130: 1, 1203
9: 1, 12026: 1, 12020: 1, 12017: 1, 11908: 1, 11907: 1, 11904: 1, 1189
7: 1, 11847: 1, 11810: 1, 11652: 1, 11624: 1, 11533: 1, 11492: 1, 1145
8: 1, 11447: 1, 11440: 1, 11375: 1, 11302: 1, 11250: 1, 11177: 1, 1101
9: 1, 10945: 1, 10893: 1, 10867: 1, 10784: 1, 10754: 1, 10679: 1, 1061
5: 1, 10567: 1, 10563: 1, 10549: 1, 10402: 1, 10399: 1, 10359: 1, 1028
2: 1, 10262: 1, 10236: 1, 10195: 1, 10181: 1, 10113: 1, 10082: 1, 1002
4: 1, 10021: 1, 10008: 1, 9945: 1, 9896: 1, 9889: 1, 9830: 1, 9802: 1,
9659: 1, 9646: 1, 9633: 1, 9631: 1, 9531: 1, 9506: 1, 9472: 1, 9425: 1,
9418: 1, 9413: 1, 9409: 1, 9353: 1, 9340: 1, 9331: 1, 9254: 1, 9220: 1,
9187: 1, 9175: 1, 9172: 1, 9167: 1, 9143: 1, 9127: 1, 9097: 1, 9092: 1,
9055: 1, 9052: 1, 9030: 1, 9017: 1, 8988: 1, 8949: 1, 8937: 1, 8928: 1,
8916: 1, 8896: 1, 8819: 1, 8788: 1, 8780: 1, 8774: 1, 8688: 1, 8584: 1,
8503: 1, 8501: 1, 8494: 1, 8424: 1, 8362: 1, 8359: 1, 8341: 1, 8337: 1,
8327: 1, 8270: 1, 8263: 1, 8239: 1, 8223: 1, 8219: 1, 8189: 1, 8179: 1,
8167: 1, 8130: 1, 8129: 1, 8108: 1, 8105: 1, 8093: 1, 8076: 1, 8039: 1,
8031: 1, 7987: 1, 7984: 1, 7982: 1, 7936: 1, 7927: 1, 7912: 1, 7874: 1,
7859: 1, 7849: 1, 7815: 1, 7792: 1, 7768: 1, 7761: 1, 7729: 1, 7716: 1,
7701: 1, 7691: 1, 7642: 1, 7635: 1, 7602: 1, 7588: 1, 7570: 1, 7559: 1,
7529: 1, 7496: 1, 7465: 1, 7458: 1, 7426: 1, 7370: 1, 7359: 1, 7337: 1,
7312: 1, 7310: 1, 7267: 1, 7261: 1, 7211: 1, 7207: 1, 7203: 1, 7201: 1,

7189: 1, 7152: 1, 7123: 1, 7120: 1, 7118: 1, 7114: 1, 7105: 1, 7097: 1,
7040: 1, 7005: 1, 7000: 1, 6951: 1, 6924: 1, 6923: 1, 6921: 1, 6918: 1,
6912: 1, 6879: 1, 6876: 1, 6864: 1, 6834: 1, 6830: 1, 6828: 1, 6812: 1,
6811: 1, 6808: 1, 6779: 1, 6775: 1, 6769: 1, 6763: 1, 6738: 1, 6721: 1,
6719: 1, 6699: 1, 6690: 1, 6671: 1, 6665: 1, 6664: 1, 6658: 1, 6646: 1,
6641: 1, 6631: 1, 6629: 1, 6543: 1, 6542: 1, 6525: 1, 6507: 1, 6501: 1,
6469: 1, 6456: 1, 6450: 1, 6448: 1, 6412: 1, 6410: 1, 6406: 1, 6382: 1,
6371: 1, 6326: 1, 6324: 1, 6321: 1, 6307: 1, 6284: 1, 6280: 1, 6246: 1,
6236: 1, 6209: 1, 6189: 1, 6136: 1, 6124: 1, 6121: 1, 6078: 1, 6055: 1,
6050: 1, 6034: 1, 6019: 1, 6017: 1, 6010: 1, 6001: 1, 5989: 1, 5985: 1,
5982: 1, 5979: 1, 5949: 1, 5942: 1, 5938: 1, 5920: 1, 5914: 1, 5897: 1,
5889: 1, 5888: 1, 5884: 1, 5877: 1, 5865: 1, 5825: 1, 5821: 1, 5799: 1,
5798: 1, 5768: 1, 5760: 1, 5747: 1, 5733: 1, 5732: 1, 5725: 1, 5718: 1,
5712: 1, 5706: 1, 5703: 1, 5698: 1, 5693: 1, 5688: 1, 5651: 1, 5641: 1,
5625: 1, 5614: 1, 5586: 1, 5578: 1, 5551: 1, 5528: 1, 5524: 1, 5504: 1,
5496: 1, 5492: 1, 5473: 1, 5443: 1, 5429: 1, 5425: 1, 5408: 1, 5405: 1,
5403: 1, 5380: 1, 5357: 1, 5354: 1, 5347: 1, 5346: 1, 5336: 1, 5334: 1,
5308: 1, 5283: 1, 5273: 1, 5258: 1, 5256: 1, 5244: 1, 5242: 1, 5207: 1,
5168: 1, 5167: 1, 5156: 1, 5154: 1, 5153: 1, 5144: 1, 5135: 1, 5134: 1,
5120: 1, 5119: 1, 5106: 1, 5096: 1, 5092: 1, 5078: 1, 5075: 1, 5058: 1,
5052: 1, 5042: 1, 5036: 1, 5026: 1, 5005: 1, 5002: 1, 4998: 1, 4986: 1,
4985: 1, 4981: 1, 4978: 1, 4975: 1, 4972: 1, 4954: 1, 4935: 1, 4932: 1,
4928: 1, 4922: 1, 4919: 1, 4917: 1, 4896: 1, 4895: 1, 4887: 1, 4885: 1,
4881: 1, 4880: 1, 4859: 1, 4858: 1, 4856: 1, 4855: 1, 4846: 1, 4826: 1,
4820: 1, 4818: 1, 4800: 1, 4789: 1, 4760: 1, 4754: 1, 4746: 1, 4745: 1,
4739: 1, 4738: 1, 4728: 1, 4720: 1, 4708: 1, 4694: 1, 4690: 1, 4629: 1,
4621: 1, 4607: 1, 4605: 1, 4590: 1, 4589: 1, 4587: 1, 4584: 1, 4582: 1,
4581: 1, 4578: 1, 4573: 1, 4569: 1, 4563: 1, 4562: 1, 4558: 1, 4541: 1,
4537: 1, 4532: 1, 4529: 1, 4524: 1, 4522: 1, 4502: 1, 4493: 1, 4480: 1,
4471: 1, 4449: 1, 4444: 1, 4439: 1, 4432: 1, 4420: 1, 4415: 1, 4402: 1,
4393: 1, 4390: 1, 4364: 1, 4363: 1, 4362: 1, 4361: 1, 4359: 1, 4346: 1,
4343: 1, 4330: 1, 4326: 1, 4317: 1, 4316: 1, 4312: 1, 4307: 1, 4300: 1,
4276: 1, 4275: 1, 4273: 1, 4268: 1, 4266: 1, 4255: 1, 4242: 1, 4227: 1,
4226: 1, 4217: 1, 4214: 1, 4210: 1, 4201: 1, 4194: 1, 4189: 1, 4188: 1,
4187: 1, 4186: 1, 4185: 1, 4184: 1, 4183: 1, 4180: 1, 4172: 1, 4166: 1,
4165: 1, 4159: 1, 4155: 1, 4153: 1, 4145: 1, 4143: 1, 4139: 1, 4131: 1,
4129: 1, 4126: 1, 4123: 1, 4109: 1, 4108: 1, 4094: 1, 4090: 1, 4082: 1,
4073: 1, 4062: 1, 4054: 1, 4048: 1, 4031: 1, 4029: 1, 4023: 1, 4022: 1,
4020: 1, 4019: 1, 4001: 1, 3999: 1, 3985: 1, 3976: 1, 3964: 1, 3961: 1,

3959: 1, 3953: 1, 3947: 1, 3936: 1, 3931: 1, 3929: 1, 3927: 1, 3924: 1,
3923: 1, 3912: 1, 3897: 1, 3891: 1, 3854: 1, 3851: 1, 3848: 1, 3842: 1,
3841: 1, 3840: 1, 3830: 1, 3829: 1, 3824: 1, 3823: 1, 3822: 1, 3815: 1,
3813: 1, 3811: 1, 3810: 1, 3803: 1, 3800: 1, 3793: 1, 3791: 1, 3790: 1,
3786: 1, 3774: 1, 3772: 1, 3771: 1, 3769: 1, 3766: 1, 3762: 1, 3747: 1,
3745: 1, 3743: 1, 3739: 1, 3736: 1, 3718: 1, 3711: 1, 3701: 1, 3700: 1,
3692: 1, 3687: 1, 3682: 1, 3672: 1, 3665: 1, 3660: 1, 3653: 1, 3648: 1,
3628: 1, 3622: 1, 3607: 1, 3603: 1, 3602: 1, 3601: 1, 3598: 1, 3595: 1,
3594: 1, 3581: 1, 3571: 1, 3569: 1, 3566: 1, 3562: 1, 3558: 1, 3557: 1,
3554: 1, 3546: 1, 3542: 1, 3537: 1, 3532: 1, 3516: 1, 3511: 1, 3509: 1,
3500: 1, 3498: 1, 3495: 1, 3494: 1, 3484: 1, 3480: 1, 3471: 1, 3467: 1,
3463: 1, 3462: 1, 3458: 1, 3453: 1, 3449: 1, 3446: 1, 3442: 1, 3439: 1,
3438: 1, 3430: 1, 3427: 1, 3426: 1, 3425: 1, 3422: 1, 3418: 1, 3414: 1,
3408: 1, 3407: 1, 3403: 1, 3390: 1, 3388: 1, 3381: 1, 3377: 1, 3370: 1,
3364: 1, 3361: 1, 3354: 1, 3349: 1, 3345: 1, 3343: 1, 3332: 1, 3331: 1,
3324: 1, 3323: 1, 3320: 1, 3319: 1, 3318: 1, 3310: 1, 3306: 1, 3304: 1,
3303: 1, 3300: 1, 3293: 1, 3287: 1, 3286: 1, 3281: 1, 3280: 1, 3279: 1,
3273: 1, 3270: 1, 3265: 1, 3263: 1, 3254: 1, 3253: 1, 3252: 1, 3251: 1,
3245: 1, 3241: 1, 3234: 1, 3233: 1, 3228: 1, 3226: 1, 3223: 1, 3219: 1,
3216: 1, 3211: 1, 3210: 1, 3209: 1, 3208: 1, 3207: 1, 3191: 1, 3182: 1,
3179: 1, 3176: 1, 3173: 1, 3168: 1, 3165: 1, 3163: 1, 3161: 1, 3156: 1,
3154: 1, 3153: 1, 3150: 1, 3143: 1, 3139: 1, 3137: 1, 3128: 1, 3120: 1,
3113: 1, 3108: 1, 3105: 1, 3102: 1, 3092: 1, 3083: 1, 3081: 1, 3077: 1,
3075: 1, 3071: 1, 3065: 1, 3061: 1, 3060: 1, 3056: 1, 3053: 1, 3051: 1,
3050: 1, 3041: 1, 3040: 1, 3037: 1, 3035: 1, 3028: 1, 3026: 1, 3023: 1,
3013: 1, 3009: 1, 3005: 1, 3000: 1, 2999: 1, 2998: 1, 2986: 1, 2982: 1,
2977: 1, 2974: 1, 2973: 1, 2972: 1, 2971: 1, 2968: 1, 2954: 1, 2952: 1,
2950: 1, 2949: 1, 2948: 1, 2940: 1, 2929: 1, 2928: 1, 2927: 1, 2916: 1,
2913: 1, 2909: 1, 2905: 1, 2904: 1, 2901: 1, 2898: 1, 2886: 1, 2878: 1,
2872: 1, 2867: 1, 2864: 1, 2863: 1, 2855: 1, 2848: 1, 2846: 1, 2842: 1,
2840: 1, 2838: 1, 2833: 1, 2826: 1, 2816: 1, 2812: 1, 2808: 1, 2807: 1,
2802: 1, 2797: 1, 2790: 1, 2782: 1, 2781: 1, 2776: 1, 2775: 1, 2757: 1,
2754: 1, 2747: 1, 2736: 1, 2735: 1, 2733: 1, 2721: 1, 2717: 1, 2714: 1,
2703: 1, 2700: 1, 2690: 1, 2689: 1, 2688: 1, 2687: 1, 2683: 1, 2673: 1,
2669: 1, 2657: 1, 2656: 1, 2654: 1, 2651: 1, 2646: 1, 2644: 1, 2641: 1,
2640: 1, 2637: 1, 2634: 1, 2632: 1, 2628: 1, 2625: 1, 2624: 1, 2623: 1,
2620: 1, 2614: 1, 2612: 1, 2608: 1, 2603: 1, 2601: 1, 2600: 1, 2598: 1,
2591: 1, 2589: 1, 2585: 1, 2579: 1, 2577: 1, 2576: 1, 2567: 1, 2563: 1,
2561: 1, 2558: 1, 2551: 1, 2549: 1, 2546: 1, 2545: 1, 2540: 1, 2533: 1,

2527: 1, 2526: 1, 2523: 1, 2520: 1, 2519: 1, 2517: 1, 2516: 1, 2510: 1,
2505: 1, 2500: 1, 2494: 1, 2491: 1, 2485: 1, 2480: 1, 2466: 1, 2462: 1,
2460: 1, 2456: 1, 2453: 1, 2451: 1, 2450: 1, 2448: 1, 2447: 1, 2445: 1,
2444: 1, 2443: 1, 2440: 1, 2437: 1, 2435: 1, 2432: 1, 2431: 1, 2428: 1,
2427: 1, 2424: 1, 2423: 1, 2417: 1, 2416: 1, 2415: 1, 2414: 1, 2411: 1,
2398: 1, 2393: 1, 2392: 1, 2390: 1, 2387: 1, 2383: 1, 2381: 1, 2377: 1,
2371: 1, 2370: 1, 2366: 1, 2363: 1, 2361: 1, 2357: 1, 2355: 1, 2353: 1,
2351: 1, 2350: 1, 2347: 1, 2346: 1, 2345: 1, 2343: 1, 2342: 1, 2341: 1,
2340: 1, 2332: 1, 2327: 1, 2317: 1, 2316: 1, 2311: 1, 2309: 1, 2305: 1,
2304: 1, 2302: 1, 2300: 1, 2299: 1, 2297: 1, 2291: 1, 2287: 1, 2285: 1,
2284: 1, 2283: 1, 2281: 1, 2277: 1, 2274: 1, 2273: 1, 2268: 1, 2265: 1,
2260: 1, 2258: 1, 2254: 1, 2252: 1, 2251: 1, 2250: 1, 2243: 1, 2242: 1,
2240: 1, 2239: 1, 2236: 1, 2235: 1, 2231: 1, 2227: 1, 2225: 1, 2224: 1,
2219: 1, 2215: 1, 2207: 1, 2206: 1, 2204: 1, 2201: 1, 2198: 1, 2197: 1,
2194: 1, 2192: 1, 2190: 1, 2188: 1, 2187: 1, 2170: 1, 2161: 1, 2158: 1,
2157: 1, 2151: 1, 2149: 1, 2148: 1, 2144: 1, 2143: 1, 2140: 1, 2135: 1,
2133: 1, 2130: 1, 2128: 1, 2126: 1, 2121: 1, 2119: 1, 2116: 1, 2098: 1,
2090: 1, 2089: 1, 2086: 1, 2080: 1, 2079: 1, 2077: 1, 2072: 1, 2069: 1,
2065: 1, 2064: 1, 2062: 1, 2061: 1, 2060: 1, 2056: 1, 2054: 1, 2053: 1,
2051: 1, 2050: 1, 2048: 1, 2047: 1, 2046: 1, 2042: 1, 2041: 1, 2039: 1,
2038: 1, 2034: 1, 2033: 1, 2031: 1, 2030: 1, 2027: 1, 2024: 1, 2022: 1,
2021: 1, 2020: 1, 2016: 1, 2015: 1, 2014: 1, 2011: 1, 2010: 1, 2006: 1,
2004: 1, 2003: 1, 2000: 1, 1996: 1, 1994: 1, 1987: 1, 1986: 1, 1983: 1,
1982: 1, 1981: 1, 1973: 1, 1972: 1, 1970: 1, 1969: 1, 1964: 1, 1963: 1,
1962: 1, 1950: 1, 1944: 1, 1942: 1, 1941: 1, 1931: 1, 1929: 1, 1927: 1,
1926: 1, 1924: 1, 1923: 1, 1922: 1, 1920: 1, 1918: 1, 1915: 1, 1909: 1,
1908: 1, 1907: 1, 1906: 1, 1900: 1, 1896: 1, 1893: 1, 1891: 1, 1888: 1,
1887: 1, 1884: 1, 1880: 1, 1878: 1, 1877: 1, 1876: 1, 1875: 1, 1874: 1,
1873: 1, 1871: 1, 1870: 1, 1868: 1, 1867: 1, 1866: 1, 1864: 1, 1856: 1,
1855: 1, 1849: 1, 1847: 1, 1846: 1, 1840: 1, 1835: 1, 1832: 1, 1829: 1,
1828: 1, 1821: 1, 1814: 1, 1806: 1, 1805: 1, 1802: 1, 1801: 1, 1799: 1,
1798: 1, 1795: 1, 1793: 1, 1790: 1, 1789: 1, 1787: 1, 1785: 1, 1783: 1,
1781: 1, 1777: 1, 1774: 1, 1771: 1, 1770: 1, 1769: 1, 1766: 1, 1760: 1,
1759: 1, 1758: 1, 1755: 1, 1749: 1, 1744: 1, 1737: 1, 1735: 1, 1733: 1,
1731: 1, 1728: 1, 1727: 1, 1726: 1, 1724: 1, 1720: 1, 1719: 1, 1715: 1,
1707: 1, 1705: 1, 1704: 1, 1703: 1, 1702: 1, 1701: 1, 1700: 1, 1698: 1,
1697: 1, 1695: 1, 1694: 1, 1690: 1, 1679: 1, 1676: 1, 1672: 1, 1667: 1,
1665: 1, 1663: 1, 1660: 1, 1658: 1, 1657: 1, 1655: 1, 1653: 1, 1652: 1,
1651: 1, 1650: 1, 1646: 1, 1639: 1, 1638: 1, 1634: 1, 1633: 1, 1631: 1,


```

1625: 1, 1624: 1, 1623: 1, 1620: 1, 1619: 1, 1618: 1, 1613: 1, 1612: 1,
1611: 1, 1606: 1, 1605: 1, 1602: 1, 1601: 1, 1593: 1, 1591: 1, 1590: 1,
1588: 1, 1582: 1, 1581: 1, 1580: 1, 1579: 1, 1577: 1, 1576: 1, 1574: 1,
1573: 1, 1572: 1, 1562: 1, 1561: 1, 1551: 1, 1548: 1, 1543: 1, 1541: 1,
1538: 1, 1537: 1, 1535: 1, 1533: 1, 1531: 1, 1530: 1, 1529: 1, 1524: 1,
1523: 1, 1520: 1, 1518: 1, 1517: 1, 1516: 1, 1515: 1, 1514: 1, 1513: 1,
1512: 1, 1507: 1, 1506: 1, 1503: 1, 1502: 1, 1501: 1, 1499: 1, 1498: 1,
1496: 1, 1494: 1, 1492: 1, 1484: 1, 1482: 1, 1481: 1, 1479: 1, 1475: 1,
1473: 1, 1471: 1, 1469: 1, 1465: 1, 1462: 1, 1458: 1, 1457: 1, 1454: 1,
1446: 1, 1445: 1, 1444: 1, 1442: 1, 1441: 1, 1440: 1, 1439: 1, 1438: 1,
1437: 1, 1432: 1, 1430: 1, 1428: 1, 1425: 1, 1424: 1, 1423: 1, 1422: 1,
1421: 1, 1419: 1, 1418: 1, 1416: 1, 1415: 1, 1414: 1, 1412: 1, 1411: 1,
1409: 1, 1407: 1, 1403: 1, 1397: 1, 1395: 1, 1394: 1, 1390: 1, 1389: 1,
1386: 1, 1385: 1, 1384: 1, 1383: 1, 1380: 1, 1379: 1, 1369: 1, 1368: 1,
1364: 1, 1363: 1, 1357: 1, 1355: 1, 1351: 1, 1347: 1, 1343: 1, 1339: 1,
1337: 1, 1332: 1, 1330: 1, 1329: 1, 1326: 1, 1324: 1, 1322: 1, 1320: 1,
1315: 1, 1314: 1, 1311: 1, 1308: 1, 1298: 1, 1297: 1, 1292: 1, 1283: 1,
1276: 1, 1274: 1, 1273: 1, 1268: 1, 1266: 1, 1259: 1, 1258: 1, 1254: 1,
1248: 1, 1245: 1, 1244: 1, 1240: 1, 1236: 1, 1229: 1, 1227: 1, 1225: 1,
1217: 1, 1215: 1, 1213: 1, 1212: 1, 1211: 1, 1209: 1, 1205: 1, 1204: 1,
1202: 1, 1199: 1, 1195: 1, 1190: 1, 1183: 1, 1178: 1, 1176: 1, 1174: 1,
1168: 1, 1166: 1, 1162: 1, 1160: 1, 1154: 1, 1152: 1, 1149: 1, 1141: 1,
1140: 1, 1139: 1, 1137: 1, 1134: 1, 1133: 1, 1130: 1, 1129: 1, 1126: 1,
1125: 1, 1124: 1, 1123: 1, 1120: 1, 1118: 1, 1117: 1, 1113: 1, 1109: 1,
1105: 1, 1098: 1, 1093: 1, 1092: 1, 1090: 1, 1085: 1, 1081: 1, 1077: 1,
1075: 1, 1073: 1, 1069: 1, 1068: 1, 1064: 1, 1063: 1, 1053: 1, 1045: 1,
1041: 1, 1029: 1, 1028: 1, 1026: 1, 1025: 1, 1023: 1, 1018: 1, 1006: 1,
1003: 1, 991: 1, 988: 1, 987: 1, 975: 1, 973: 1, 959: 1, 946: 1, 944:
1, 940: 1, 939: 1, 937: 1, 935: 1, 934: 1, 927: 1, 924: 1, 916: 1, 915:
1, 909: 1, 906: 1, 900: 1, 897: 1, 896: 1, 884: 1, 861: 1, 846: 1, 826:
1, 819: 1, 775: 1, 771: 1, 768: 1, 765: 1, 756: 1, 750: 1, 722: 1, 684:
1, 642: 1, 629: 1, 618: 1, 602: 1})

```

```

In [49]: # Train a Logistic regression+Calibration model using text features whi
cha re on-hot encoded
alpha = [10 ** x for x in range(-5, 1)]

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html

```

```

# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])      Fit linear model with S
tochastic Gradient Descent.
# predict(X)      Predict class labels for samples in X.

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

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

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

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[i], np.round(txt,3)), (alpha[i], cv_log_error_arra
y[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")

```

```

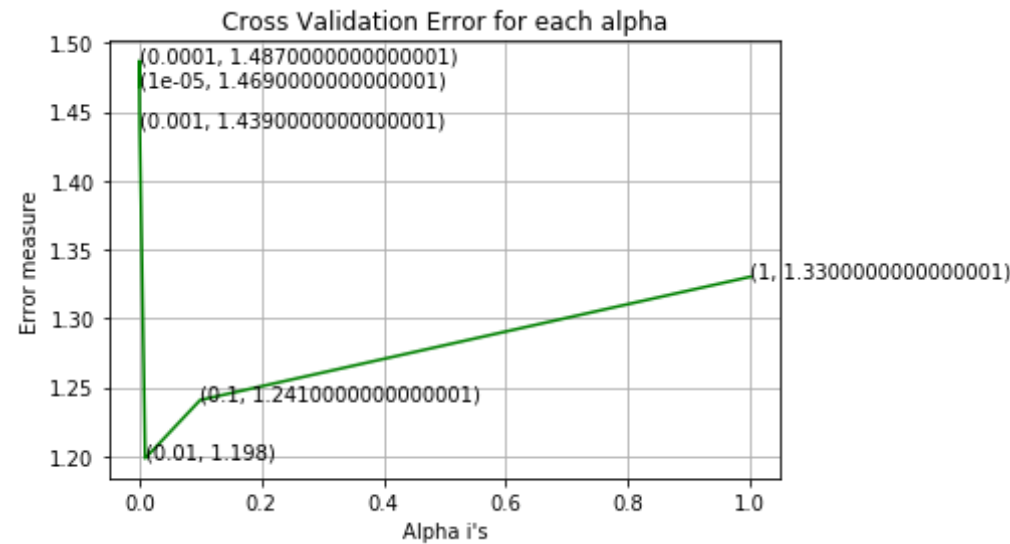
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

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

predict_y = sig_clf.predict_proba(train_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

For values of alpha = 1e-05 The log loss is: 1.46907964683
For values of alpha = 0.0001 The log loss is: 1.48695144385
For values of alpha = 0.001 The log loss is: 1.43904597963
For values of alpha = 0.01 The log loss is: 1.19836112542
For values of alpha = 0.1 The log loss is: 1.24068717183
For values of alpha = 1 The log loss is: 1.33015047883

```



For values of best alpha = 0.01 The train log loss is: 0.904093859744
 For values of best alpha = 0.01 The cross validation log loss is: 1.19836112542
 For values of best alpha = 0.01 The test log loss is: 1.26360583739

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

Ans. Yes, it seems like!

```
In [50]: def get_intersec_text(df):
df_text_vec = CountVectorizer(min_df=3, ngram_range=(1, 2))
df_text_fea = df_text_vec.fit_transform(df['TEXT'])
df_text_features = df_text_vec.get_feature_names()

df_text_fea_counts = df_text_fea.sum(axis=0).A1
df_text_fea_dict = dict(zip(list(df_text_features), df_text_fea_counts))

len1 = len(set(df_text_features))
len2 = len(set(train_text_features) & set(df_text_features))
return len1, len2
```

```
In [51]: len1,len2 = get_intersec_text(test_df)
print(np.round((len2/len1)*100, 3), "% of word of test data appeared in
train data")
len1,len2 = get_intersec_text(cv_df)
print(np.round((len2/len1)*100, 3), "% of word of Cross Validation appe
ared in train data")
```

95.598 % of word of test data appeared in train data
94.149 % of word of Cross Validation appeared in train data

4. Machine Learning Models

```
In [52]: #Data preparation for ML models.

#Misc. fonctionns for ML models

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

    # for calculating log_loss we willl provide the array of probabilit
ies belongs to each class
    print("Log loss :", log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    print("Number of mis-classified points :", np.count_nonzero((pred_y
- test_y))/test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

```
In [53]: def report_log_loss(train_x, train_y, test_x, test_y, clf):
    clf.fit(train_x, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x, train_y)
```

```
sig_clf_probs = sig_clf.predict_proba(test_x)
return log_loss(test_y, sig_clf_probs, eps=1e-15)
```

```
In [61]: # this function will be used just for naive bayes
# for the given indices, we will print the name of the features
# and we will check whether the feature present in the test point text
or not
def get_impfeature_names(indices, text, gene, var, no_features):
    gene_count_vec = CountVectorizer(ngram_range=(1,2))
    var_count_vec = CountVectorizer(ngram_range=(1,2))
    text_count_vec = CountVectorizer(min_df=3,ngram_range=(1,2))

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

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

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

```

        word_present += 1
        print(i, "Text feature [{}] present in test data point
        [{}]" .format(word,yes_no))

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

```

Stacking the three types of features

```

In [55]: # merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding,t
rain_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,tes
t_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_vari
ation_feature_onehotCoding))

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

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

cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_o
nehotCoding)).tocsr()

```

```

cv_y = np.array(list(cv_df['Class']))

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding, train_variation_feature_responseCoding))
test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding, test_variation_feature_responseCoding))
cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding, cv_variation_feature_responseCoding))

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

```

```

In [56]: print("One hot encoding features :")
print("(number of data points * number of features) in train data = ", train_x_onehotCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_onehotCoding.shape)
print("(number of data points * number of features) in cross validation data = ", cv_x_onehotCoding.shape)

```

```

One hot encoding features :
(number of data points * number of features) in train data = (2124, 768511)
(number of data points * number of features) in test data = (665, 768511)
(number of data points * number of features) in cross validation data = (532, 768511)

```

```

In [57]: print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCoding.shape)
print("(number of data points * number of features) in test data = ", test_x_responseCoding.shape)

```



```
print("(number of data points * number of features) in cross validation  
data =", cv_x_responseCoding.shape)
```

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

4.1. Base Line Model

4.1.1. Naive Bayes

4.1.1.1. Hyper parameter tuning

```
In [58]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html  
# -----  
# default paramters  
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)  
  
# some of methods of MultinomialNB()  
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y  
# predict(X)    Perform classification on an array of test vectors X.  
# predict_log_proba(X)    Return log-probability estimates for the test vector X.  
# -----  
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/  
# -----
```

```

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])           Get parameters for this estimator.
# predict(X)                    Predict the target of new samples.
# predict_proba(X)              Posterior probabilities of classification
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

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

fig, ax = plt.subplots()
ax.plot(np.log10(alpha), cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], str(txt)), (np.log10(alpha[i]), cv_log_error_array[i]))

```

```

plt.grid()
plt.xticks(np.log10(alpha))
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

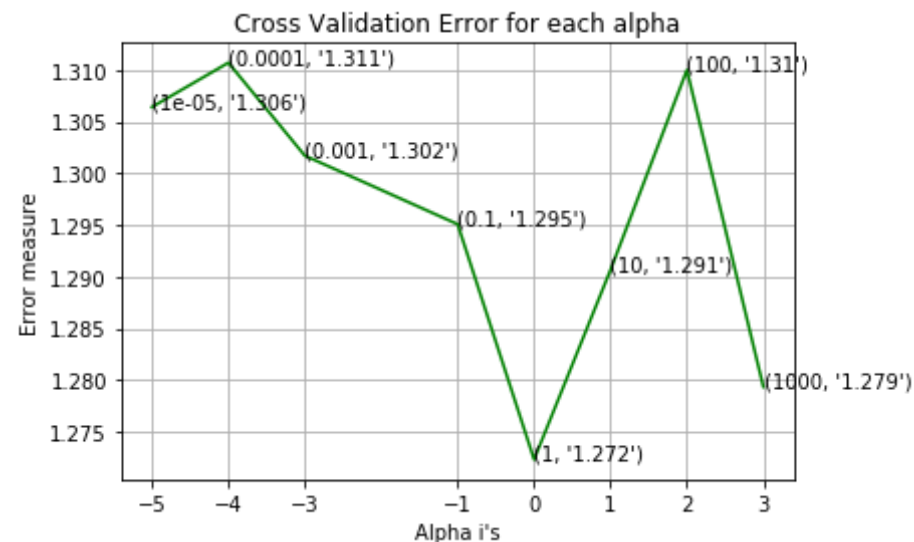
best_alpha = np.argmin(cv_log_error_array)
clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

for alpha = 1e-05
Log Loss : 1.30640395562
for alpha = 0.0001
Log Loss : 1.31069040358
for alpha = 0.001
Log Loss : 1.30170005859
for alpha = 0.1
Log Loss : 1.295064424
for alpha = 1
Log Loss : 1.27230913762
for alpha = 10
Log Loss : 1.29064538288
for alpha = 100

```

Log Loss : 1.30998157705
for alpha = 1000
Log Loss : 1.2793315103



For values of best alpha = 1 The train log loss is: 0.950103440601
For values of best alpha = 1 The cross validation log loss is: 1.27230913762
For values of best alpha = 1 The test log loss is: 1.29908404339

4.1.1.2. Testing the model with best hyper paramters

```
In [59]: # find more about Multinomial Naive base function here http://scikit-learn.org/stable/modules/generated/sklearn.naive\_bayes.MultinomialNB.html
# -----
# default paramters
# sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

# some of methods of MultinomialNB()
# fit(X, y[, sample_weight])    Fit Naive Bayes classifier according to X, y
```

```

# predict(X)    Perform classification on an array of test vectors X.
# predict_log_proba(X) Return log-probability estimates for the test v
ector X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/naive-bayes-algorithm-1/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)    Posterior probabilities of classification
# -----

clf = MultinomialNB(alpha=alpha[best_alpha])
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)
sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
# to avoid rounding error while multiplying probabilitites we use log-pro
bability estimates
print("Log Loss :", log_loss(cv_y, sig_clf_probs))
print("Number of missclassified point :", np.count_nonzero((sig_clf.pre
dict(cv_x_onehotCoding) - cv_y))/cv_y.shape[0])
plot_confusion_matrix(cv_y, sig_clf.predict(cv_x_onehotCoding.toarray
()))

```

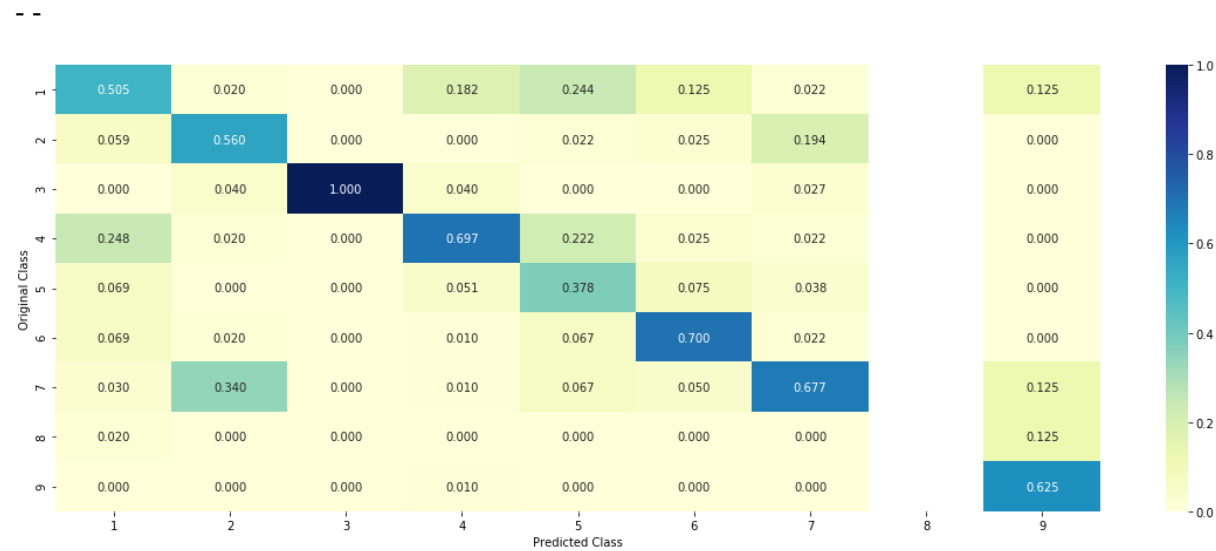
Log Loss : 1.27230913762

Number of missclassified point : 0.38533834586466165

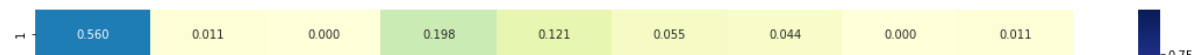
----- Confusion matrix -----

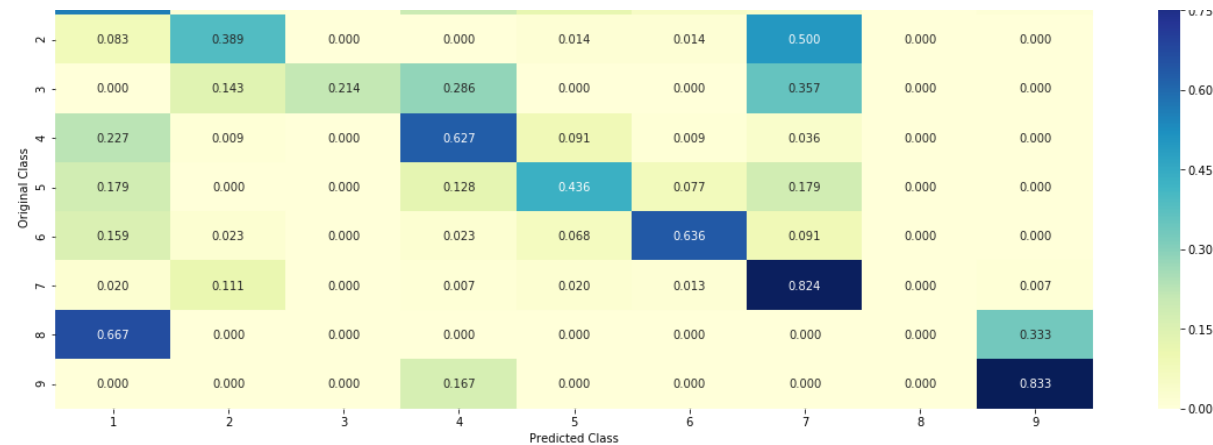


----- Precision matrix (Column Sum=1) -----



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





4.1.1.3. Feature Importance, Correctly classified point

```
In [62]: test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0733  0.0773  0.0175  0.6261  0.039
4  0.0331  0.1249  0.005  0.0033]]
```

```
Actual Class : 4
```

```
-----
10 Text feature [function] present in test data point [True]
11 Text feature [protein] present in test data point [True]
16 Text feature [mammalian] present in test data point [True]
17 Text feature [proteins] present in test data point [True]
```

19 Text feature [experiments] present in test data point [True]
20 Text feature [suppressor] present in test data point [True]
21 Text feature [missense] present in test data point [True]
22 Text feature [activity] present in test data point [True]
23 Text feature [acid] present in test data point [True]
27 Text feature [functional] present in test data point [True]
28 Text feature [results] present in test data point [True]
29 Text feature [amino] present in test data point [True]
32 Text feature [partially] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [critical] present in test data point [True]
38 Text feature [transfected] present in test data point [True]
40 Text feature [transfection] present in test data point [True]
41 Text feature [ability] present in test data point [True]
42 Text feature [type] present in test data point [True]
44 Text feature [indicate] present in test data point [True]
46 Text feature [thus] present in test data point [True]
48 Text feature [related] present in test data point [True]
49 Text feature [retained] present in test data point [True]
50 Text feature [co] present in test data point [True]
51 Text feature [stability] present in test data point [True]
52 Text feature [affect] present in test data point [True]
53 Text feature [made] present in test data point [True]
54 Text feature [whereas] present in test data point [True]
55 Text feature [abrogate] present in test data point [True]
56 Text feature [pten] present in test data point [True]
57 Text feature [two] present in test data point [True]
58 Text feature [either] present in test data point [True]
59 Text feature [indicates] present in test data point [True]
62 Text feature [important] present in test data point [True]
63 Text feature [shown] present in test data point [True]
65 Text feature [loss] present in test data point [True]
66 Text feature [caenorhabditis] present in test data point [True]
71 Text feature [generated] present in test data point [True]
73 Text feature [wild] present in test data point [True]
74 Text feature [vivo] present in test data point [True]
76 Text feature [yeast] present in test data point [True]
77 Text feature [containing] present in test data point [True]
80 Text feature [purified] present in test data point [True]


```

82 Text feature [conservative] present in test data point [True]
84 Text feature [tested] present in test data point [True]
85 Text feature [tagged] present in test data point [True]
86 Text feature [also] present in test data point [True]
87 Text feature [resulting] present in test data point [True]
88 Text feature [three] present in test data point [True]
89 Text feature [presented] present in test data point [True]
91 Text feature [assay] present in test data point [True]
92 Text feature [although] present in test data point [True]
94 Text feature [putative] present in test data point [True]
96 Text feature [system] present in test data point [True]
97 Text feature [deviation] present in test data point [True]
99 Text feature [associated] present in test data point [True]
Out of the top 100 features 56 are present in query point

```

4.1.1.4. Feature Importance, Incorrectly classified point

```

In [63]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)

```

```

Predicted Class : 4
Predicted Class Probabilities: [[ 0.0733  0.0773  0.0175  0.6261  0.039
4  0.0331  0.1249  0.005  0.0033]]
Actual Class : 4
-----
10 Text feature [function] present in test data point [True]
11 Text feature [protein] present in test data point [True]
16 Text feature [mammalian] present in test data point [True]

```

17 Text feature [proteins] present in test data point [True]
19 Text feature [experiments] present in test data point [True]
20 Text feature [suppressor] present in test data point [True]
21 Text feature [missense] present in test data point [True]
22 Text feature [activity] present in test data point [True]
23 Text feature [acid] present in test data point [True]
27 Text feature [functional] present in test data point [True]
28 Text feature [results] present in test data point [True]
29 Text feature [amino] present in test data point [True]
32 Text feature [partially] present in test data point [True]
34 Text feature [determined] present in test data point [True]
35 Text feature [critical] present in test data point [True]
38 Text feature [transfected] present in test data point [True]
40 Text feature [transfection] present in test data point [True]
41 Text feature [ability] present in test data point [True]
42 Text feature [type] present in test data point [True]
44 Text feature [indicate] present in test data point [True]
46 Text feature [thus] present in test data point [True]
48 Text feature [related] present in test data point [True]
49 Text feature [retained] present in test data point [True]
50 Text feature [co] present in test data point [True]
51 Text feature [stability] present in test data point [True]
52 Text feature [affect] present in test data point [True]
53 Text feature [made] present in test data point [True]
54 Text feature [whereas] present in test data point [True]
55 Text feature [abrogate] present in test data point [True]
56 Text feature [pten] present in test data point [True]
57 Text feature [two] present in test data point [True]
58 Text feature [either] present in test data point [True]
59 Text feature [indicates] present in test data point [True]
62 Text feature [important] present in test data point [True]
63 Text feature [shown] present in test data point [True]
65 Text feature [loss] present in test data point [True]
66 Text feature [caenorhabditis] present in test data point [True]
71 Text feature [generated] present in test data point [True]
73 Text feature [wild] present in test data point [True]
74 Text feature [vivo] present in test data point [True]
76 Text feature [yeast] present in test data point [True]
77 Text feature [containing] present in test data point [True]

```
80 Text feature [purified] present in test data point [True]
82 Text feature [conservative] present in test data point [True]
84 Text feature [tested] present in test data point [True]
85 Text feature [tagged] present in test data point [True]
86 Text feature [also] present in test data point [True]
87 Text feature [resulting] present in test data point [True]
88 Text feature [three] present in test data point [True]
89 Text feature [presented] present in test data point [True]
91 Text feature [assay] present in test data point [True]
92 Text feature [although] present in test data point [True]
94 Text feature [putative] present in test data point [True]
96 Text feature [system] present in test data point [True]
97 Text feature [deviation] present in test data point [True]
99 Text feature [associated] present in test data point [True]
Out of the top 100 features 56 are present in query point
```

4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```
In [64]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

```

le-1/
#-----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])           Get parameters for this estimator.
# predict(X)                    Predict the target of new samples.
# predict_proba(X)              Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [5, 11, 15, 21, 31, 41, 51, 99]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = KNeighborsClassifier(n_neighbors=i)
    clf.fit(train_x_responseCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_responseCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilities we use log-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')

```

```

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

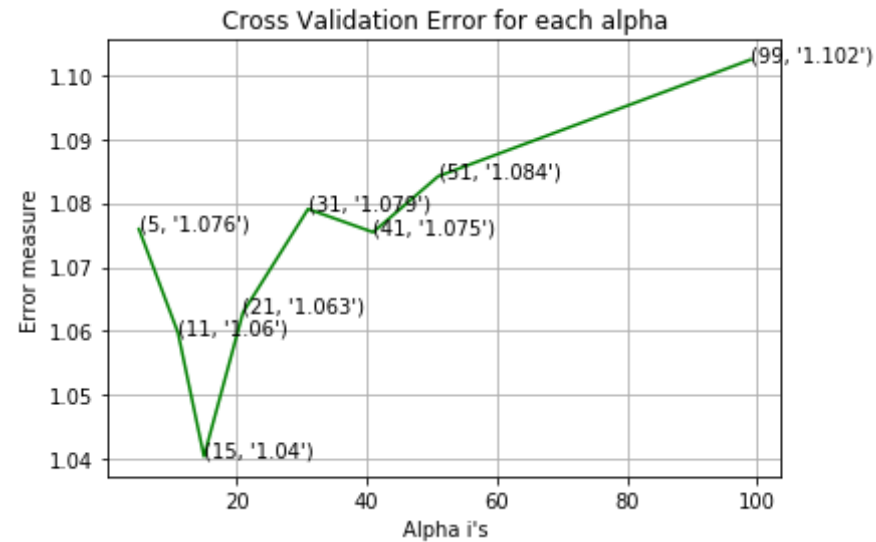
best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

for alpha = 5
Log Loss : 1.0759272314
for alpha = 11
Log Loss : 1.05977107249
for alpha = 15
Log Loss : 1.04042911086
for alpha = 21
Log Loss : 1.06291449647
for alpha = 31
Log Loss : 1.07904794826
for alpha = 41
Log Loss : 1.07542397053
for alpha = 51

```

Log Loss : 1.08415296093
for alpha = 99
Log Loss : 1.10234736376



For values of best alpha = 15 The train log loss is: 0.696666770146
For values of best alpha = 15 The cross validation log loss is: 1.04042911086
For values of best alpha = 15 The test log loss is: 1.08428999149

4.2.2. Testing the model with best hyper paramters

```
In [65]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=2,
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

# methods of
# fit(X, y) : Fit the model using X as training data and y as target va
```

```

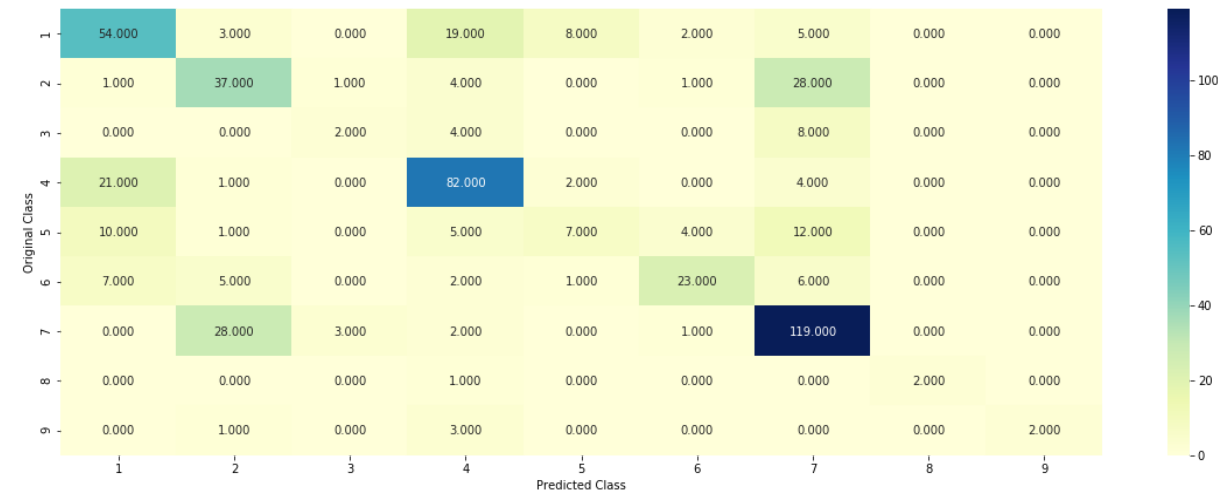
lues
# predict(X):Predict the class labels for the provided data
# predict_proba(X):Return probability estimates for the test data X.
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/k-nearest-neighbors-geometric-intuition-with-a-toy-example-1/
#-----
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCoding, cv_y, clf)

```

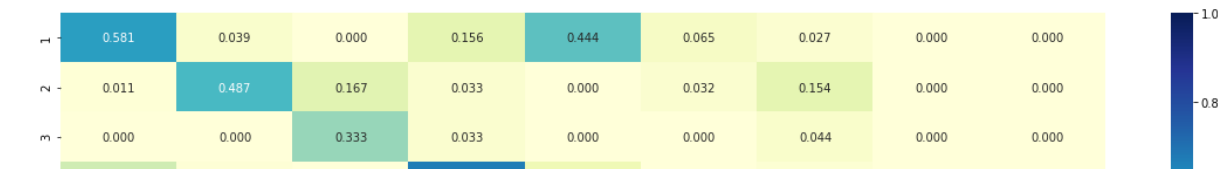
Log loss : 1.04042911086

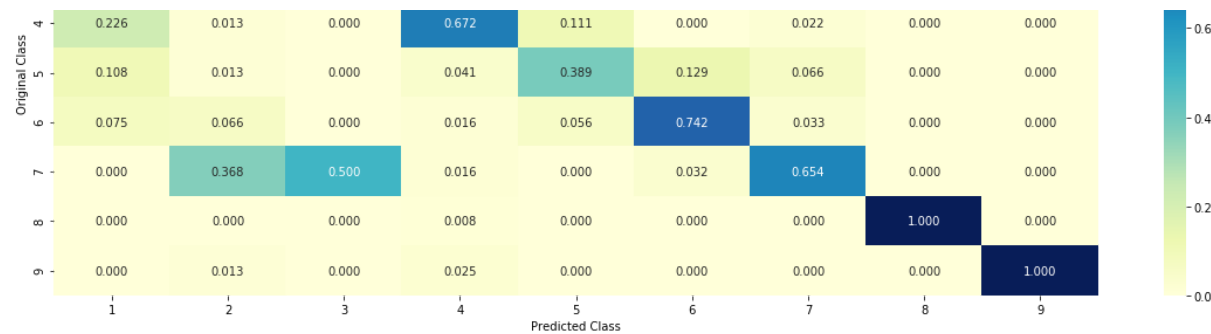
Number of mis-classified points : 0.38345864661654133

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

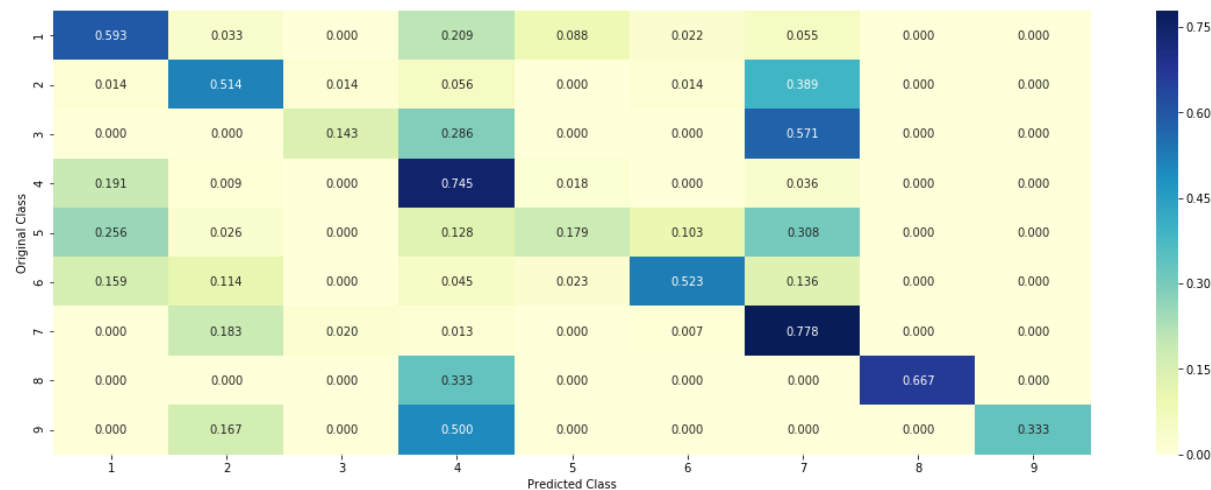


----- Precision matrix (Column Sum=1) -----





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



4.2.3.Sample Query point -1

```
In [66]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
predicted_cls = sig_clf.predict(test_x_responseCoding[0].reshape(1, -1))
print("Predicted Class :", predicted_cls[0])
```



```

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

```

Predicted Class : 4
 Actual Class : 4
 The 15 nearest neighbours of the test points belongs to classes [3 3
 3 3 4 4 4 4 4 4 4 4 4 4 4]
 Fequency of nearest points : Counter({4: 11, 3: 4})

4.2.4. Sample Query Point-2

```

In [67]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].resh
ape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neigh
bours of the test points belongs to classes",train_y[neighbors[1][0]])
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))

```

Predicted Class : 4
 Actual Class : 4
 the k value for knn is 15 and the nearest neighbours of the test points
 belongs to classes [3 3 3 3 4 4 4 4 4 4 4 4 4 4 4]
 Fequency of nearest points : Counter({4: 11, 3: 4})

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

```
In [68]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])      Fit linear model with Stochastic Gradient Descent.
# predict(X)      Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----

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

```

# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])           Get parameters for this estimator.
# predict(X)                    Predict the target of new samples.
# predict_proba(X)              Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [10 ** x for x in range(-6, 3)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(class_weight='balanced', alpha=i, penalty='l2',
loss='log', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilitites we use log
-probability estimates
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding, train_y)

```

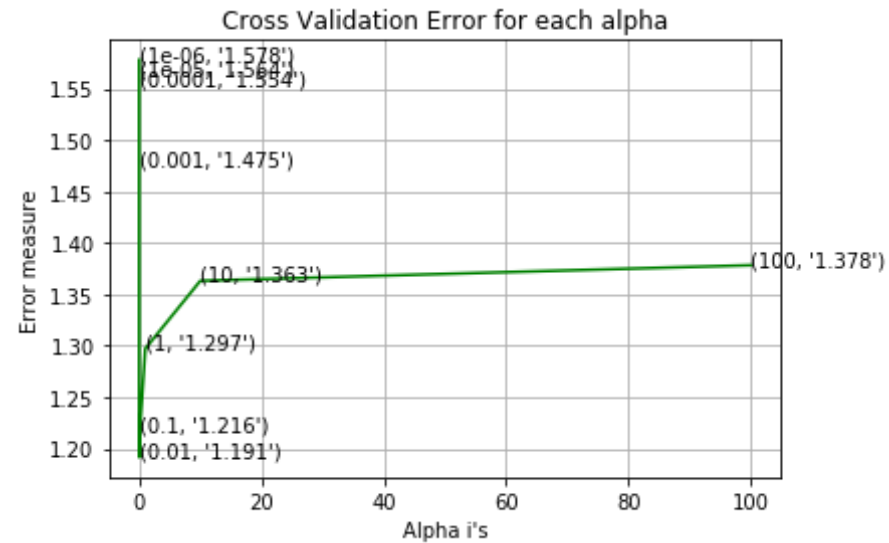
```

sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
      loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
      ))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
      dation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps
      =1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
      oss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

for alpha = 1e-06
Log Loss : 1.5781209566
for alpha = 1e-05
Log Loss : 1.56356907324
for alpha = 0.0001
Log Loss : 1.55443045766
for alpha = 0.001
Log Loss : 1.47519969696
for alpha = 0.01
Log Loss : 1.19141948592
for alpha = 0.1
Log Loss : 1.21647874996
for alpha = 1
Log Loss : 1.29709574047
for alpha = 10
Log Loss : 1.36311499631
for alpha = 100
Log Loss : 1.37829979759

```



For values of best alpha = 0.01 The train log loss is: 0.859336147034
 For values of best alpha = 0.01 The cross validation log loss is: 1.19141948592
 For values of best alpha = 0.01 The test log loss is: 1.23316439859

4.3.1.2. Testing the model with best hyper paramters

In [69]:

```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

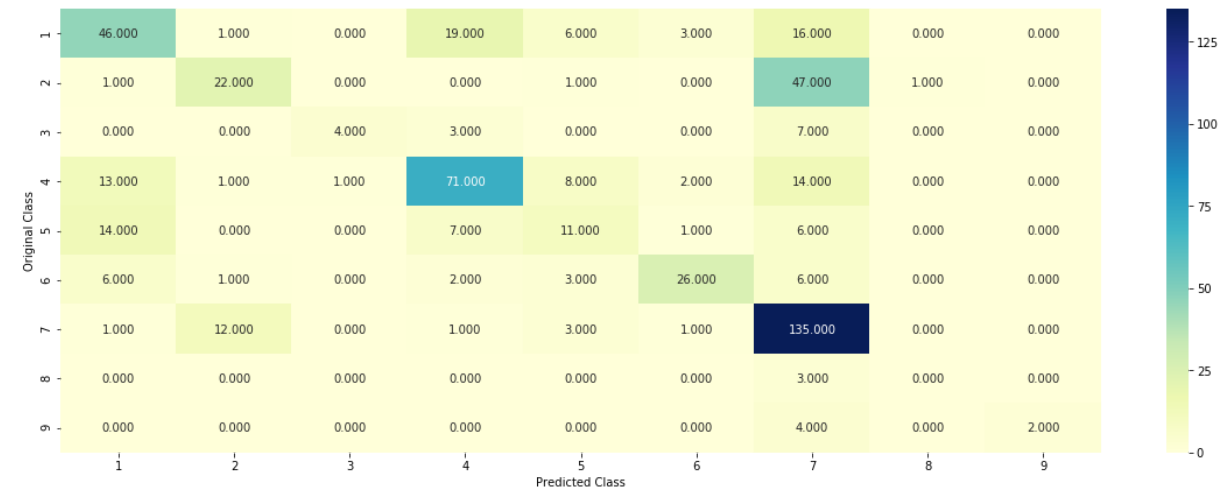
# some of methods
# fit(X, y[, coef_init, intercept_init, ...])    Fit linear model with Stochastic Gradient Descent.
# predict(X)    Predict class labels for samples in X.
```

```
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
#-----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_o
nehotCoding, cv_y, clf)
```

Log loss : 1.19141948592

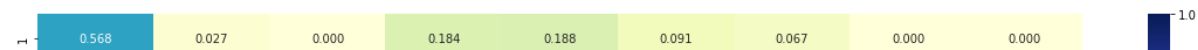
Number of mis-classified points : 0.4041353383458647

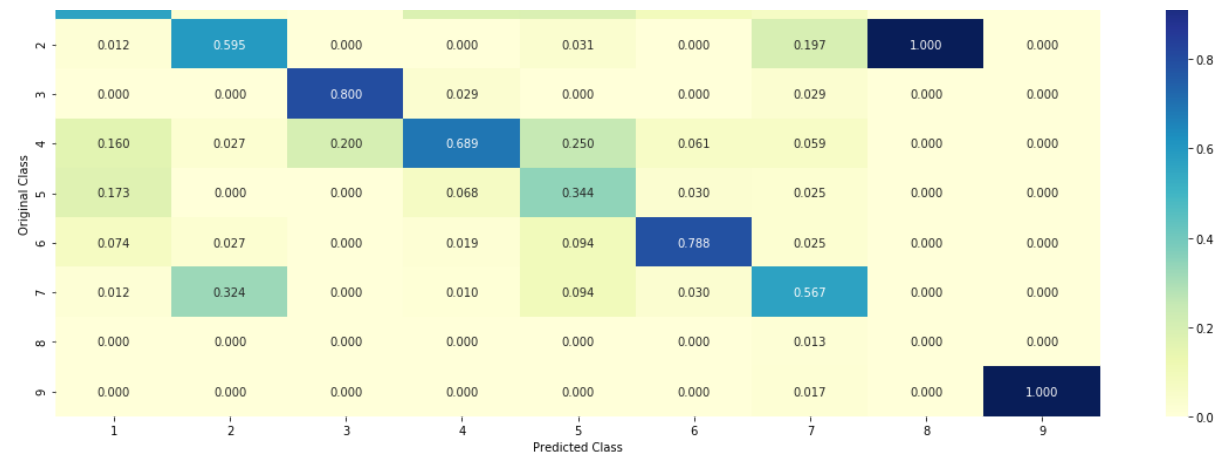
----- Confusion matrix -----



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

--





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



4.3.1.3. Feature Importance

```
In [70]: def get_imp_feature_names(text, indices, removed_ind = []):
          word_present = 0
          tabulte_list = []
          increasingorder_ind = 0
          for i in indices:
```

```

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

```

4.3.1.3.1. Correctly Classified point

```

In [71]: # from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='log', random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index

```



```
],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 4

Predicted Class Probabilities: [[0.0344 0.0478 0.0693 0.7868 0.0194 0.0041 0.0275 0.0058 0.0049]]

Actual Class : 4

Out of the top 500 features 0 are present in query point

4.3.1.3.2. Incorrectly Classified point

```
In [72]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)
```

Predicted Class : 4

Predicted Class Probabilities: [[0.0344 0.0478 0.0693 0.7868 0.0194 0.0041 0.0275 0.0058 0.0049]]

Actual Class : 4

Out of the top 500 features 0 are present in query point

4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```

In [73]: # read more about SGDClassifier() at http://scikit-learn.org/stable/mod
          ules/generated/sklearn.linear_model.SGDClassifier.html
          # -----
          # default parameters
          # SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
          5, fit_intercept=True, max_iter=None, tol=None,
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
          arning_rate='optimal', eta0=0.0, power_t=0.5,
          # class_weight=None, warm_start=False, average=False, n_iter=None)

          # some of methods
          # fit(X, y[, coef_init, intercept_init, ...])      Fit linear model with S
          tochastic Gradient Descent.
          # predict(X)      Predict class labels for samples in X.

          #-----
          # video link: https://www.appliedaicourse.com/course/applied-ai-course-
          online/lessons/geometric-intuition-1/
          #-----

          # find more about CalibratedClassifierCV here at http://scikit-learn.or
          g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
          tml
          # -----
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
          d='sigmoid', cv=3)
          #
          # some of the methods of CalibratedClassifierCV()
          # fit(X, y[, sample_weight])      Fit the calibrated model
          # get_params([deep])      Get parameters for this estimator.
          # predict(X)      Predict the target of new samples.
          # predict_proba(X)      Posterior probabilities of classification
          #-----
          # video link:
          #-----

```

```

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state
=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.
classes_, eps=1e-15))
    print("Log Loss :", log_loss(cv_y, sig_clf_probs))

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

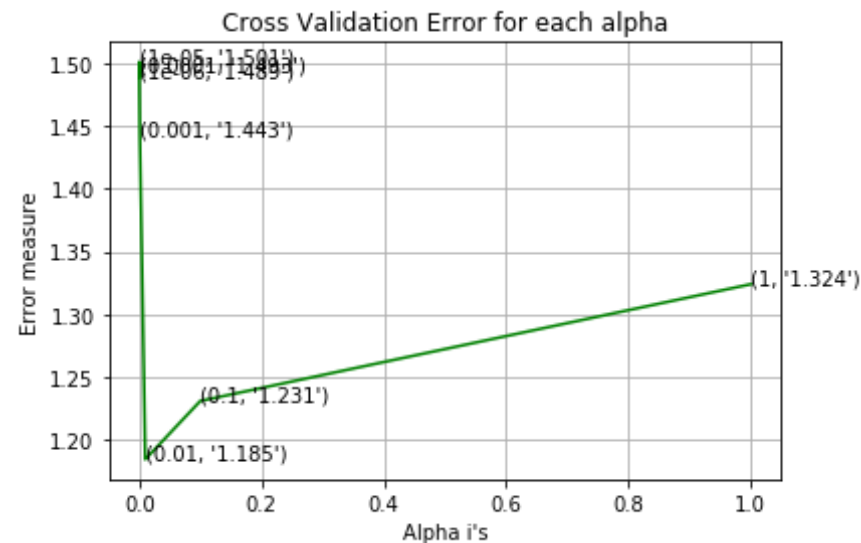
best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

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

```

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

```
for alpha = 1e-06
Log Loss : 1.48893984609
for alpha = 1e-05
Log Loss : 1.50087072601
for alpha = 0.0001
Log Loss : 1.49317375722
for alpha = 0.001
Log Loss : 1.44332210286
for alpha = 0.01
Log Loss : 1.18453069342
for alpha = 0.1
Log Loss : 1.2309810384
for alpha = 1
Log Loss : 1.32386052841
```



For values of best alpha = 0.01 The train log loss is: 0.881938455152
 For values of best alpha = 0.01 The cross validation log loss is: 1.18

453069342

.....
For values of best alpha = 0.01 The test log loss is: 1.25154816383

4.3.2.2. Testing model with best hyper parameters

```
In [74]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])      Fit linear model with Stochastic Gradient Descent.
# predict(X)      Predict class labels for samples in X.

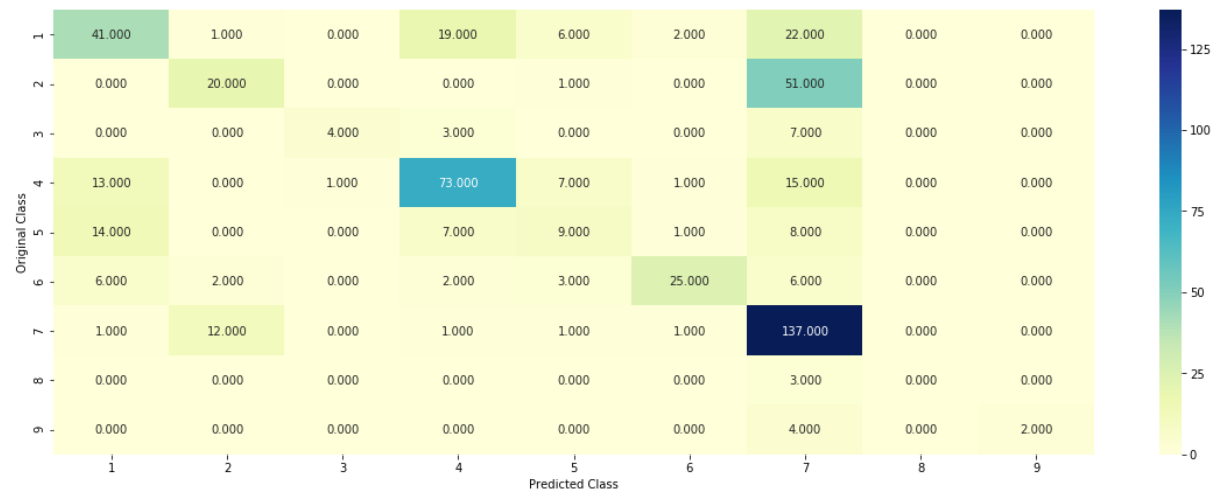
#-----
# video link:
#-----

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

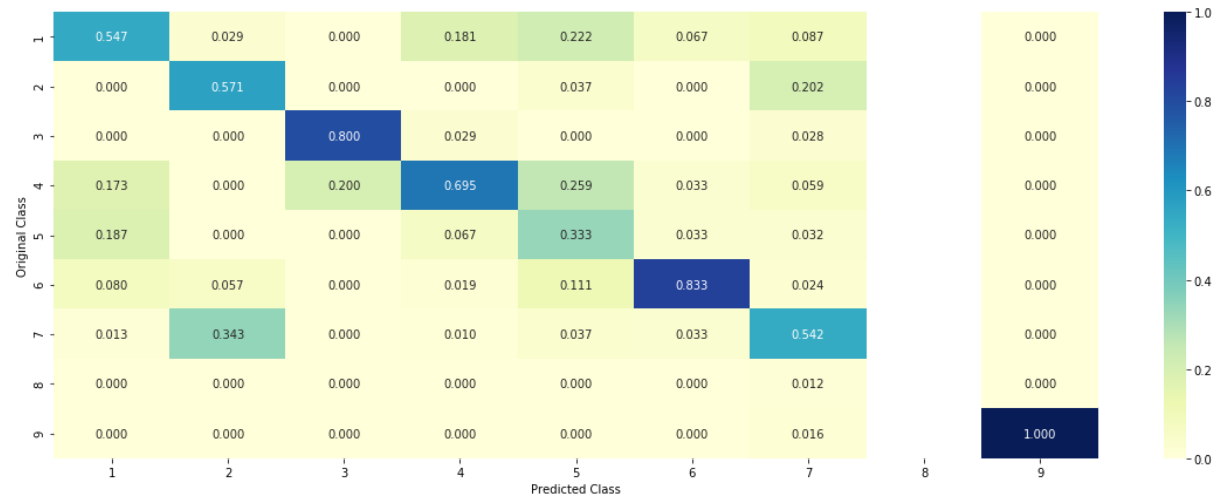
Log loss : 1.18453069342

Number of mis-classified points : 0.41541353383458646

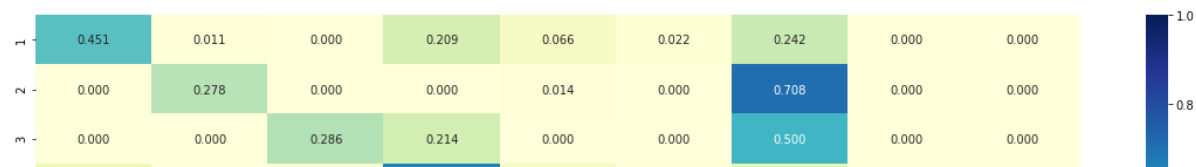
----- Confusion matrix -----

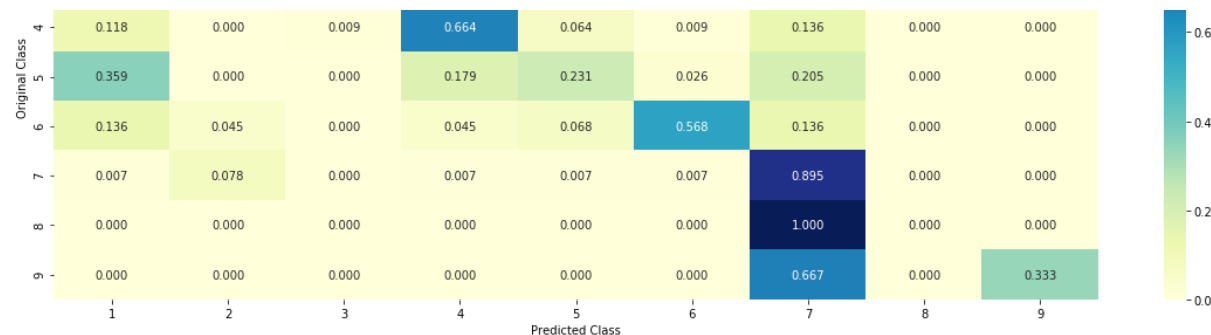


----- Precision matrix (Column Sum=1) -----



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





4.3.2.3. Feature Importance, Correctly Classified point

```
In [75]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log',
        random_state=42)
        clf.fit(train_x_onehotCoding,train_y)
        test_point_index = 1
        no_feature = 500
        predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
        print("Predicted Class :", predicted_cls[0])
        print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
        test_x_onehotCoding[test_point_index]),4))
        print("Actual Class :", test_y[test_point_index])
        indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
        print("-"*50)
        get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
        ],test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
        _point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0457  0.0509  0.0281  0.7966  0.015
8  0.0034  0.0532  0.0054  0.0008]]
Actual Class : 4
```

```
-----
Out of the top 500 features 0 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [76]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index]
,test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test
_point_index], no_feature)
```

```
Predicted Class : 4
Predicted Class Probabilities: [[ 0.0457  0.0509  0.0281  0.7966  0.015
8  0.0034  0.0532  0.0054  0.0008]]
Actual Class : 4
-----
Out of the top 500 features 0 are present in query point
```

4.4. Linear Support Vector Machines

4.4.1. Hyper paramter tuning

```
In [77]: # read more about support vector machines with linear kernals here htt
p://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

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



```

n training data.
# predict(X)    Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])    Get parameters for this estimator.
# predict(X)    Predict the target of new samples.
# predict_proba(X)    Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [10 ** x for x in range(-5, 3)]
cv_log_error_array = []
for i in alpha:
    print("for C =", i)
    # clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
    clf = SGDClassifier( class_weight='balanced', alpha=i, penalty='l2', loss='hinge', random_state=42)
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.

```

```

classes_, eps=1e-15))
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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

best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i,kernel='linear',probability=True, class_weight='balance
d')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], p
enalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log
loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15
))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross vali
dation log loss is:",log_loss(y_cv, predict_y, labels=clf.classes_, eps
=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log l
oss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

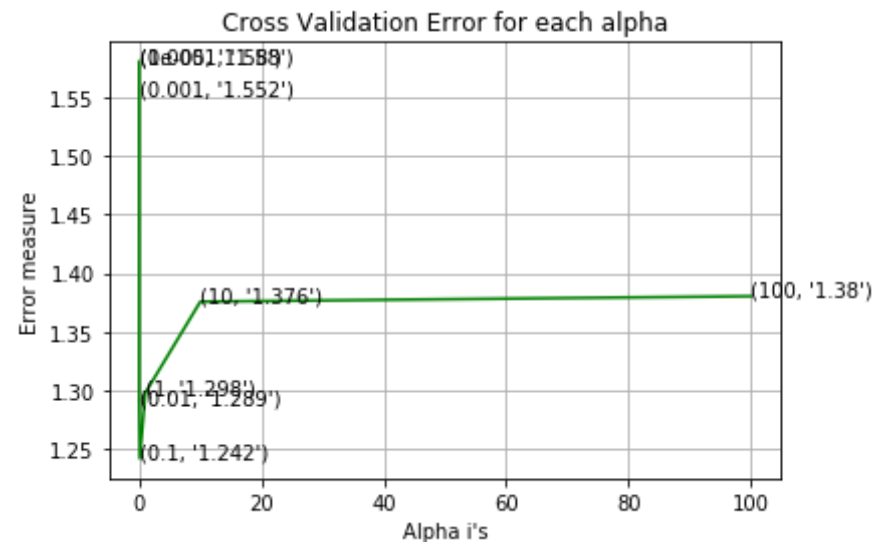
for C = 1e-05
Log Loss : 1.5802838133
for C = 0.0001
Log Loss : 1.57993978043
for C = 0.001

```

```

Log Loss : 1.55242899157
for C = 0.01
Log Loss : 1.28904430277
for C = 0.1
Log Loss : 1.24204144284
for C = 1
Log Loss : 1.29799187033
for C = 10
Log Loss : 1.37573829454
for C = 100
Log Loss : 1.38036166895

```



```

For values of best alpha = 0.1 The train log loss is: 0.875406574329
For values of best alpha = 0.1 The cross validation log loss is: 1.242
04144284
For values of best alpha = 0.1 The test log loss is: 1.23768746384

```

4.4.2. Testing model with best hyper parameters

```

In [78]: # read more about support vector machines with linear kernals here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

```

```

# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

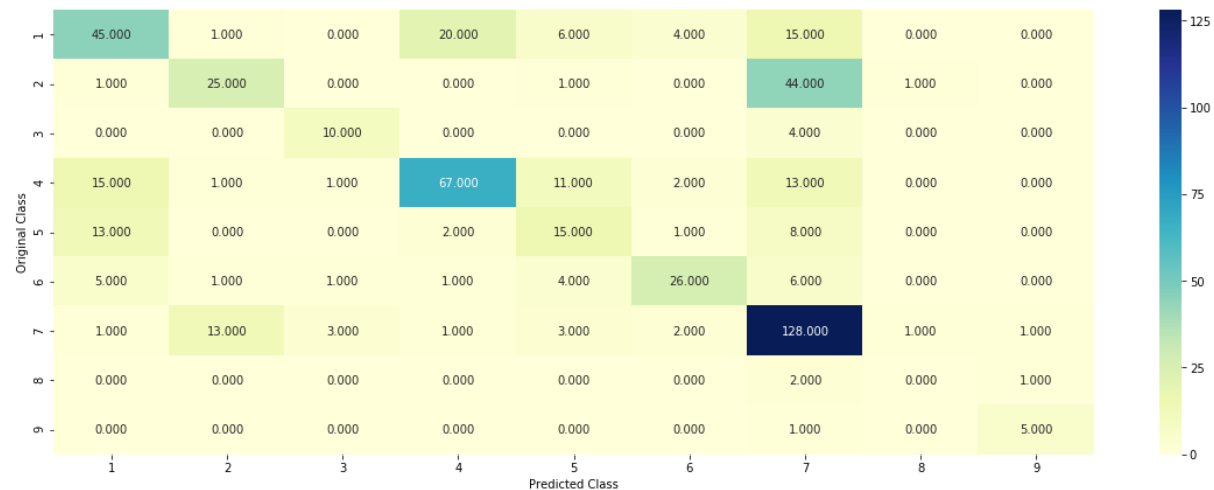
# clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class
_weight='balanced')
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42,class_weight='balanced')
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)

```

Log loss : 1.24204144284

Number of mis-classified points : 0.3966165413533835

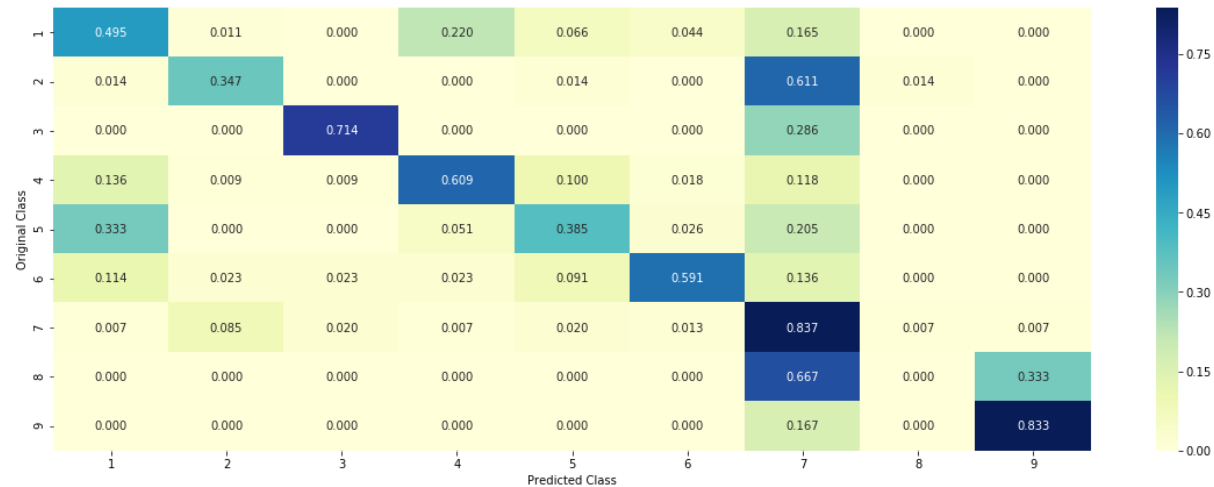
----- Confusion matrix -----



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



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



4.3.3. Feature Importance

4.3.3.1. For Correctly classified point

```
In [79]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge'
, random_state=42)
clf.fit(train_x_onehotCoding,train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
```

```

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

```

```

Predicted Class : 4
Predicted Class Probabilities: [[ 0.0435  0.0444  0.0127  0.7999  0.023
 0.0106  0.0584  0.0045  0.0029]]
Actual Class : 4
-----
Out of the top 500 features 0 are present in query point

```

4.3.3.2. For Incorrectly classified point

```

In [80]: test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.coef_)[predicted_cls-1][:,:no_feature]
print("-"*50)
get_impfeature_names(indices[0], test_df['TEXT'].iloc[test_point_index],
test_df['Gene'].iloc[test_point_index],test_df['Variation'].iloc[test_point_index], no_feature)

```

```

Predicted Class : 4
Predicted Class Probabilities: [[ 0.0435  0.0444  0.0127  0.7999  0.023
 0.0106  0.0584  0.0045  0.0029]]
Actual Class : 4
-----
Out of the top 500 features 0 are present in query point

```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

```
In [81]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)    Perform classification on samples in X.

# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# -----

# find more about CalibratedClassifierCV here at http://scikit-learn.or
g/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.h
tml
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, metho
d='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
```



```

# get_params([deep])    Get parameters for this estimator.
# predict(X)           Predict the target of new samples.
# predict_proba(X)      Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [100,200,500,1000,2000]
max_depth = [5, 10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_onehotCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

'''fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[: ,None],np.array(max_depth)[None]).ra
vel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/2)],max_depth[int(i%2)],str(txt)), (featur
es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri

```

```

terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_,
eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.cl
asses_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_onehotCoding)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The
test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, ep
s=1e-15))

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.29721159435
for n_estimators = 100 and max depth = 10
Log Loss : 1.20236359215
for n_estimators = 200 and max depth = 5
Log Loss : 1.28607309051
for n_estimators = 200 and max depth = 10
Log Loss : 1.19286343421
for n_estimators = 500 and max depth = 5
Log Loss : 1.27671411317
for n_estimators = 500 and max depth = 10
Log Loss : 1.18915649882
for n_estimators = 1000 and max depth = 5
Log Loss : 1.28041032711
for n_estimators = 1000 and max depth = 10
Log Loss : 1.19049100154
for n_estimators = 2000 and max depth = 5
Log Loss : 1.27825571382
for n_estimators = 2000 and max depth = 10
Log Loss : 1.18889181968
For values of best estimator = 2000 The train log loss is: 0.883427280

```

015

For values of best estimator = 2000 The cross validation log loss is:
1.18893137243

For values of best estimator = 2000 The test log loss is: 1.2213657215
6

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

```
In [82]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)    Perform classification on samples in X.

# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

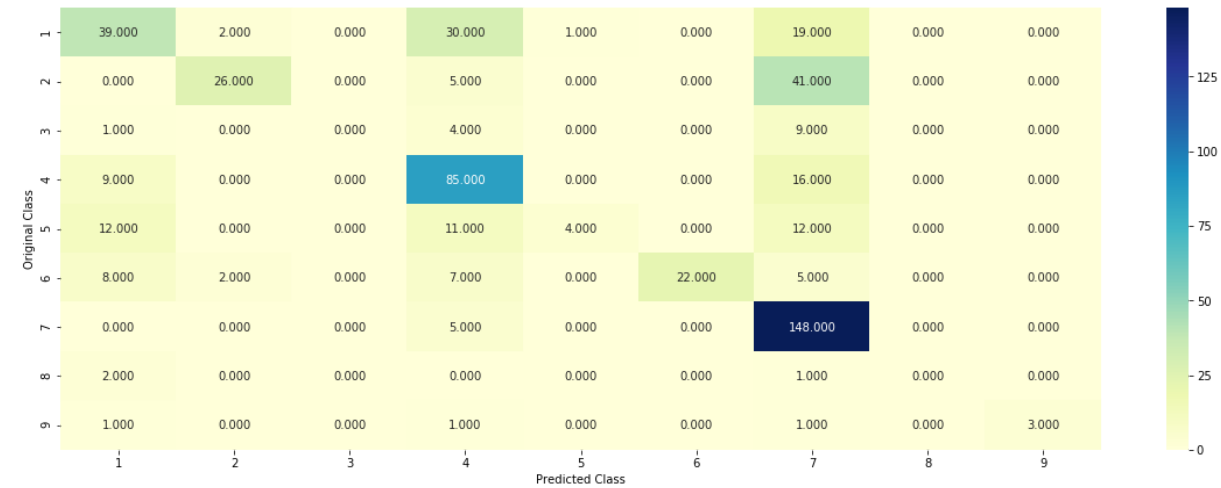
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# -----

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], cri
terion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42,
n_jobs=-1)
predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_on
ehotCoding,cv_y, clf)
```

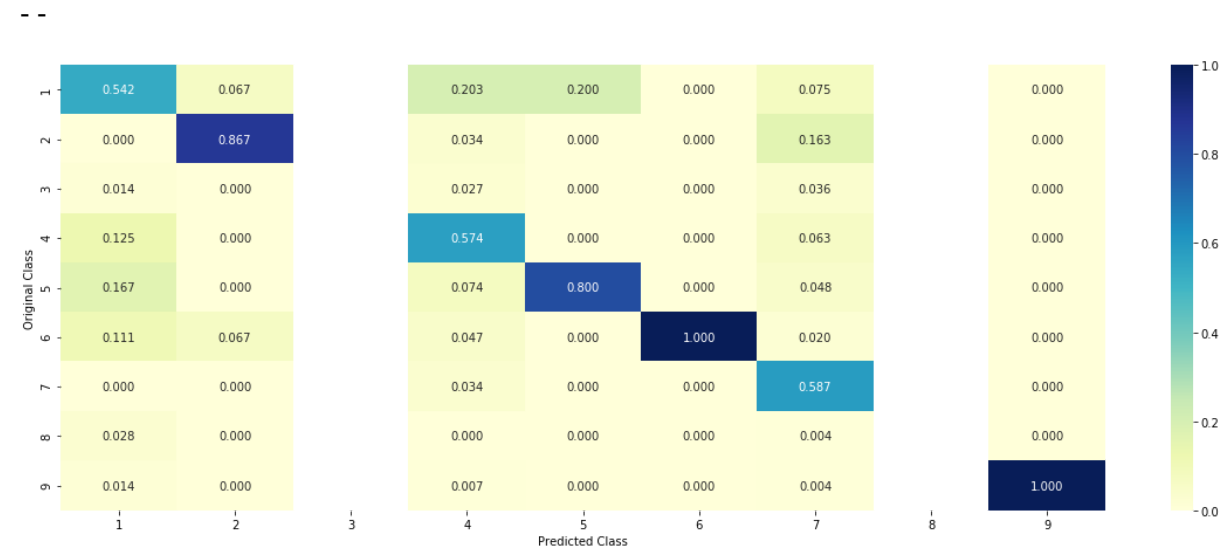
Log loss : 1.18890451569

Number of mis-classified points : 0.38533834586466165

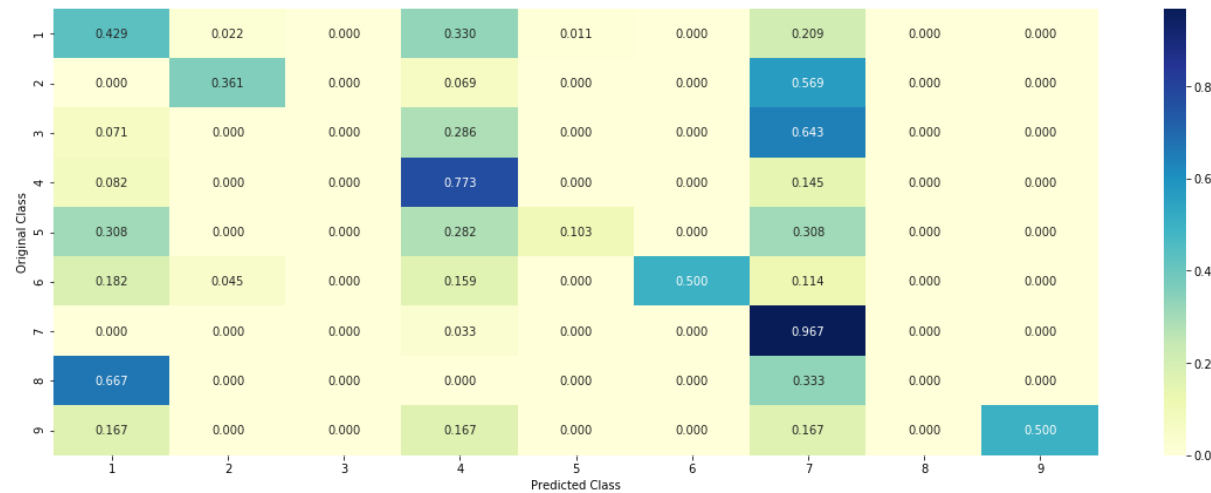
----- Confusion matrix -----



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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [83]: # test_point_index = 10
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=max_depth[int(best_alpha%2)], random_state=42, n_jobs=-1)
clf.fit(train_x_onehotCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_onehotCoding, train_y)

test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCoding[test_point_index]),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
```

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

Predicted Class : 4

Predicted Class Probabilities: [[0.0178 0.0054 0.1195 0.7964 0.0294 0.0203 0.0048 0.0031 0.0033]]

Actual Class : 4

1 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
6 Text feature [missense] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
12 Text feature [activation] present in test data point [True]
16 Text feature [activating] present in test data point [True]
19 Text feature [phosphorylation] present in test data point [True]
20 Text feature [akt] present in test data point [True]
24 Text feature [function] present in test data point [True]
25 Text feature [pathogenic] present in test data point [True]
29 Text feature [loss] present in test data point [True]
30 Text feature [expressing] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
35 Text feature [growth] present in test data point [True]
36 Text feature [kinases] present in test data point [True]
43 Text feature [patients] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
47 Text feature [stability] present in test data point [True]
54 Text feature [cells] present in test data point [True]
64 Text feature [functional] present in test data point [True]
67 Text feature [cell] present in test data point [True]
69 Text feature [deleterious] present in test data point [True]
78 Text feature [suppressor] present in test data point [True]
79 Text feature [inhibition] present in test data point [True]
86 Text feature [nonsense] present in test data point [True]
97 Text feature [unstable] present in test data point [True]
Out of the top 100 features 27 are present in query point

4.5.3.2. Inorrectly Classified point

```
In [84]: test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_onehotCoding[test_point_index]),4))
print("Actuall Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_po
int_index],test_df['Gene'].iloc[test_point_index],test_df['Variation'].
iloc[test_point_index], no_feature)
```

Predicted Class : 4

Predicted Class Probabilities: [[0.0178 0.0054 0.1195 0.7964 0.029
4 0.0203 0.0048 0.0031 0.0033]]

Actuall Class : 4

1 Text feature [kinase] present in test data point [True]
2 Text feature [tyrosine] present in test data point [True]
6 Text feature [missense] present in test data point [True]
10 Text feature [oncogenic] present in test data point [True]
11 Text feature [signaling] present in test data point [True]
12 Text feature [activation] present in test data point [True]
16 Text feature [activating] present in test data point [True]
19 Text feature [phosphorylation] present in test data point [True]
20 Text feature [akt] present in test data point [True]
24 Text feature [function] present in test data point [True]
25 Text feature [pathogenic] present in test data point [True]
29 Text feature [loss] present in test data point [True]
30 Text feature [expressing] present in test data point [True]
31 Text feature [therapy] present in test data point [True]
35 Text feature [growth] present in test data point [True]
36 Text feature [kinases] present in test data point [True]
43 Text feature [patients] present in test data point [True]
46 Text feature [downstream] present in test data point [True]
47 Text feature [stability] present in test data point [True]

```
54 Text feature [cells] present in test data point [True]
64 Text feature [functional] present in test data point [True]
67 Text feature [cell] present in test data point [True]
69 Text feature [deleterious] present in test data point [True]
78 Text feature [suppressor] present in test data point [True]
79 Text feature [inhibition] present in test data point [True]
86 Text feature [nonsense] present in test data point [True]
97 Text feature [unstable] present in test data point [True]
Out of the top 100 features 27 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

```
In [85]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba (X)    Perform classification on samples in X.

# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----
```



```

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method='sigmoid', cv=3)
#
# some of the methods of CalibratedClassifierCV()
# fit(X, y[, sample_weight])    Fit the calibrated model
# get_params([deep])           Get parameters for this estimator.
# predict(X)                    Predict the target of new samples.
# predict_proba(X)              Posterior probabilities of classification
#-----
# video link:
#-----

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini',
max_depth=j, random_state=42, n_jobs=-1)
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=
clf.classes_, eps=1e-15))
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
    ...
fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[: ,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (featur

```

```

es[i],cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()
'''

best_alpha = np.argmin(cv_log_error_array)
clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], cri
terion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42,
n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

predict_y = sig_clf.predict_proba(train_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tra
in log loss is:",log_loss(y_train, predict_y, labels=clf.classes_, eps=
1e-15))
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cro
ss validation log loss is:",log_loss(y_cv, predict_y, labels=clf.classe
s_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The tes
t log loss is:",log_loss(y_test, predict_y, labels=clf.classes_, eps=1e
-15))

for n_estimators = 10 and max depth = 2
Log Loss : 2.1139487959
for n_estimators = 10 and max depth = 3
Log Loss : 1.67126092632
for n_estimators = 10 and max depth = 5
Log Loss : 1.54388126641
for n_estimators = 10 and max depth = 10
Log Loss : 1.96831185841
for n_estimators = 50 and max depth = 2
Log Loss : 1.75055554888
for n_estimators = 50 and max depth = 3

```

```
Log Loss : 1.34215789922
for n_estimators = 50 and max depth = 5
Log Loss : 1.38567764095
for n_estimators = 50 and max depth = 10
Log Loss : 1.5589362555
for n_estimators = 100 and max depth = 2
Log Loss : 1.49145806595
for n_estimators = 100 and max depth = 3
Log Loss : 1.42211787609
for n_estimators = 100 and max depth = 5
Log Loss : 1.25058440571
for n_estimators = 100 and max depth = 10
Log Loss : 1.5353701171
for n_estimators = 200 and max depth = 2
Log Loss : 1.63259164865
for n_estimators = 200 and max depth = 3
Log Loss : 1.43947001448
for n_estimators = 200 and max depth = 5
Log Loss : 1.26435546851
for n_estimators = 200 and max depth = 10
Log Loss : 1.50525422716
for n_estimators = 500 and max depth = 2
Log Loss : 1.65479384728
for n_estimators = 500 and max depth = 3
Log Loss : 1.50569422654
for n_estimators = 500 and max depth = 5
Log Loss : 1.29154926332
for n_estimators = 500 and max depth = 10
Log Loss : 1.53857634069
for n_estimators = 1000 and max depth = 2
Log Loss : 1.62230825303
for n_estimators = 1000 and max depth = 3
Log Loss : 1.49511867229
for n_estimators = 1000 and max depth = 5
Log Loss : 1.29127337825
for n_estimators = 1000 and max depth = 10
Log Loss : 1.54362499961
For values of best alpha = 100 The train log loss is: 0.0480421682522
For values of best alpha = 100 The cross validation log loss is: 1.250
```

50757375

For values of best alpha = 100 The test log loss is: 1.278378762

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

```
In [86]: # -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# predict_proba(X)    Perform classification on samples in X.

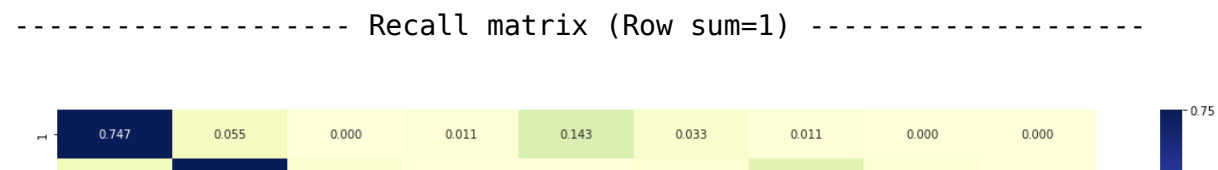
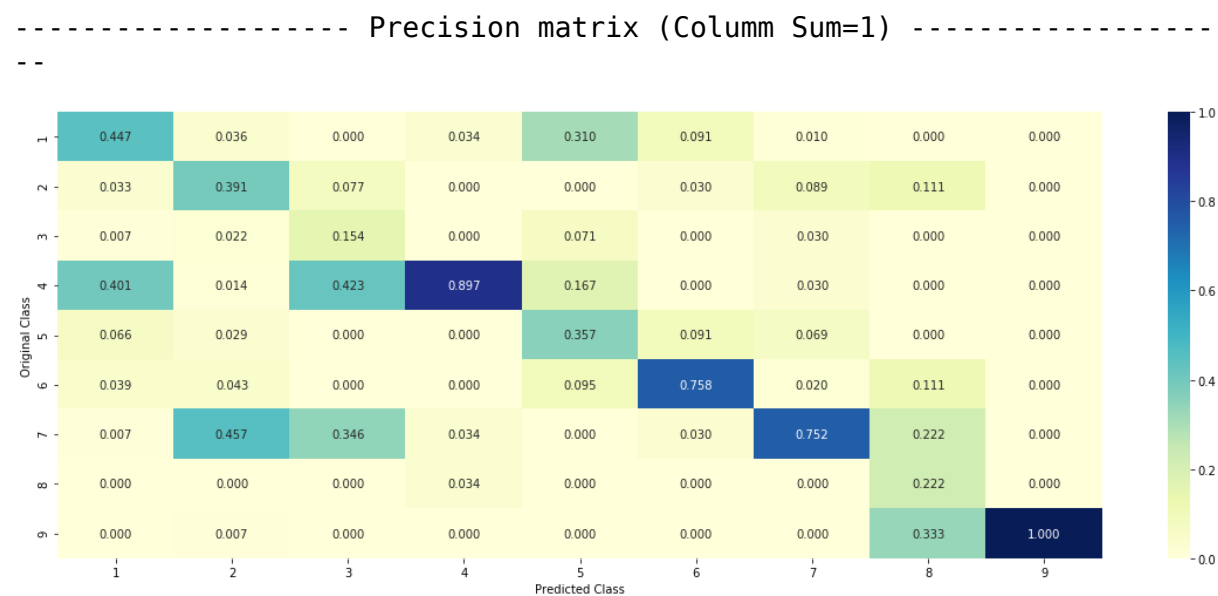
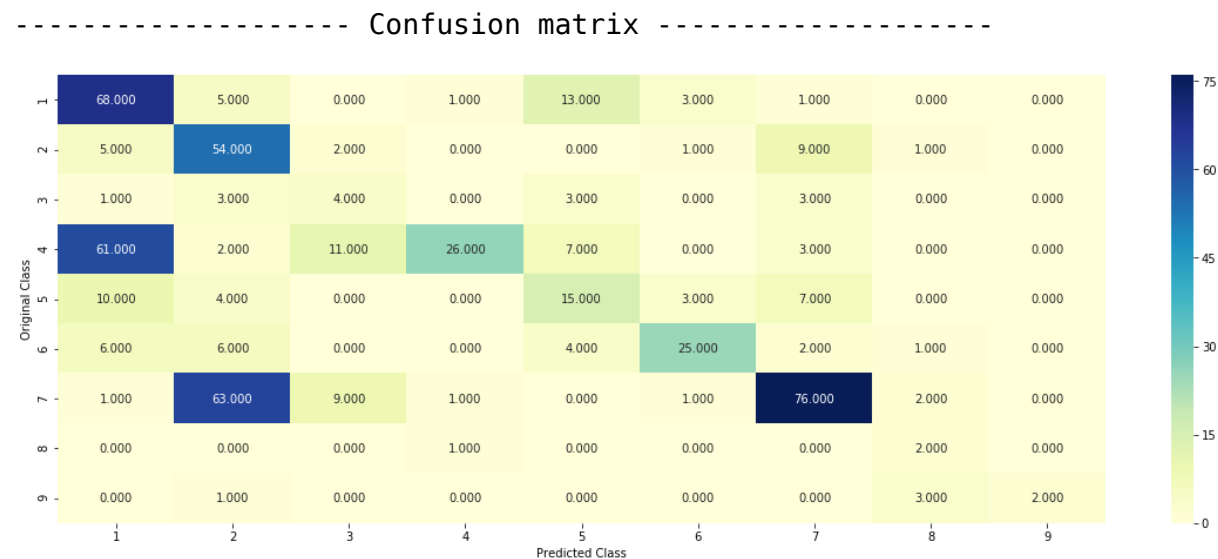
# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

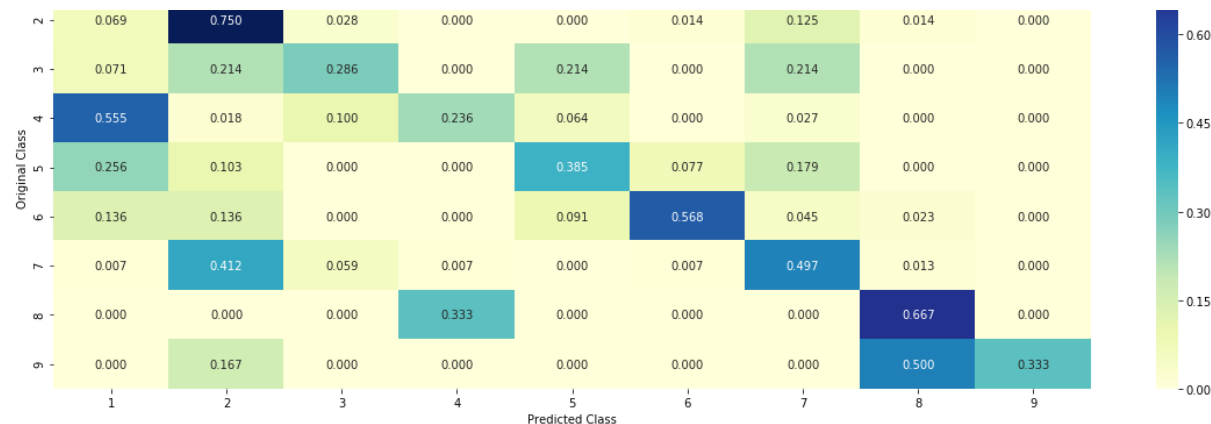
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-
online/lessons/random-forest-and-their-construction-2/
# -----

clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_
estimators=alpha[int(best_alpha/4)], criterion='gini', max_features='au
to', random_state=42)
predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_
responseCoding, cv_y, clf)
```

Log loss : 1.25058440571

Number of mis-classified points : 0.48872180451127817





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [87]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_depth=max_depth[int(best_alpha%4)], random_state=42, n_jobs=-1)
clf.fit(train_x_responseCoding, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_responseCoding, train_y)

test_point_index = 1
no_feature = 27
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
```

```
if i<9:
    print("Gene is important feature")
elif i<18:
    print("Variation is important feature")
else:
    print("Text is important feature")
```

Predicted Class : 3

Predicted Class Probabilities: [[0.0951 0.0115 0.5949 0.2397 0.0117 0.0165 0.0039 0.0152 0.0114]]

Actual Class : 4

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

4.5.5.2. Incorrectly Classified point

```
In [88]: test_point_index = 100
predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index]
.reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(
test_x_responseCoding[test_point_index].reshape(1,-1)),4))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")
```

```
Predicted Class : 3
Predicted Class Probabilities: [[ 0.0951  0.0115  0.5949  0.2397  0.011
7 0.0165  0.0039  0.0152  0.0114]]
Actual Class : 4
```

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

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

```
In [89]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/sklearn.linear\_model.SGDClassifier.html
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5,
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, ...])      Fit linear model with Stochastic Gradient Descent.
# predict(X)      Predict class labels for samples in X.

#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-intuition-1/
```

```

#-----

# read more about support vector machines with linear kernal here http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
# -----
# default parameters
# SVC(C=1.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0, shrinking
=True, probability=False, tol=0.001,
# cache_size=200, class_weight=None, verbose=False, max_iter=-1, decisi
on_function_shape='ovr', random_state=None)

# Some of methods of SVM()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathematical-derivation-copy-8/
# -----

# read more about support vector machines with linear kernal here http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomFo
restClassifier.html
# -----
# default parameters
# sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion='g
ini', max_depth=None, min_samples_split=2,
# min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='aut
o', max_leaf_nodes=None, min_impurity_decrease=0.0,
# min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, r
andom_state=None, verbose=0, warm_start=False,
# class_weight=None)

# Some of methods of RandomForestClassifier()
# fit(X, y, [sample_weight])    Fit the SVM model according to the give
n training data.
# predict(X)    Perform classification on samples in X.

```

```

# predict_proba (X)      Perform classification on samples in X.

# some of attributes of RandomForestClassifier()
# feature_importances_ : array of shape = [n_features]
# The feature importances (the higher, the more important the feature).

# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-and-their-construction-2/
# -----

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

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

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

sig_clf1.fit(train_x_onehotCoding, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_onehotCoding))))
sig_clf2.fit(train_x_onehotCoding, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_onehotCoding))))
sig_clf3.fit(train_x_onehotCoding, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_onehotCoding))))
print("-"*50)
alpha = [0.0001,0.001,0.01,0.1,1,10]
best_alpha = 999

```

```

for i in alpha:
    lr = LogisticRegression(C=i)
    sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3
], meta_classifier=lr, use_probas=True)
    sclf.fit(train_x_onehotCoding, train_y)
    print("Stacking Classifier : for the value of alpha: %f Log Loss: %
0.3f" % (i, log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))))
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
    if best_alpha > log_error:
        best_alpha = log_error

```

Logistic Regression : Log Loss: 1.47

Support vector machines : Log Loss: 1.30

Naive Bayes : Log Loss: 1.30

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.182
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.072
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.671
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.240
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.215
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.379

```

4.7.2 testing the model with the best hyper parameters

```

In [90]: lr = LogisticRegression(C=0.1)
sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], m
eta_classifier=lr, use_probas=True)
sclf.fit(train_x_onehotCoding, train_y)

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

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

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

```

```
print("Number of missclassified point :", np.count_nonzero((sclf.predict(
test_x_onehotCoding)- test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=sclf.predict(test_x_oneh
otCoding))
```

Log loss (train) on the stacking classifier : 0.919213343336

Log loss (CV) on the stacking classifier : 1.23955567173

Log loss (test) on the stacking classifier : 1.25748056

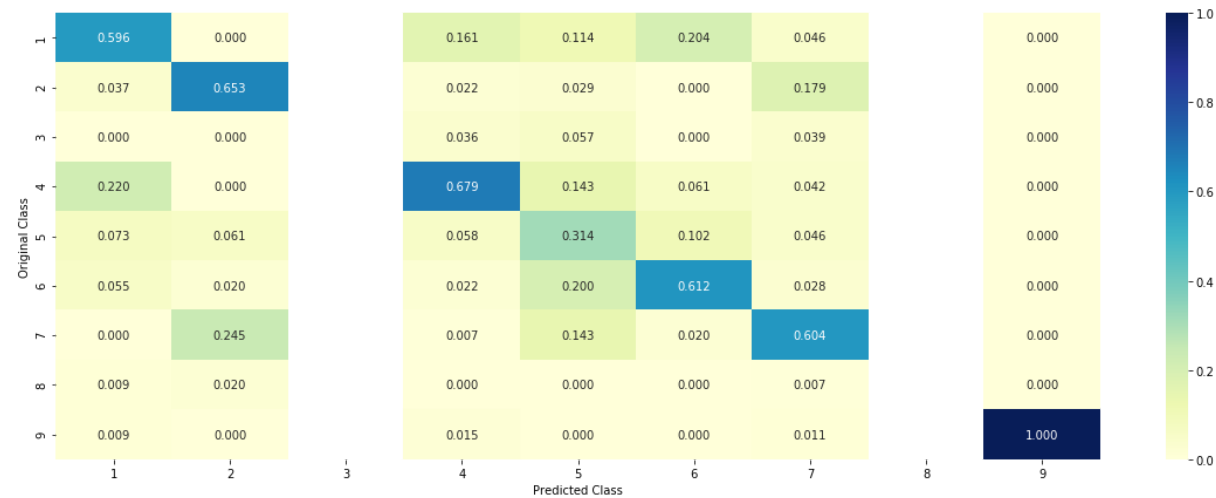
Number of missclassified point : 0.3924812030075188

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



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

--



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



4.7.3 Maximum Voting classifier

In [91]: `#Refer: http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html`
`from sklearn.ensemble import VotingClassifier`

```

vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_clf3)], voting='soft')
vclf.fit(train_x_onehotCoding, train_y)
print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_proba(train_x_onehotCoding)))
print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv_x_onehotCoding)))
print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_proba(test_x_onehotCoding)))
print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_onehotCoding) - test_y))/test_y.shape[0])
plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))

```

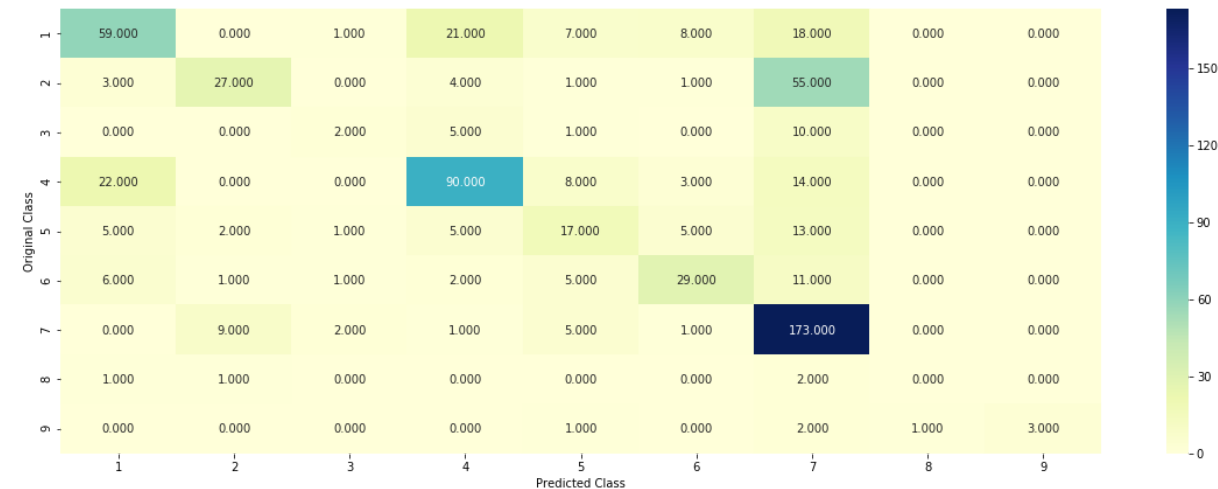
Log loss (train) on the VotingClassifier : 1.06982718995

Log loss (CV) on the VotingClassifier : 1.27393322926

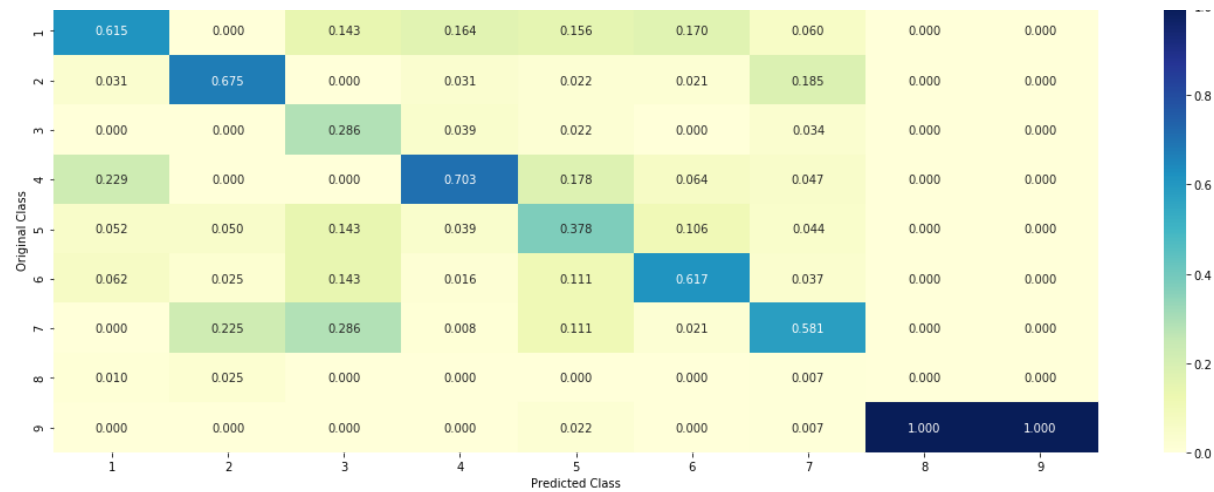
Log loss (test) on the VotingClassifier : 1.31434006665

Number of missclassified point : 0.39849624060150374

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



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



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

