

Personalized cancer diagnosis

1. Business Problem

1.1. Description

Source: <https://www.kaggle.com/c/msk-redefining-cancer-treatment/> (<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>
(<https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462>)

Problem statement :

Classify the given genetic variations/mutations based on evidence from text-based clinical literature.

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID, Text
 0|Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome. Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some

cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

Objective: Predict the probability of each data-point belonging to each of the nine classes.

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

In [0]:

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [0]:

```
data = pd.read_csv('/content/drive/My Drive/Case_Study/PersonalizedCancerDiagnosis/training_
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 [0]:

```
# note the separator in this file
data_text = pd.read_csv('/content/drive/My Drive/Case_Study/PersonalizedCancerDiagnosis/train.csv')
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 [0]:

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

# 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 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 [0]:

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

there is no text description for id: 1109
there is no text description for id: 1277
there is no text description for id: 1407
there is no text description for id: 1639
there is no text description for id: 2755
Time took for preprocessing the text: 200.951952 seconds

In [0]:

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

Out[7]:

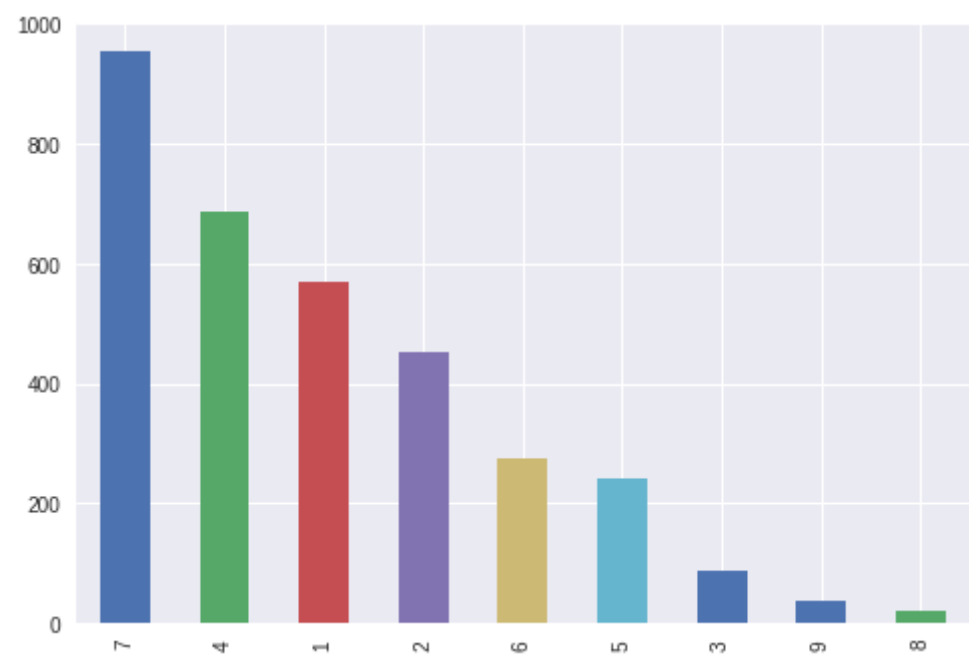
	ID	Gene	Variation	Class	TEXT
0	0	FAM58A	Truncating Mutations	1	cyclin dependent kinases cdks regulate variety...
1	1	CBL	W802*	2	abstract background non small cell lung cancer...
2	2	CBL	Q249E	2	abstract background non small cell lung cancer...
3	3	CBL	N454D	3	recent evidence demonstrated acquired uniparen...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

In [0]:

```
result['Class'].value_counts().plot(kind='bar')
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f4a57eb1390>



Classes are highly imbalanced.

In [0]:

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

Out[9]:

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

In [0]:

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

In [0]:

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

Out[11]:

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

3.1.4. Test, Train and Cross Validation Split

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

In [0]:

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

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

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

In [0]:

```
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 [0]:

```
train_df['Class'].value_counts().sortlevel()
```

Out[14]:

```
1    363
2    289
3     57
4    439
5    155
6    176
7    609
8     12
9     24
```

Name: Class, dtype: int64

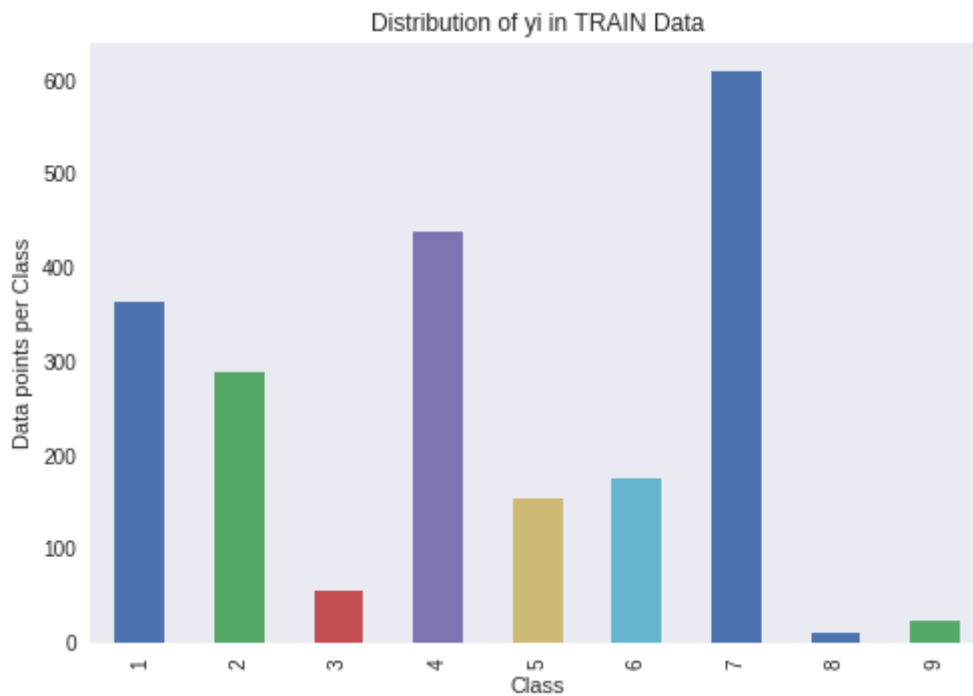
In [0]:

```
def plot_class_disb(class_distribution, data, distribution_name):  
    """  
    Method for plotting class distribution of train, test and cv data.  
    """  
  
    my_colors = 'rgbkymc'  
    class_distribution.plot(kind='bar')  
    plt.xlabel('Class')  
    plt.ylabel('Data points per Class')  
    plt.title(f'Distribution of yi in {distribution_name}')  
    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(-class_distribution.values)  
    for i in sorted_yi:  
        print('Number of data points in class', i+1, ':', class_distribution.values[i], '(', r  
  
    print('-'*80)
```

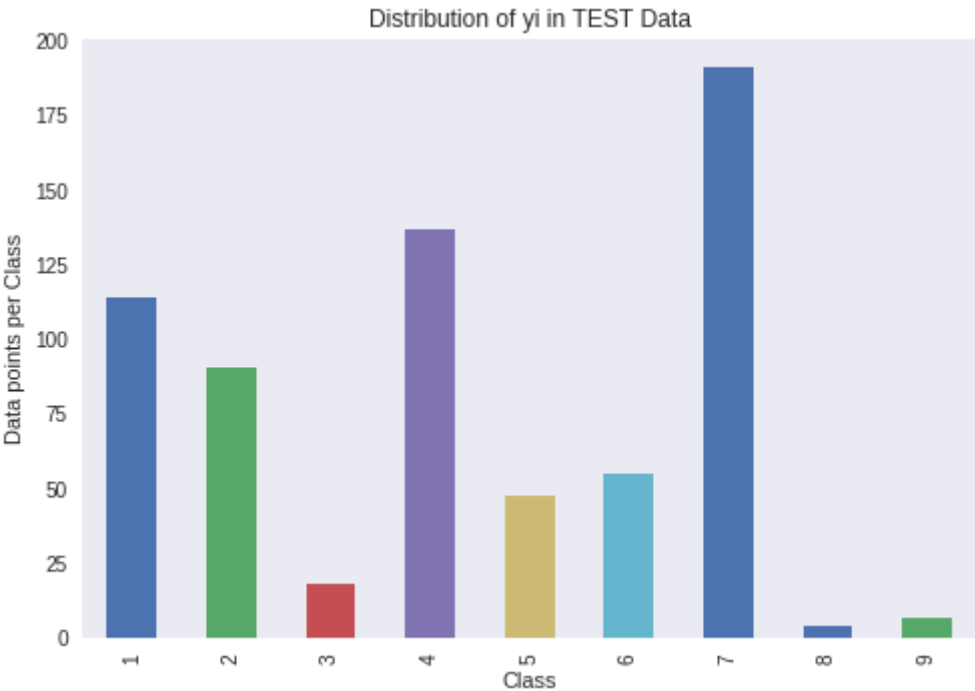
In [0]:

```
# it returns a dict, keys as class labels and values as the number of data 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()

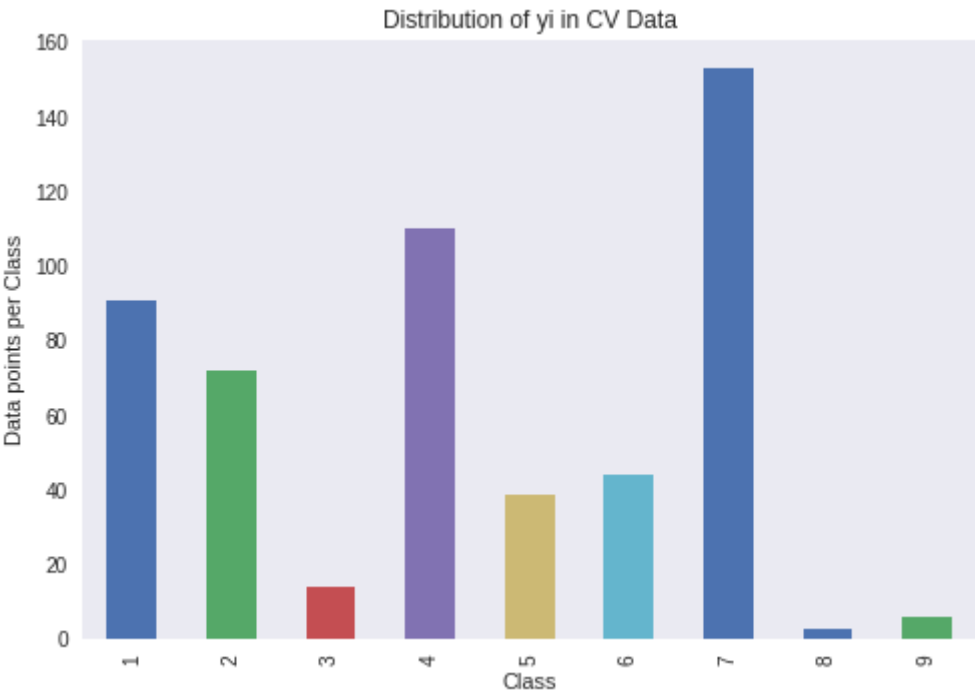
plot_class_disb(train_class_distribution, train_df, 'TRAIN Data')
plot_class_disb(test_class_distribution, test_df, 'TEST Data')
plot_class_disb(cv_class_distribution, cv_df, 'CV Data')
```



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 %)

Observations

- Train, Test and CV have roughly equal proportion of classes.

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 [0]:

```

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

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

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

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

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

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

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

```

In [0]:

```

# 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,

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs))))[0]
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-1

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 2.5468849606935215

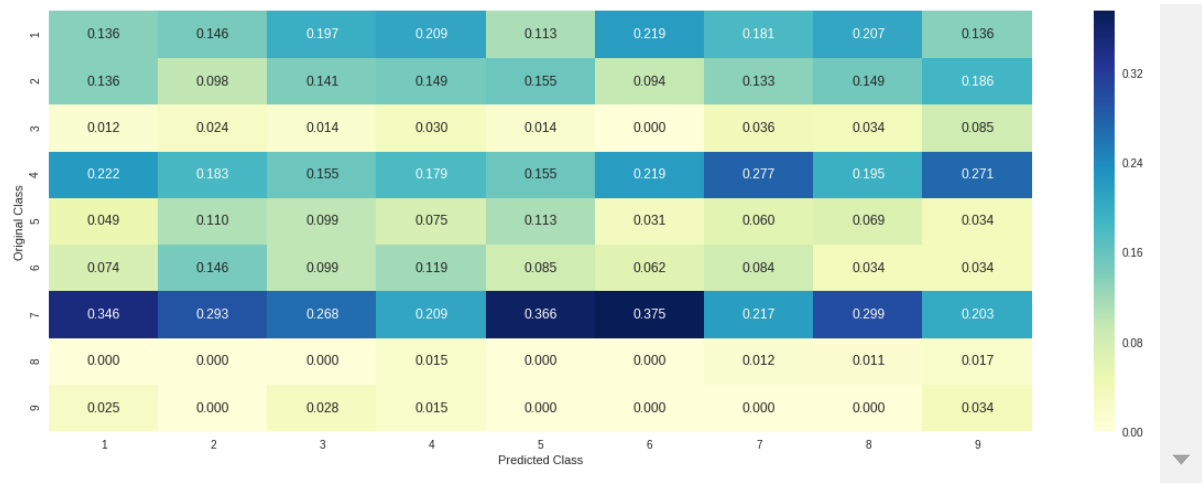
Log loss on Test Data using Random Model 2.4967379141693704

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

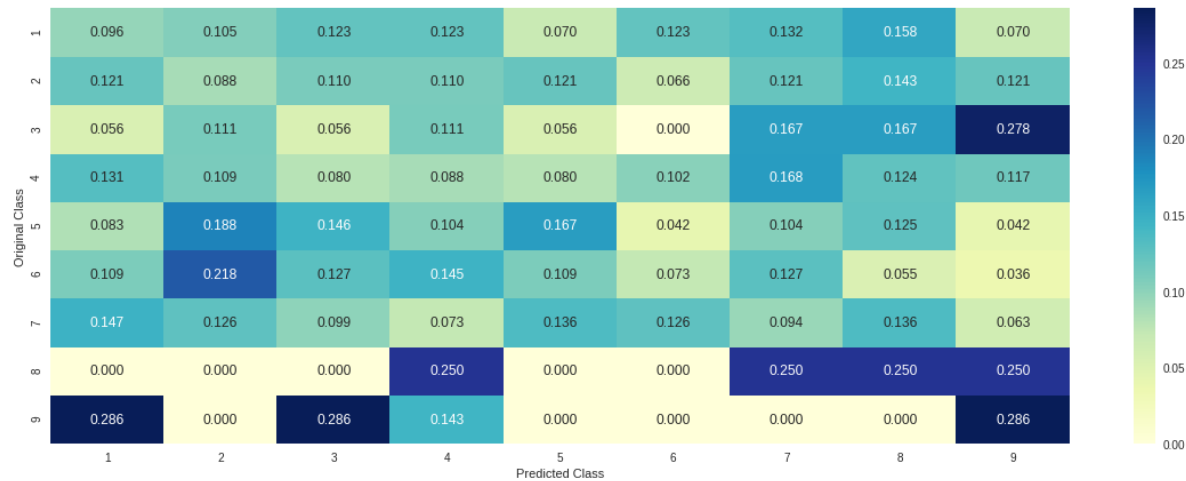


----- Precision matrix (Column Sum=1) -----





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



3.3 Univariate Analysis

In [0]:

```

# 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
# build a vector (1*9) , the first element = (number of times it occurred in class1 + 10*alp
# gv_dict is like a look up table, for every gene it store a (1*9) representation of it
# for a value of feature in df:
# if it is in train data:
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# -----

# get_gv_fea_dict: Get Gene 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/var
    gv_dict = dict()

    # denominator will contain the number of time that particular feature occurred in whole
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to particular
        # 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 time that particular f
vec.append((cls_cnt.shape[0] + alpha*10)/ (denominator + 90*alpha))

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

# Get Gene variation feature
def get_gv_feature(alpha, feature, df):
    # print(gv_dict)
    # {'BRCA1': [0.20075757575757575, 0.03787878787878788, 0.068181818181818177, 0.1363
    # 'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 0.2704
    # 'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.068181818181818177
    # 'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 0.078
    # 'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 0.465
    # 'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0.07284
    # 'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 0.0733
    # ...
    # }
    gv_dict = get_gv_fea_dict(alpha, feature, df)
    # value_count is similar in get_gv_fea_dict
    value_count = train_df[feature].value_counts()

    # gv_fea: Gene_variation feature, it will contain the feature for each feature value in
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there in the
    # if not we will add [1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9] to gv_fea
    for index, row in df.iterrows():
        if row[feature] in dict(value_count).keys():
            gv_fea.append(gv_dict[row[feature]])
        else:
            gv_fea.append([1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9,1/9])
    # gv_fea.append([-1,-1,-1,-1,-1,-1,-1,-1,-1])
    return gv_fea

```

when we caculate the probability of a feature belongs to any particular class, we apply laplace 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 [0]:

```
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 : 235

BRCA1	171
TP53	105
EGFR	93
BRCA2	83
PTEN	80
BRAF	67
KIT	61
ERBB2	45
ALK	45
PDGFRA	40

Name: Gene, dtype: int64

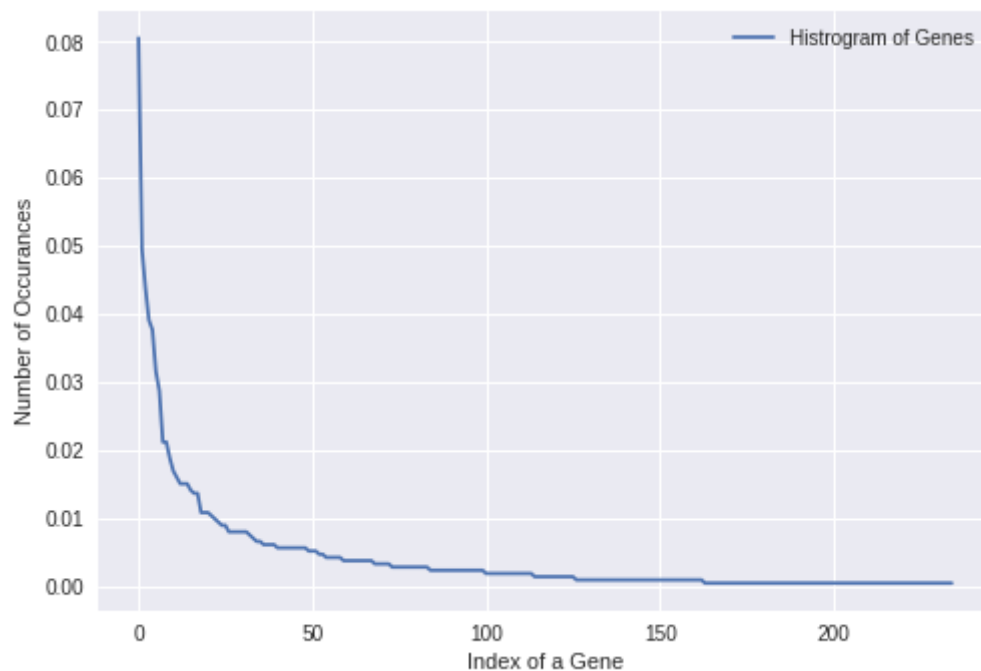
In [0]:

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

Ans: There are 235 different categories of genes in the train data, and they are distributed as follows

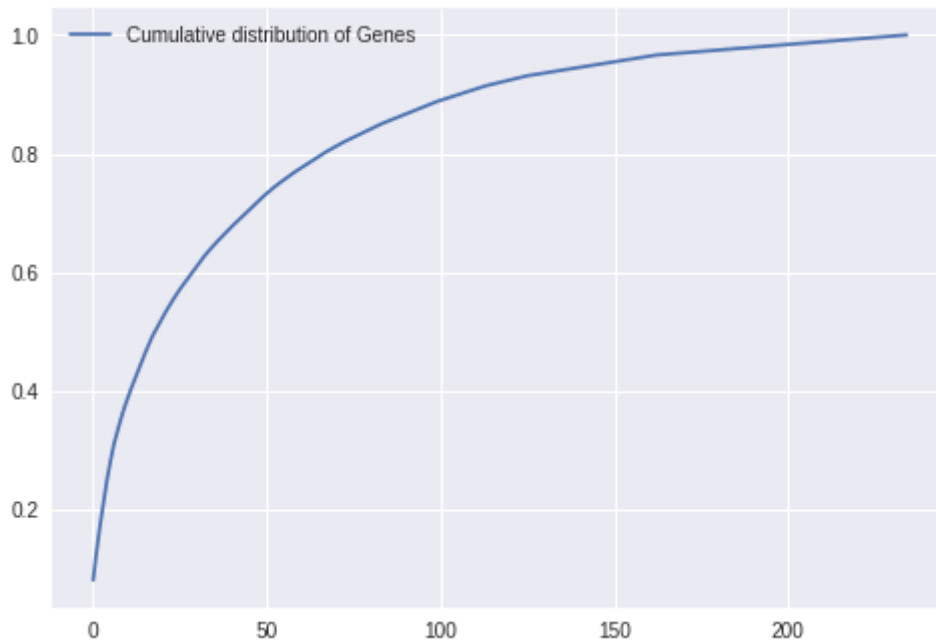
In [0]:

```
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(True)  
plt.show()
```



In [0]:

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



Observations

- There are 228 different categories of Genes, and most of the Genes are unique.

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/> (<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 [0]:

```
#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 [0]:

```
print("train_gene_feature_responseCoding is converted feature using response coding method.
```

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

In [0]:

```
# one-hot encoding of Gene feature.
gene_vectorizer = CountVectorizer()
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 [0]:

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

Out[30]:

```
1988    CTNNB1
2413    PTPRD
101     TGFB2
1122     MET
744     ERBB2
Name: Gene, dtype: object
```

In [0]:

```
gene_vectorizer.get_feature_names()
```

Out[31]:

```
['abl1',  
'acvr1',  
'ago2',  
'akt1',  
'akt2',  
'akt3',  
'alk',  
'apc',  
'ar',  
'araf',  
'arid1a',  
'arid1b',  
'arid5b',  
'asx11',  
'asx12',  
'atm',  
'atr',  
'aurka']
```

In [0]:

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

```
train_gene_feature_onehotCoding is converted feature using one-hot encoding  
method. The shape of gene feature: (2124, 234)
```

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 [0]:

```

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
# -----
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# class_weight=None, warm_start=False, average=False, n_iter=None)

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

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

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

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

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

predict_y = sig_clf.predict_proba(train_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_gene_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

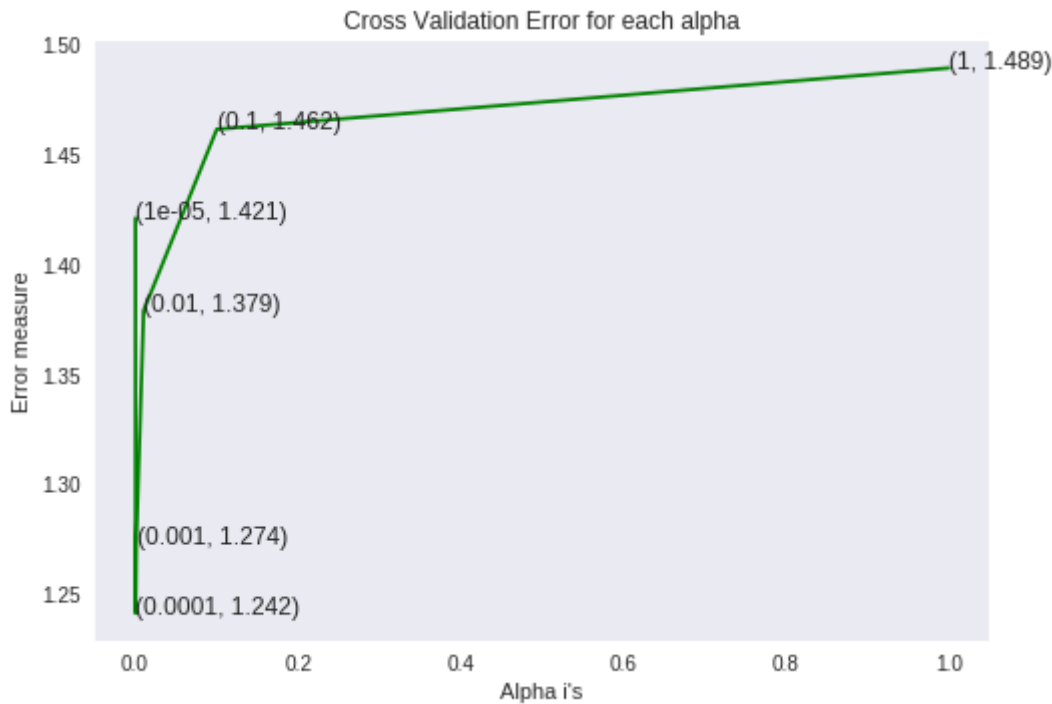
```

For values of alpha = 1e-05 The log loss is: 1.4212527439577995
For values of alpha = 0.0001 The log loss is: 1.241522912383149
For values of alpha = 0.001 The log loss is: 1.2735153537186683
For values of alpha = 0.01 The log loss is: 1.379118446518656

```


For values of alpha = 0.1 The log loss is: 1.4616569301224607

For values of alpha = 1 The log loss is: 1.4894965496860753



For values of best alpha = 0.0001 The train log loss is: 1.038169821062594

For values of best alpha = 0.0001 The cross validation log loss is: 1.241522912383149

For values of best alpha = 0.0001 The test log loss is: 1.1934556154096183

Observation

- Log loss is less than the loss of a random model and hence gene feature is useful in predicting class label.

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 [0]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.
test_coverage=test_df[test_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]
cv_coverage=cv_df[cv_df['Gene'].isin(list(set(train_df['Gene'])))].shape[0]

print('Ans\n1. In test data',test_coverage, 'out of',test_df.shape[0], ":",(test_coverage/t
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage
```

Q6. How many data points in Test and CV datasets are covered by the 235 genes in train dataset?

Ans

1. In test data 643 out of 665 : 96.69172932330827

2. In cross validation data 516 out of 532 : 96.99248120300751

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 [0]:

```
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 : 1933

Deletion 53

Truncating_Mutations 51

Amplification 48

Fusions 19

Q61H 3

G12C 2

S222D 2

S308A 2

A146V 2

T167A 2

Name: Variation, dtype: int64

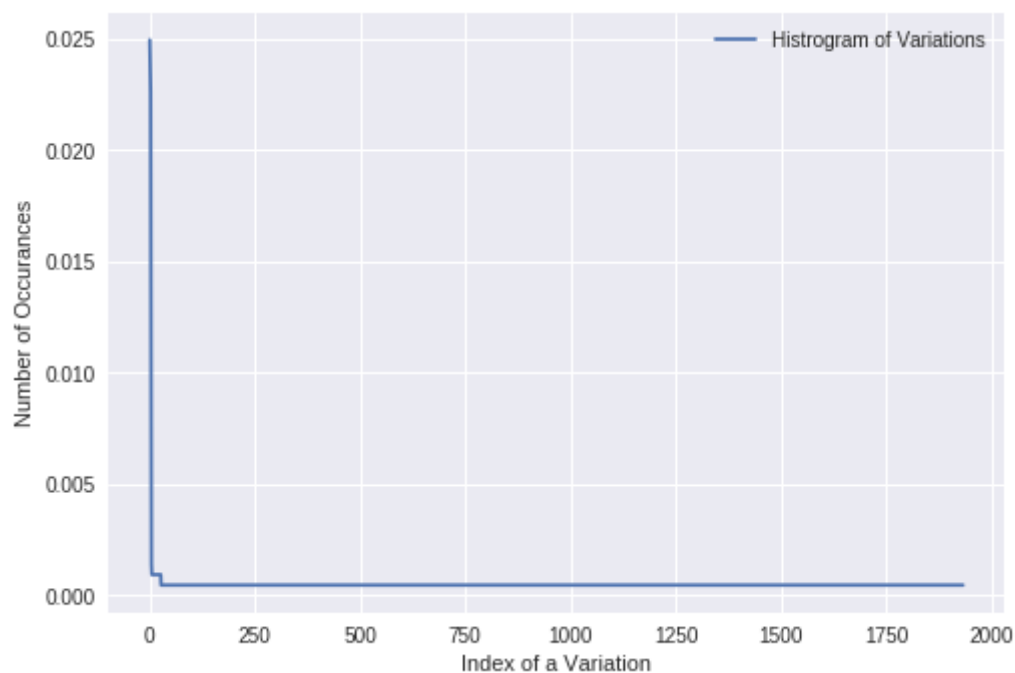
In [0]:

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

Ans: There are 1933 different categories of variations in the train data, and they are distributed as follows

In [0]:

```
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(True)  
plt.show()
```



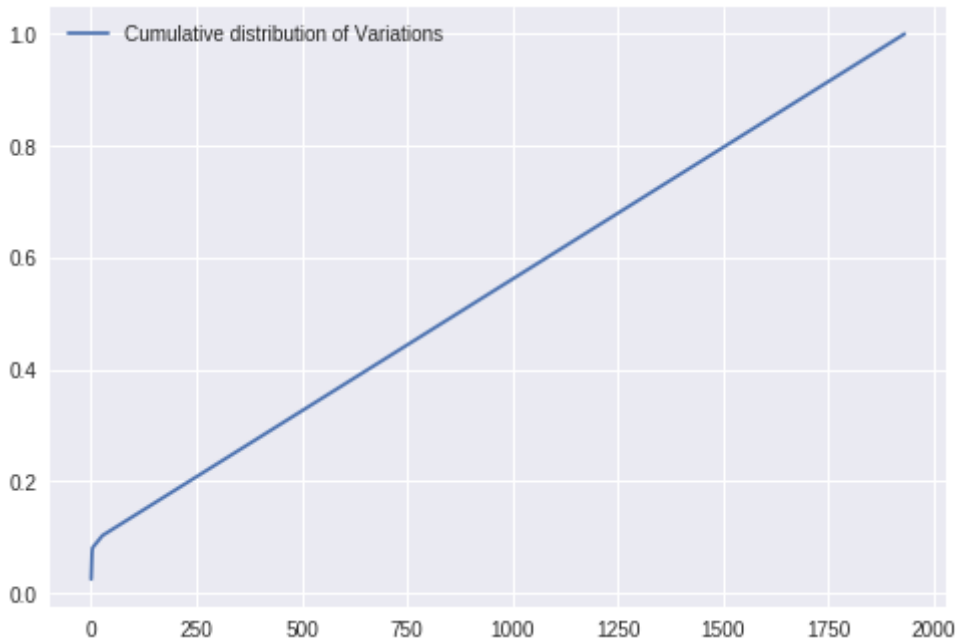
In [0]:

```

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

```

```
[0.02495292 0.04896422 0.07156309 ... 0.99905838 0.99952919 1.      ]
```



Observations

- There are 1927 different categories of Variation and majority of them are unique.

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/> (<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 [0]:

```

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

```

In [0]:

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

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

In [0]:

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

In [0]:

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

train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 1959)

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

Let's build a model just like the earlier!

In [0]:

```

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

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

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

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

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

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

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

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

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

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

```

```

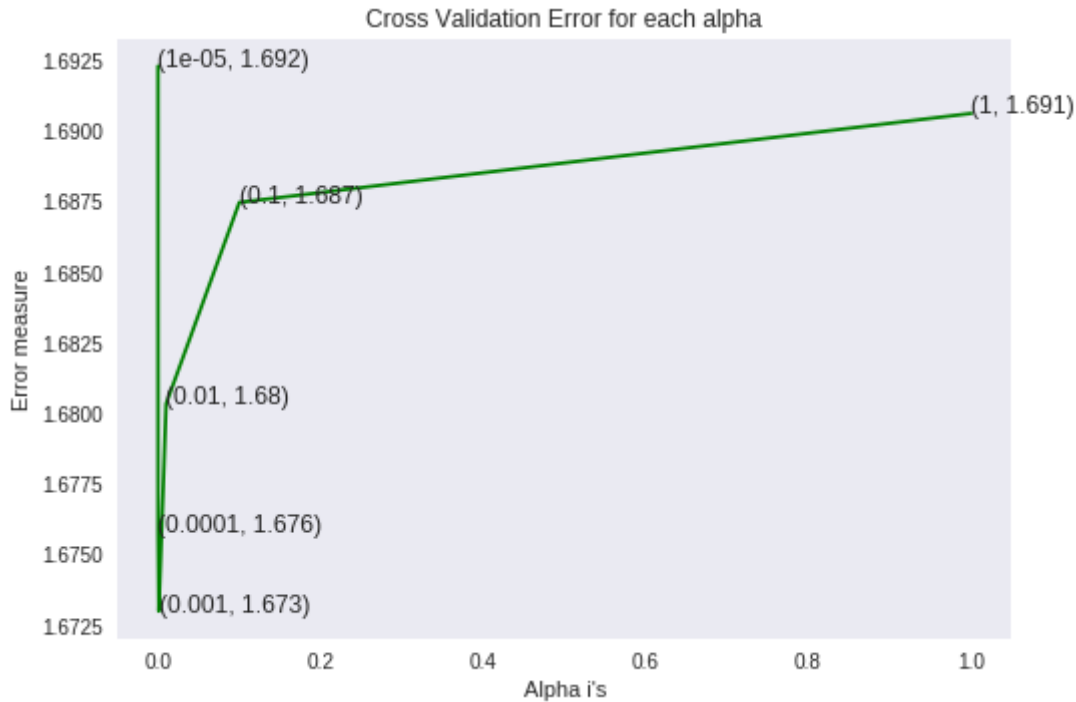
For values of alpha = 1e-05 The log loss is: 1.6923051953210095
For values of alpha = 0.0001 The log loss is: 1.6757733051452106
For values of alpha = 0.001 The log loss is: 1.6729942743392452

```

For values of alpha = 0.01 The log loss is: 1.6803411758730864

For values of alpha = 0.1 The log loss is: 1.6874720483180088

For values of alpha = 1 The log loss is: 1.6906262344000382



For values of best alpha = 0.001 The train log loss is: 1.1038537257899532

For values of best alpha = 0.001 The cross validation log loss is: 1.6729942743392452

For values of best alpha = 0.001 The test log loss is: 1.714414035373943

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 [0]:

```
print("Q12. How many data points are covered by total ", unique_variations.shape[0], " genes in test and cross validation data sets?", unique_variations.shape[0])
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]))
print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":", (cv_coverage/cv_df.shape[0]))
```

Q12. How many data points are covered by total 1933 genes in test and cross validation data sets?

Ans

1. In test data 61 out of 665 : 9.172932330827068

2. In cross validation data 68 out of 532 : 12.781954887218044

Observations

- Variation feature is not stable across all the data sets.

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 [0]:

```
# 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 [0]:

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

In [0]:

```
# building a CountVectorizer with all the words that occurred minimum 3 times in train data
text_vectorizer = CountVectorizer(min_df=3)
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 words)
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 : 53742

In [0]:

```
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 [0]:

```
#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 [0]:

```
# 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.T).T
test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_feature_responseCoding.T).T
cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_responseCoding.T).T
```

In [0]:

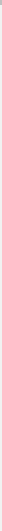
```
# don't forget to normalize every feature
train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

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

```
# Number of words for a given frequency.
print(Counter(sorted text occur))
```



In [0]:

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

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

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

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

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

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

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

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

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

```
For values of alpha = 1e-05 The log loss is: 1.422172833756881
For values of alpha = 0.0001 The log loss is: 1.4080637454795875
For values of alpha = 0.001 The log loss is: 1.3011255107960231
```

For values of alpha = 0.01 The log loss is: 1.3610436499592808

For values of alpha = 0.1 The log loss is: 1.5011338738459512

For values of alpha = 1 The log loss is: 1.6600904466200095



For values of best alpha = 0.001 The train log loss is: 0.7445756986108502

For values of best alpha = 0.001 The cross validation log loss is: 1.3011255107960231

For values of best alpha = 0.001 The test log loss is: 1.203896897643536

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

Ans. Yes, it seems like!

In [0]:

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

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

In [0]:

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

97.322 % of word of test data appeared in train data

98.333 % of word of Cross Validation appeared in train data

4. Machine Learning Models

In [0]:

```
#Data preparation for ML models.

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

In [0]:

```
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 [0]:

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

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

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

    word_present = 0
    for i,v in enumerate(indices):
        if (v < fea1_len):
            word = gene_vec.get_feature_names()[v]
            yes_no = True if word == gene else False
            if yes_no:
                word_present += 1
                print(i, "Gene feature [{}]" .format(word,ye
        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 [{}]" .format(wc
        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 [{}]" .format(word,ye

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

Stacking the three types of features

In [0]:

```
# merging gene, variance and text features

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

train_gene_var_onehotCoding = hstack((train_gene_feature_onehotCoding, train_variation_feature_onehotCoding))
test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding, test_variation_feature_onehotCoding))
cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding, cv_variation_feature_onehotCoding))

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

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

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

train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding, train_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 [0]:

```
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, 55935)

(number of data points * number of features) in test data = (665, 55935)

(number of data points * number of features) in cross validation data = (53

2, 55935)

In [0]:

```
print(" Response encoding features :")
print("(number of data points * number of features) in train data = ", train_x_responseCodi
print("(number of data points * number of features) in test data = ", test_x_responseCoding
print("(number of data points * number of features) in cross validation data =", cv_x_respc
```

```
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 = (53
2, 27)
```

Text: TFIDF encoded

In [0]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf = TfidfVectorizer(min_df=3, max_features=1000) # top 1000 words.

train_text_tfidf = tfidf.fit_transform(train_df['TEXT'])
test_text_tfidf = tfidf.transform(test_df['TEXT'])
cv_text_tfidf = tfidf.transform(cv_df['TEXT'])

train_text_features_tfidf= tfidf.get_feature_names()
```

In [0]:

```
train_text_tfidf.shape
```

Out[68]:

```
(2124, 1000)
```


In [0]:

```
# DataMatrix

train_x_tfidf = hstack((
    train_gene_feature_onehotCoding,
    train_variation_feature_onehotCoding,
    train_text_tfidf
)).tocsr()

test_x_tfidf = hstack((
    test_gene_feature_onehotCoding,
    test_variation_feature_onehotCoding,
    test_text_tfidf
)).tocsr()

cv_x_tfidf = hstack((
    cv_gene_feature_onehotCoding,
    cv_variation_feature_onehotCoding,
    cv_text_tfidf
)).tocsr()

print("One hot encoding features(Gene, Variation) and TFIDF(Text) :")
print("(number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_tfidf
```

```
One hot encoding features(Gene, Variation) and TFIDF(Text) :
(number of data points * number of features) in train data = (2124, 3193)
(number of data points * number of features) in test data = (665, 3193)
(number of data points * number of features) in cross validation data = (53
2, 3193)
```

4.1. Base Line Model

4.1.1. Naive Bayes(Text TFIDF encoded)

4.1.1.1. Hyper parameter tuning

In [0]:

```

# 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
# -----

# 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
# -----

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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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(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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

```

```

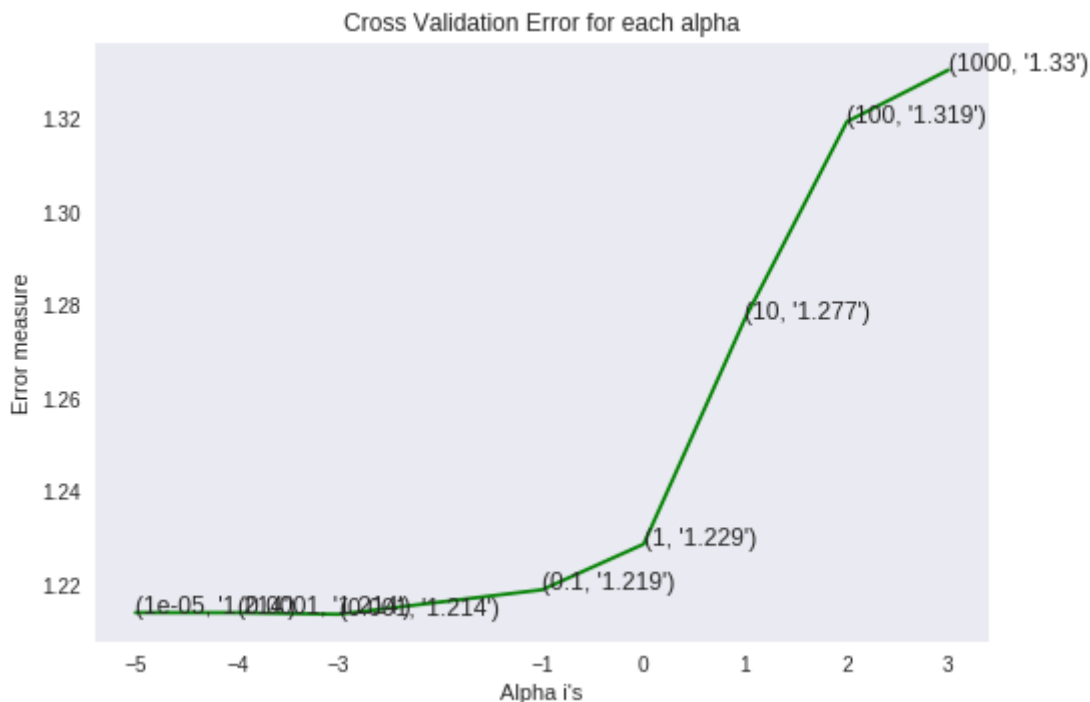
predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 1e-05
Log Loss : 1.2140753532020883
for alpha = 0.0001
Log Loss : 1.2140627966571604
for alpha = 0.001
Log Loss : 1.213730537688673
for alpha = 0.1
Log Loss : 1.2189755279134538
for alpha = 1
Log Loss : 1.2287600081711512
for alpha = 10
Log Loss : 1.2771937630148824
for alpha = 100
Log Loss : 1.3192679914201446
for alpha = 1000
Log Loss : 1.3303515255958058

```



```

For values of best alpha = 0.001 The train log loss is: 0.5640646097413443
For values of best alpha = 0.001 The cross validation log loss is: 1.213730
537688673
For values of best alpha = 0.001 The test log loss is: 1.2010434053780354

```

4.1.1.2. Testing the model with best hyper paramters

In [0]:

```

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

```

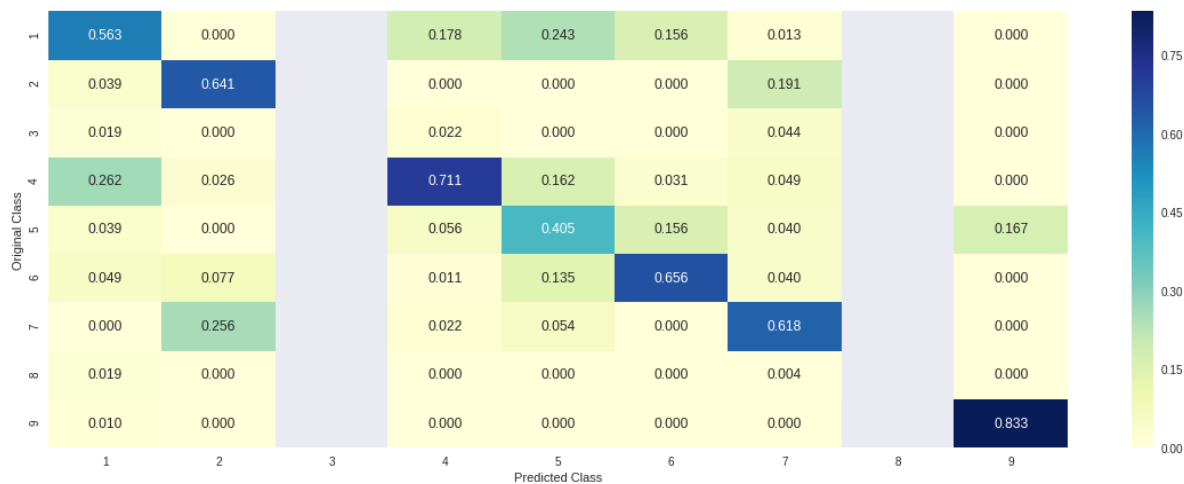
Log Loss : 1.213730537688673

Number of missclassified point : 0.38533834586466165

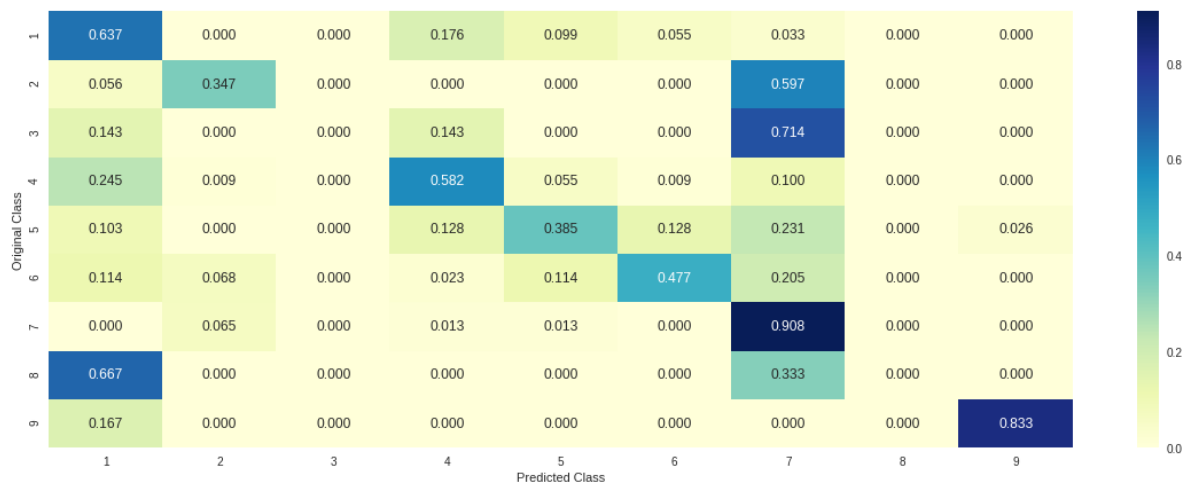
----- Confusion matrix -----



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



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



4.1.1.3. Feature Importance, Correctly classified point

In [0]:

```
test_point_index = 1
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]), 5))
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])
```

Predicted Class : 4

Predicted Class Probabilities: [[0.2047 0.0588 0.0105 0.5337 0.0399 0.0334
0.1107 0.0042 0.0041]]

Actual Class : 4

```
-----
29 Text feature [036] present in test data point [True]
54 Text feature [127k] present in test data point [True]
96 Text feature [01] present in test data point [True]
Out of the top 100 features 3 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

In [0]:

```

test_point_index = 50
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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])

```

```

Predicted Class : 6
Predicted Class Probabilities: [[0.0645 0.0485 0.0095 0.0707 0.057 0.6364
0.1055 0.004 0.0039]]
Actual Class : 1
-----
16 Text feature [04] present in test data point [True]
43 Text feature [013] present in test data point [True]
57 Text feature [049] present in test data point [True]
70 Text feature [123] present in test data point [True]
84 Text feature [110] present in test data point [True]
98 Text feature [108table] present in test data point [True]
Out of the top 100 features 6 are present in query point

```

4.2. K Nearest Neighbour Classification(Text TFIDF encoded)

4.2.1. Hyper parameter tuning

In [0]:

```

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/gen
# -----
# 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_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    # to avoid rounding error while multiplying probabilites we use log-probability estimat
    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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

```

```

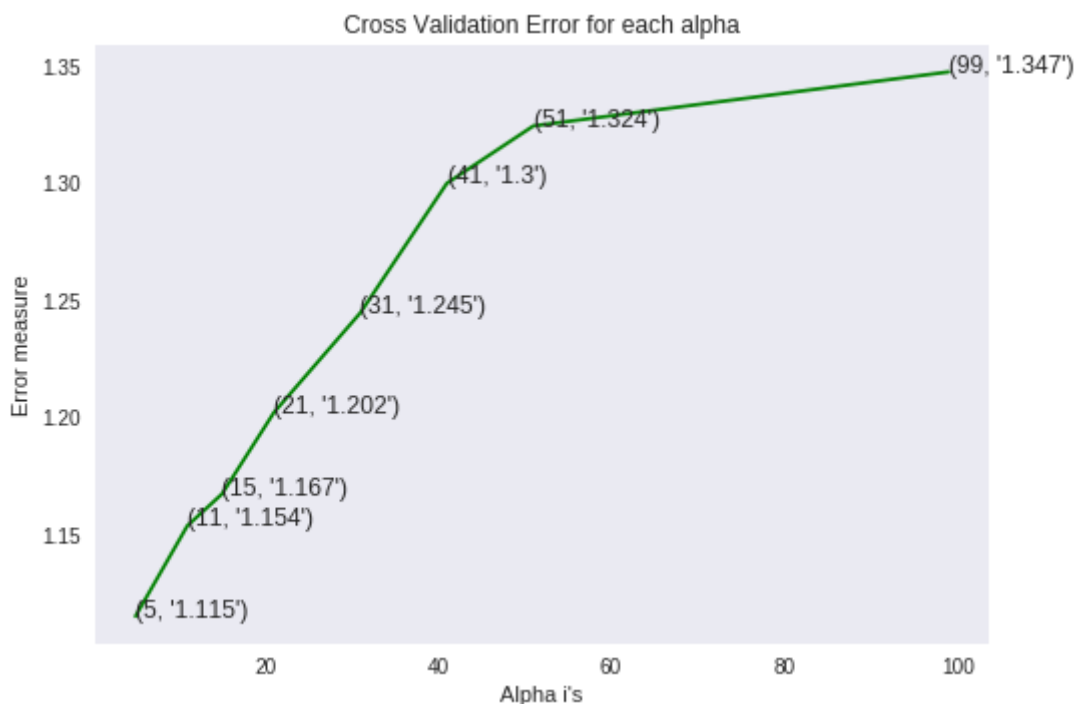
predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 5
Log Loss : 1.115122789015937
for alpha = 11
Log Loss : 1.153758058881782
for alpha = 15
Log Loss : 1.1672991320785975
for alpha = 21
Log Loss : 1.2020383658188596
for alpha = 31
Log Loss : 1.2446107591233986
for alpha = 41
Log Loss : 1.2998910950990434
for alpha = 51
Log Loss : 1.3243595207226775
for alpha = 99
Log Loss : 1.3474003938028911

```



```

For values of best alpha = 5 The train log loss is: 0.8831987266074923
For values of best alpha = 5 The cross validation log loss is: 1.1151227890
15937
For values of best alpha = 5 The test log loss is: 1.126170451989203

```

4.2.2. Testing the model with best hyper paramters

In [0]:

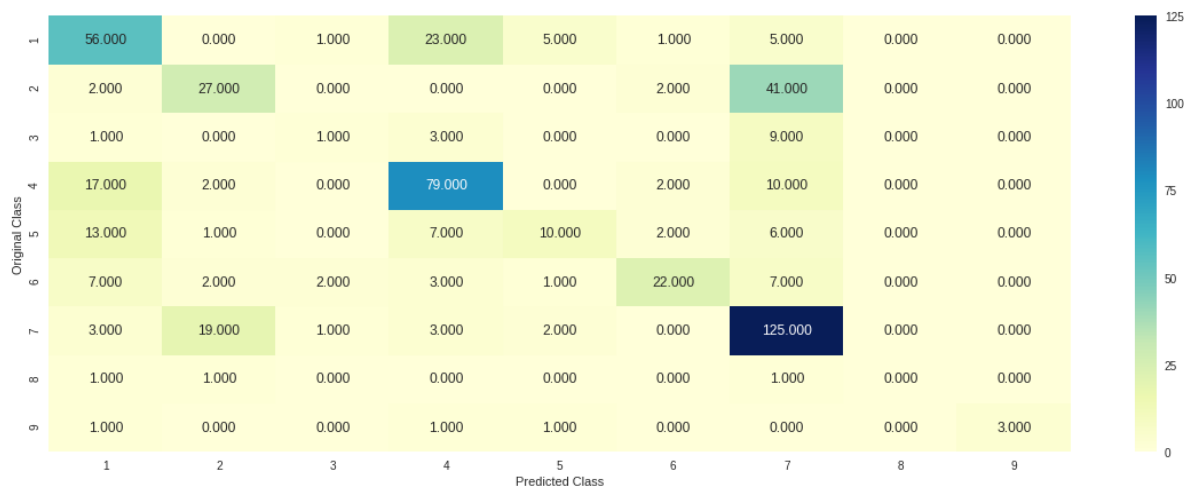
```
# find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/genera
# -----
# default parameter
# KNeighborsClassifier(n_neighbors=5, weights='uniform', algorithm='auto', leaf_size=30, p=
# metric='minkowski', metric_params=None, n_jobs=1, **kwargs)

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

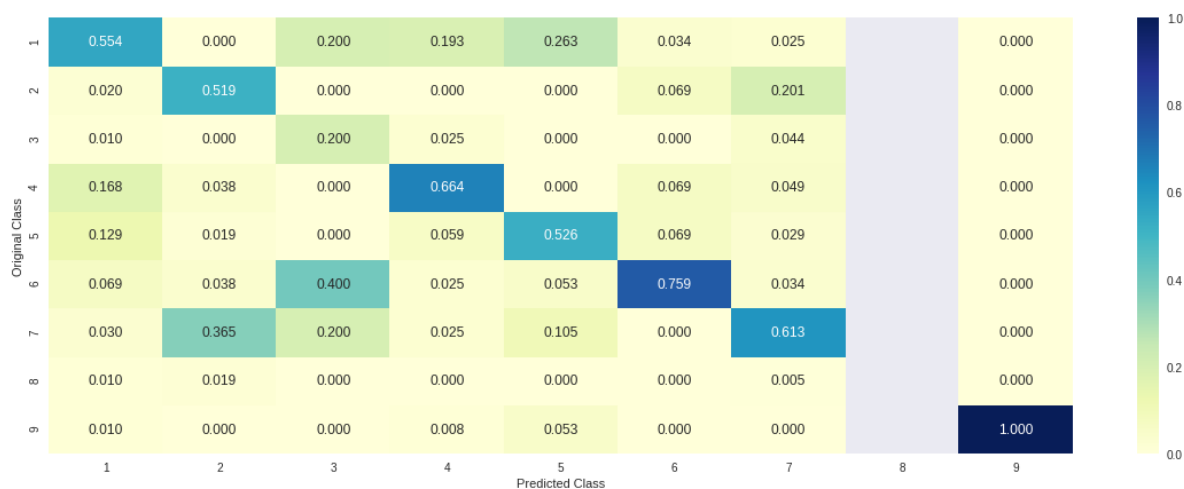
Log loss : 1.115122789015937

Number of mis-classified points : 0.39285714285714285

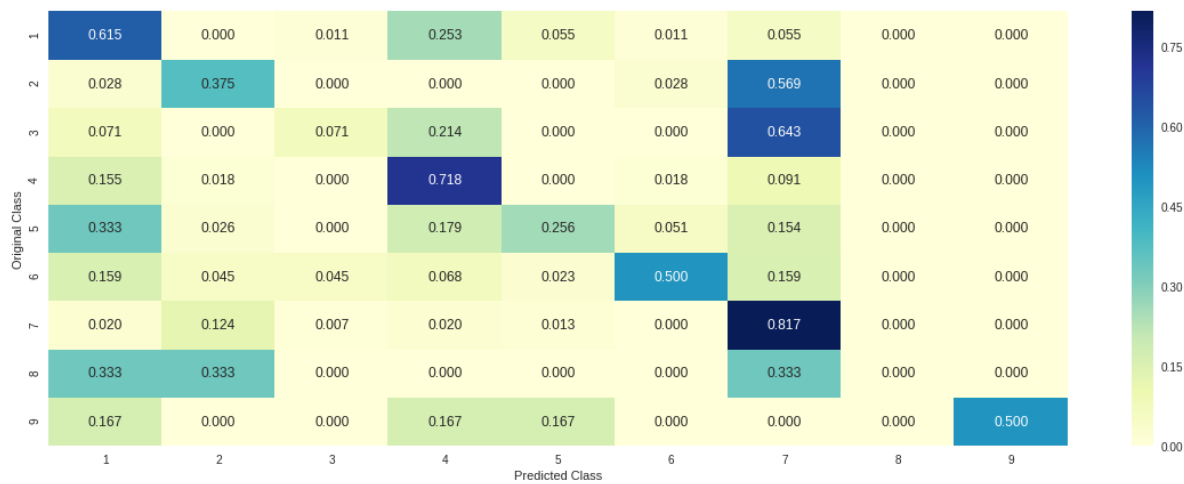
----- Confusion matrix -----



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



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



4.2.3. Sample Query point -1

In [0]:

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

test_point_index = 10
predicted_cls = sig_clf.predict(test_x_tfidf[0].reshape(1,-1))
print("Predicted Class :", predicted_cls[0])
print("Actual Class :", test_y[test_point_index])
neighbors = clf.kneighbors(test_x_tfidf[test_point_index].reshape(1, -1), alpha[best_alpha])
print("The ", alpha[best_alpha], " nearest neighbours of the test points belongs to classes",
print("Frequency of nearest points :", Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 7

Actual Class : 7

The 5 nearest neighbours of the test points belongs to classes [7 7 7 7 7]

Frequency of nearest points : Counter({7: 5})

4.2.4. Sample Query Point-2

In [0]:

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

test_point_index = 100

predicted_cls = sig_clf.predict(test_x_tfidf[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_tfidf[test_point_index].reshape(1, -1), alpha[best_alpha])
print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the test po")
print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

Predicted Class : 4

Actual Class : 1

the k value for knn is 5 and the nearest neighbours of the test points belong to classes [1 3 4 1 4]

Fequency of nearest points : Counter({1: 2, 4: 2, 3: 1})

4.3. Logistic Regression(Text TFIDF encoded)

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [0]:

```

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn
# -----
# default parameters
# 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=None)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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 estimator
    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(True)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=None)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```

```

sig_clf.fit(train_x_tfidf, train_y)

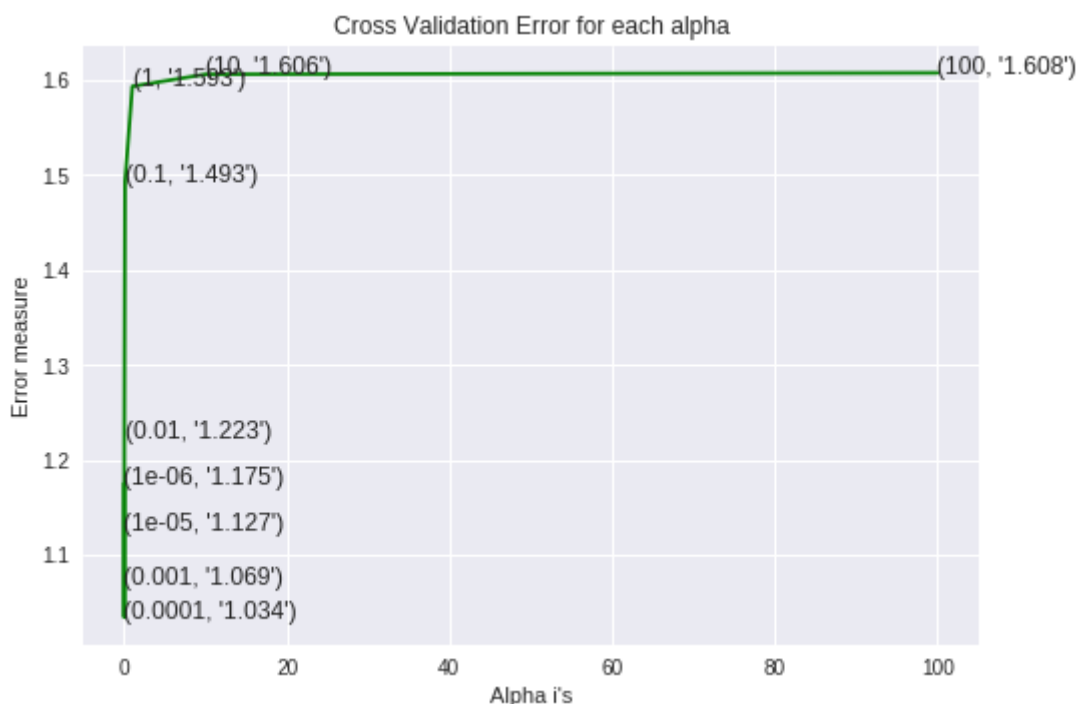
predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 1e-06
Log Loss : 1.1753899547792381
for alpha = 1e-05
Log Loss : 1.1273217115832748
for alpha = 0.0001
Log Loss : 1.0343497975159646
for alpha = 0.001
Log Loss : 1.0693022355903392
for alpha = 0.01
Log Loss : 1.223020743340451
for alpha = 0.1
Log Loss : 1.4932986784639242
for alpha = 1
Log Loss : 1.5933837675516143
for alpha = 10
Log Loss : 1.6060519480450672
for alpha = 100
Log Loss : 1.607505239056134

```



```

For values of best alpha = 0.0001 The train log loss is: 0.45739684150698
For values of best alpha = 0.0001 The cross validation log loss is: 1.03434
97975159646
For values of best alpha = 0.0001 The test log loss is: 1.0185449501528434

```

4.3.1.2. Testing the model with best hyper paramters

In [0]:

```
# 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,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
# 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-interpretation-of-linear-classifiers-in-svm/
#-----

clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge',
                    shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal',
                    class_weight=None, warm_start=False, average=False, n_iter=None)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

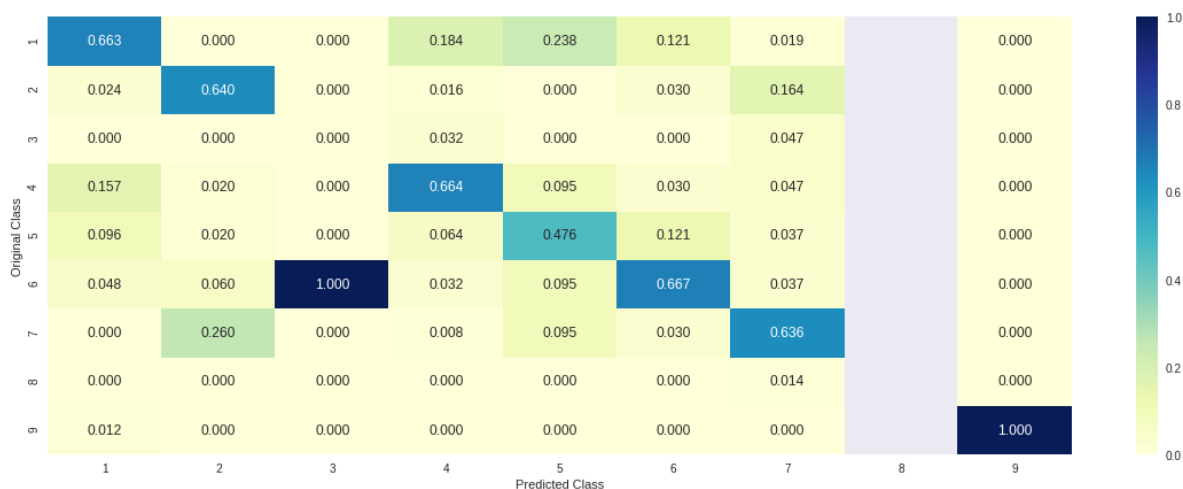
Log loss : 1.0343497975159646

Number of mis-classified points : 0.35526315789473684

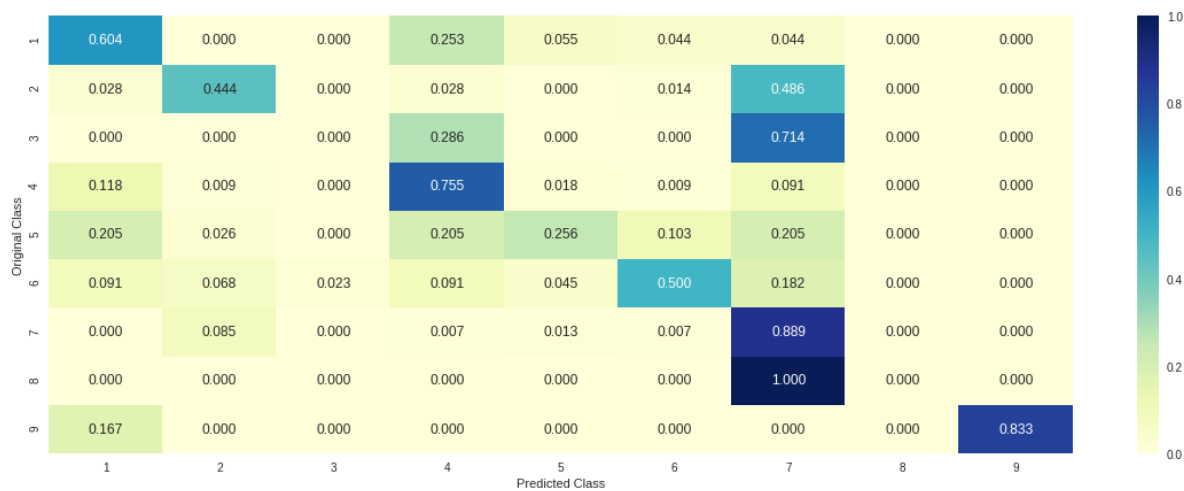
----- Confusion matrix -----



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



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



4.3.1.3. Feature Importance

In [0]:

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

4.3.1.3.1. Correctly Classified point

In [0]:

```
# from tabulate import tabulate
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l2')
clf.fit(train_x_tfidf, train_y)
test_point_index = 1
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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])
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1718 0.08    0.0055 0.5762 0.0814 0.0062
0.0706 0.006  0.0022]]
Actual Class : 4
-----
235 Text feature [01] present in test data point [True]
488 Text feature [12] present in test data point [True]
Out of the top 500 features 2 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

In [0]:

```
test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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])
```

```
Predicted Class : 3
Predicted Class Probabilities: [[0.223  0.0742 0.381  0.1637 0.0287 0.0298
0.0891 0.0044 0.0062]]
Actual Class : 1
-----
98 Text feature [1155] present in test data point [True]
119 Text feature [110] present in test data point [True]
135 Text feature [10] present in test data point [True]
137 Text feature [02] present in test data point [True]
152 Text feature [112] present in test data point [True]
166 Text feature [116] present in test data point [True]
167 Text feature [117] present in test data point [True]
215 Text feature [1108] present in test data point [True]
257 Text feature [1000] present in test data point [True]
280 Text feature [1238] present in test data point [True]
281 Text feature [124] present in test data point [True]
Out of the top 500 features 11 are present in query point
```


4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

In [0]:

```

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

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

alpha = [10 ** x for x in range(-6, 1)]
cv_log_error_array = []
for i in alpha:
    print("for alpha =", i)
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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(True)
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_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, train_y)

```

```

predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 1e-06
Log Loss : 1.1759948865036889
for alpha = 1e-05
Log Loss : 1.1372625134005867
for alpha = 0.0001
Log Loss : 1.0405876140132555
for alpha = 0.001
Log Loss : 1.0785288893787006
for alpha = 0.01
Log Loss : 1.2205471971473159
for alpha = 0.1
Log Loss : 1.4339755717446563
for alpha = 1
Log Loss : 1.5326108273000258

```



For values of best alpha = 0.0001 The train log loss is: 0.44720830344284335

For values of best alpha = 0.0001 The cross validation log loss is: 1.0405876140132555

For values of best alpha = 0.0001 The test log loss is: 1.0194371464945986

4.3.2.2. Testing model with best hyper parameters

In [0]:

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

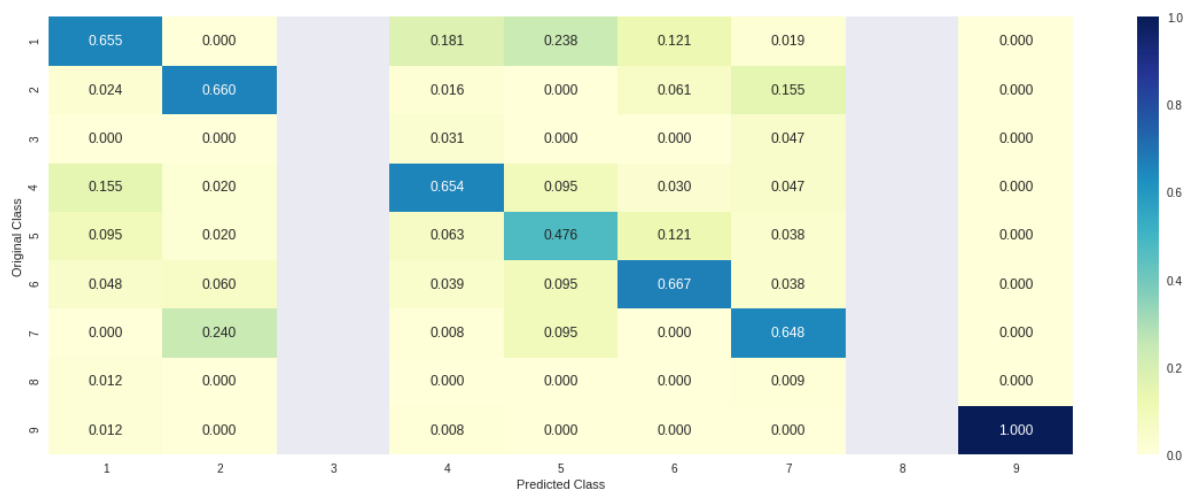
Log loss : 1.0405876140132555

Number of mis-classified points : 0.35150375939849626

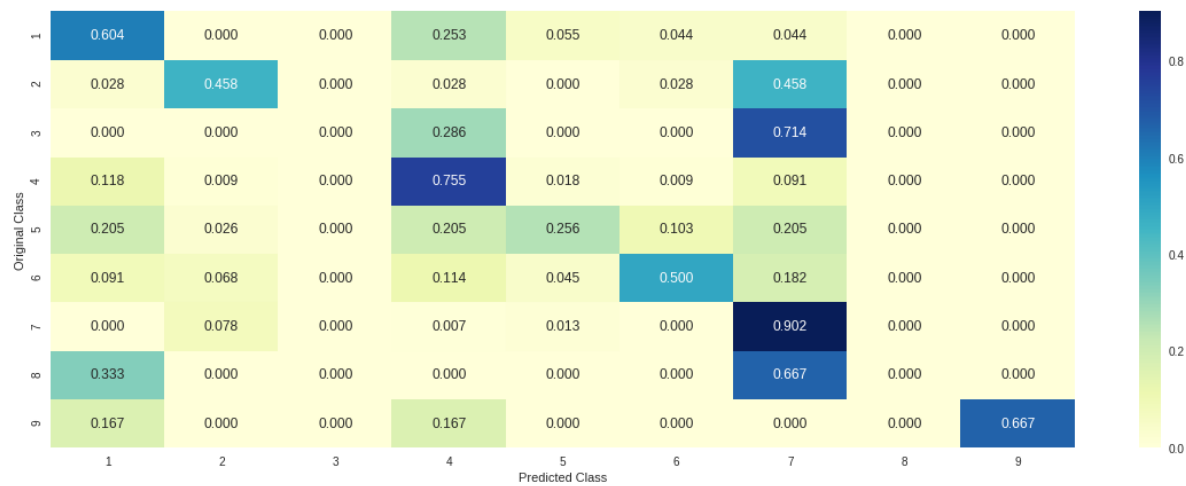
----- Confusion matrix -----



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



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



4.3.2.3. Feature Importance, Correctly Classified point

In [0]:

```

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_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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])

```

Predicted Class : 4

Predicted Class Probabilities: [[0.1654 0.076 0.0095 0.5892 0.0717 0.0057
0.0754 0.0054 0.0017]]

Actual Class : 4

```

-----
48 Text feature [dab2ip] present in test data point [True]
66 Text feature [traf2] present in test data point [True]
130 Text feature [scffbw7] present in test data point [True]
164 Text feature [907] present in test data point [True]
168 Text feature [beads] present in test data point [True]
171 Text feature [ask1] present in test data point [True]
185 Text feature [strategene] present in test data point [True]
192 Text feature [coordinate] present in test data point [True]
202 Text feature [327] present in test data point [True]
212 Text feature [chasing] present in test data point [True]
234 Text feature [impetus] present in test data point [True]
237 Text feature [puma] present in test data point [True]
244 Text feature [conjugates] present in test data point [True]
277 Text feature [aurora] present in test data point [True]
292 Text feature [knockout] present in test data point [True]
326 Text feature [pmdlg] present in test data point [True]
367 Text feature [annealed] present in test data point [True]
380 Text feature [arg505] present in test data point [True]
389 Text feature [arg465] present in test data point [True]
448 Text feature [381] present in test data point [True]
477 Text feature [ww] present in test data point [True]
499 Text feature [phosphodegron] present in test data point [True]
Out of the top 500 features 22 are present in query point

```

4.3.2.4. Feature Importance, Incorrectly Classified point

In [0]:

```

test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index])[0], 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])

```

```

Predicted Class : 3
Predicted Class Probabilities: [[0.2266 0.0836 0.3431 0.1661 0.0331 0.0333
0.0987 0.007 0.0084]]
Actual Class : 1
-----
142 Text feature [191] present in test data point [True]
158 Text feature [281] present in test data point [True]
179 Text feature [hnf] present in test data point [True]
218 Text feature [276] present in test data point [True]
222 Text feature [defective] present in test data point [True]
365 Text feature [214] present in test data point [True]
399 Text feature [nonproductive] present in test data point [True]
400 Text feature [breaks] present in test data point [True]
481 Text feature [hnf4a] present in test data point [True]
493 Text feature [monogenic] present in test data point [True]
Out of the top 500 features 10 are present in query point

```

4.4. Linear Support Vector Machines(Text TFIDF encoded)

4.4.1. Hyper paramter tuning

In [0]:

```

# read more about support vector machines with linear kernal's here http://scikit-learn.org/

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

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

# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modules/generated/sklearn.calibration.CalibratedClassifierCV.html
# -----
# default parameters
# 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=0)
    clf.fit(train_x_tfidf, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_tfidf, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_tfidf)
    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(True)
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

best_alpha = np.argmin(cv_log_error_array)
# clf = SVC(C=i, kernel='linear', probability=True, class_weight='balanced')
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=0)
clf.fit(train_x_tfidf, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")

```



```

sig_clf.fit(train_x_tfidf, train_y)

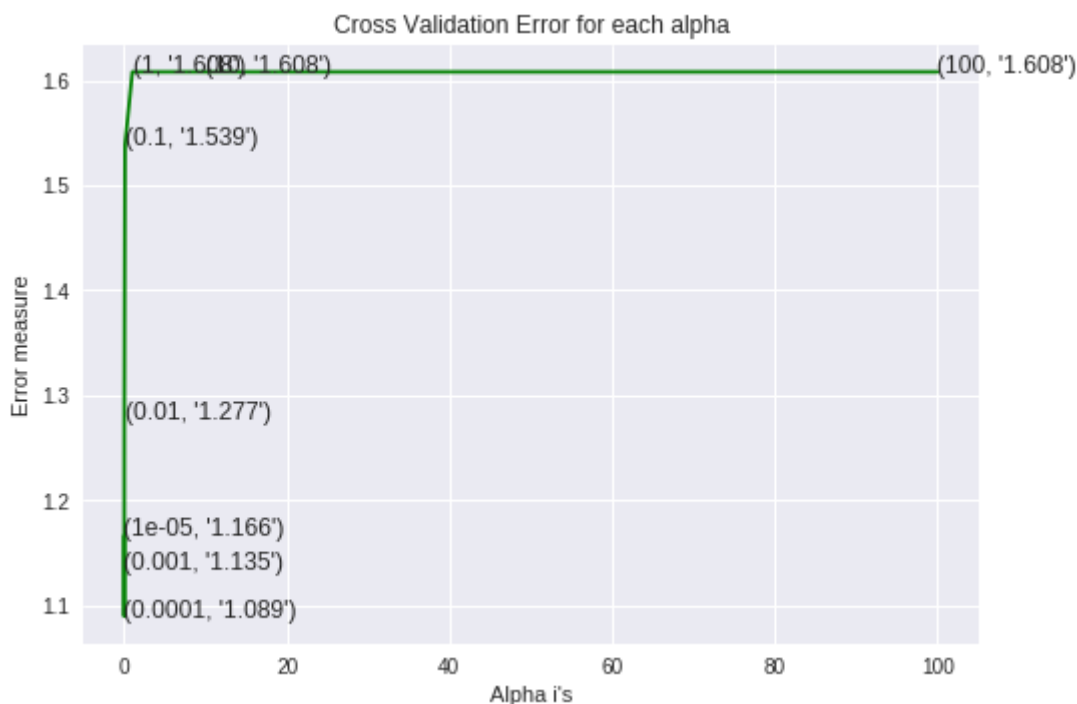
predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for C = 1e-05
Log Loss : 1.1661120253062023
for C = 0.0001
Log Loss : 1.0893347623363687
for C = 0.001
Log Loss : 1.1352053049598427
for C = 0.01
Log Loss : 1.2767846424021878
for C = 0.1
Log Loss : 1.5388065084965183
for C = 1
Log Loss : 1.6078515434118779
for C = 10
Log Loss : 1.6078515420264572
for C = 100
Log Loss : 1.6078515249402405

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4833088777360694
5
For values of best alpha = 0.0001 The cross validation log loss is: 1.08933
47623363687
For values of best alpha = 0.0001 The test log loss is: 1.075515281201142

```

4.4.2. Testing model with best hyper parameters

In [0]:

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

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

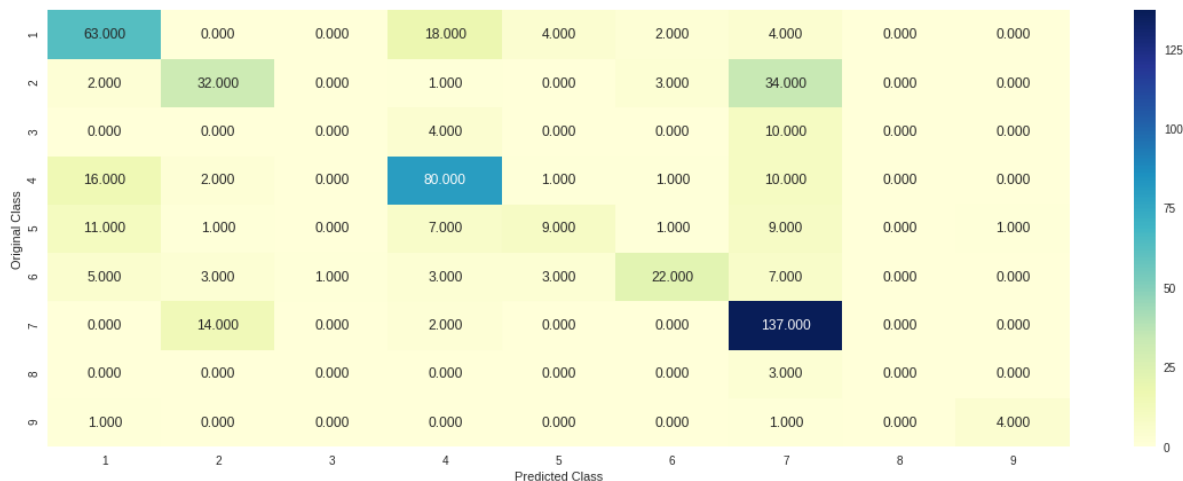
# Some of methods of SVM()
# fit(X, y, [sample_weight]) Fit the SVM model according to the given training data.
# predict(X) Perform classification on samples in X.
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/mathe
# -----

# 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_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

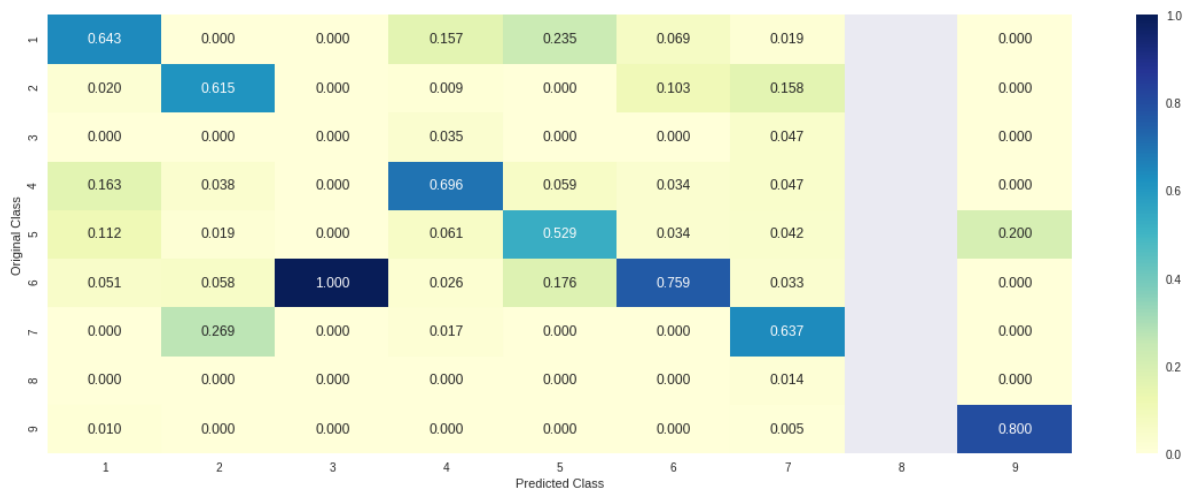
Log loss : 1.0893347623363687

Number of mis-classified points : 0.34774436090225563

----- 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 [0]:

```
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state=42)
clf.fit(train_x_tfidf, train_y)
test_point_index = 1
# test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]), 5))
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])
```

Predicted Class : 4

Predicted Class Probabilities: [[0.0821 0.1068 0.0861 0.5329 0.076 0.0099
0.0986 0.0053 0.0023]]

Actual Class : 4

 231 Text feature [01] present in test data point [True]
 479 Text feature [12] present in test data point [True]
 Out of the top 500 features 2 are present in query point

4.3.3.2. For Incorrectly classified point

In [0]:

```

test_point_index = 100
no_feature = 500
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[test_point_index]), 5))
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])

```

Predicted Class : 3

Predicted Class Probabilities: [[0.1782 0.0737 0.3756 0.1694 0.0255 0.0423
0.1254 0.0041 0.0059]]

Actual Class : 1

```

-----
137 Text feature [117] present in test data point [True]
148 Text feature [02] present in test data point [True]
152 Text feature [10] present in test data point [True]
172 Text feature [110] present in test data point [True]
175 Text feature [1155] present in test data point [True]
185 Text feature [09] present in test data point [True]
190 Text feature [116] present in test data point [True]
206 Text feature [1108] present in test data point [True]
234 Text feature [1238] present in test data point [True]
246 Text feature [112] present in test data point [True]
261 Text feature [1000] present in test data point [True]
285 Text feature [106] present in test data point [True]
295 Text feature [120] present in test data point [True]
310 Text feature [1028] present in test data point [True]
321 Text feature [124] present in test data point [True]
327 Text feature [025] present in test data point [True]
328 Text feature [11] present in test data point [True]
Out of the top 500 features 17 are present in query point

```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding(Gene, Variation) and TFIDF(Text))

In [0]:

```

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

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

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

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

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

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

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

```

```

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

predict_y = sig_clf.predict_proba(train_x_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The train log loss is:"
predict_y = sig_clf.predict_proba(cv_x_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The cross validation lo
predict_y = sig_clf.predict_proba(test_x_tfidf)
print('For values of best estimator = ', alpha[int(best_alpha/2)], "The test log loss is:",

```

```

for n_estimators = 100 and max depth = 5
Log Loss : 1.149178901099796
for n_estimators = 100 and max depth = 10
Log Loss : 1.1184233513966555
for n_estimators = 200 and max depth = 5
Log Loss : 1.140644882835148
for n_estimators = 200 and max depth = 10
Log Loss : 1.1113765538761116
for n_estimators = 500 and max depth = 5
Log Loss : 1.1343765052375674
for n_estimators = 500 and max depth = 10
Log Loss : 1.1073382578634328
for n_estimators = 1000 and max depth = 5
Log Loss : 1.1327537500665148
for n_estimators = 1000 and max depth = 10
Log Loss : 1.108734652234028
for n_estimators = 2000 and max depth = 5
Log Loss : 1.1312545424677865
for n_estimators = 2000 and max depth = 10
Log Loss : 1.1105480106829095
For values of best estimator = 500 The train log loss is: 0.531723568740166
8
For values of best estimator = 500 The cross validation log loss is: 1.1073
382578634328
For values of best estimator = 500 The test log loss is: 1.0853509654242688

```

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

In [0]:

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

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

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

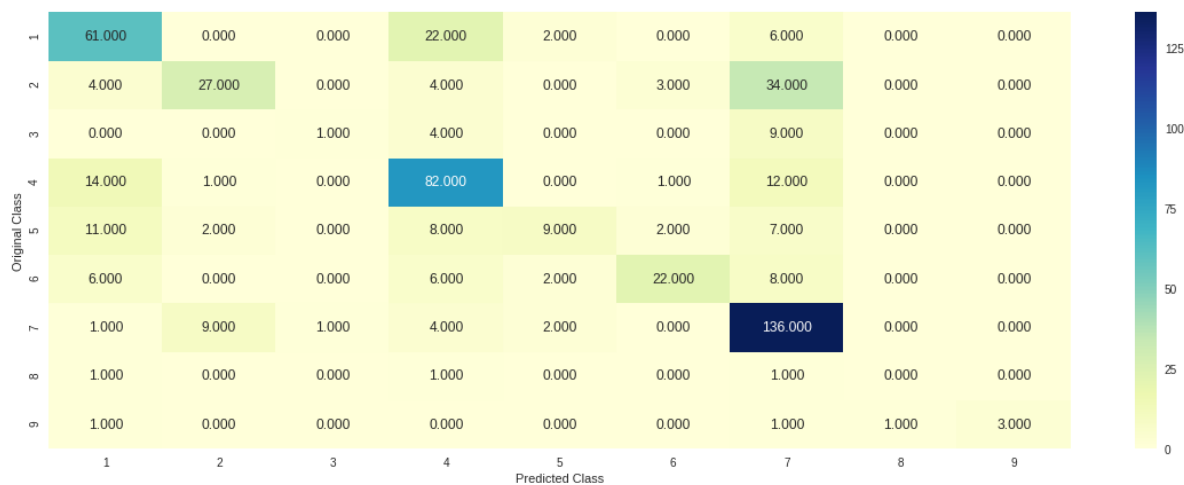
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-classifier/
# -----

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini', max_depth=None,
                             min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
                             max_leaf_nodes=None, min_impurity_split=None, bootstrap=True, oob_score=False,
                             n_jobs=1, random_state=None, verbose=0, class_weight=None)
predict_and_plot_confusion_matrix(train_x_tfidf, train_y, cv_x_tfidf, cv_y, clf)
```

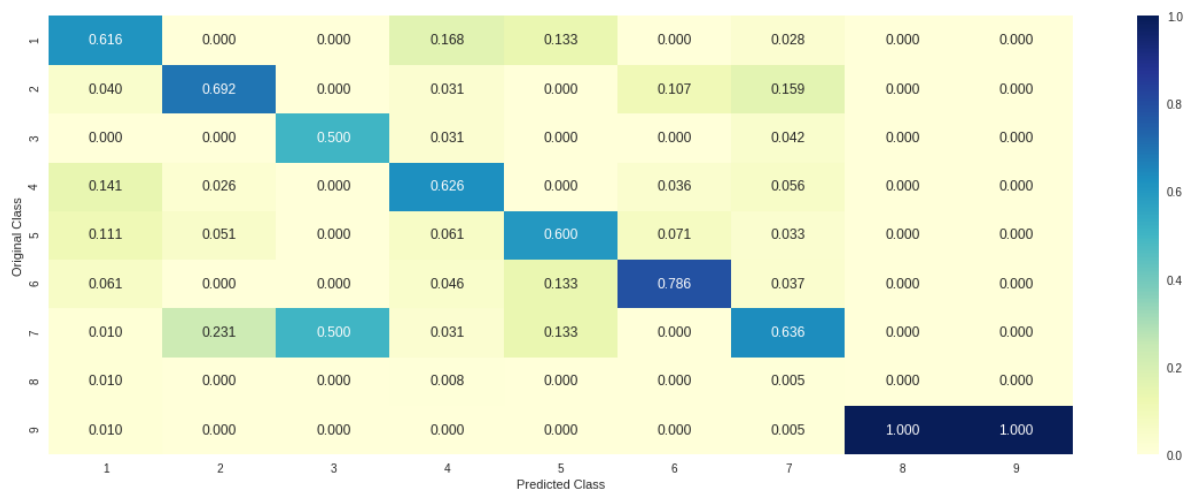
Log loss : 1.1073382578634328

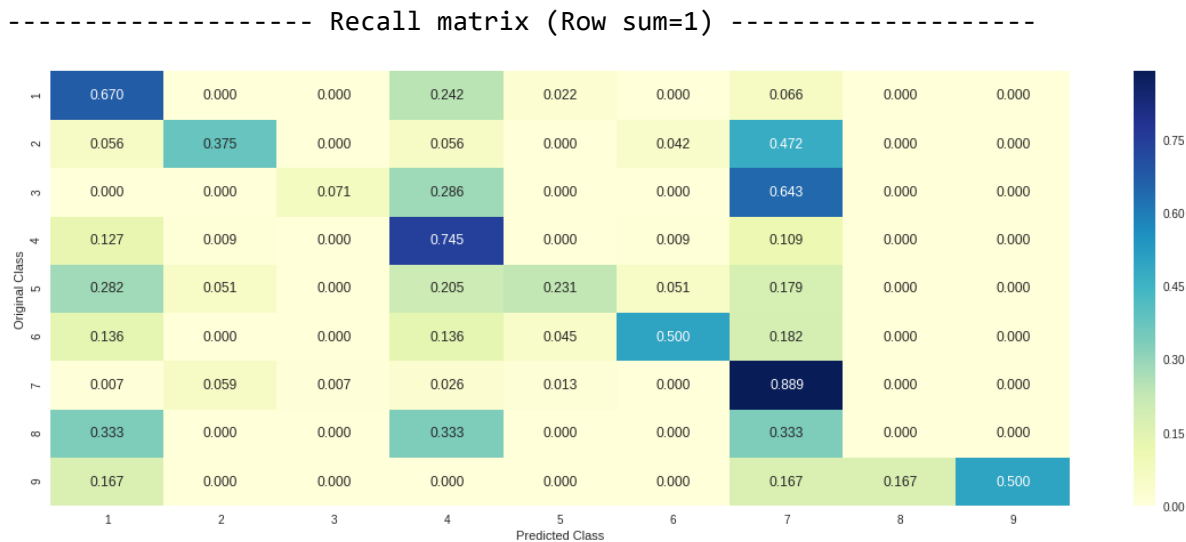
Number of mis-classified points : 0.35902255639097747

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



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





4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

In [0]:

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

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

Predicted Class : 4

Predicted Class Probabilities: [[0.1641 0.1121 0.0162 0.3987 0.0538 0.0381
0.1726 0.0087 0.0357]]

Actual Class : 4

46 Text feature [110] present in test data point [True]
Out of the top 100 features 1 are present in query point

4.5.3.2. Inorrectly Classified point

In [0]:

```

test_point_index = 100
no_feature = 100
predicted_cls = sig_clf.predict(test_x_tfidf[test_point_index])
print("Predicted Class :", predicted_cls[0])
print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_tfidf[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['TEXT'].iloc[test_point_index])

```

```

Predicted Class : 4
Predicted Class Probabilities: [[0.3183 0.0927 0.0174 0.3587 0.0578 0.0411
0.0988 0.006 0.0092]]
Actual Class : 1
-----
46 Text feature [110] present in test data point [True]
Out of the top 100 features 1 are present in query point

```

4.5.3. Hyper paramter tuning (With Response Coding)

In [0]:

```

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

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

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

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

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

alpha = [10,50,100,200,500,1000]
max_depth = [2,3,5,10]
cv_log_error_array = []
for i in alpha:
    for j in max_depth:
        print("for n_estimators =", i,"and max depth = ", j)
        clf = RandomForestClassifier(n_estimators=i, criterion='gini', max_depth=j, random_
        clf.fit(train_x_responseCoding, train_y)
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_responseCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_responseCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=1e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))
    ...

fig, ax = plt.subplots()
features = np.dot(np.array(alpha)[: ,None],np.array(max_depth)[None]).ravel()
ax.plot(features, cv_log_error_array,c='g')
for i, txt in enumerate(np.round(cv_log_error_array,3)):
    ax.annotate((alpha[int(i/4)],max_depth[int(i%4)],str(txt)), (features[i],cv_log_error_a
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()

```

```

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

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

```

```

for n_estimators = 10 and max depth = 2
Log Loss : 2.2657048897349608
for n_estimators = 10 and max depth = 3
Log Loss : 1.7459205010556096
for n_estimators = 10 and max depth = 5
Log Loss : 1.4368353925512503
for n_estimators = 10 and max depth = 10
Log Loss : 1.904597809032912
for n_estimators = 50 and max depth = 2
Log Loss : 1.7221951095007484
for n_estimators = 50 and max depth = 3
Log Loss : 1.4984825877845531
for n_estimators = 50 and max depth = 5
Log Loss : 1.4593628982873716
for n_estimators = 50 and max depth = 10
Log Loss : 1.8434939703555409
for n_estimators = 100 and max depth = 2
Log Loss : 1.6182209245331227
for n_estimators = 100 and max depth = 3
Log Loss : 1.5199297988828253
for n_estimators = 100 and max depth = 5
Log Loss : 1.4177501184246677
for n_estimators = 100 and max depth = 10
Log Loss : 1.8227504417195126
for n_estimators = 200 and max depth = 2
Log Loss : 1.6622571648074496
for n_estimators = 200 and max depth = 3
Log Loss : 1.4800771339141767
for n_estimators = 200 and max depth = 5
Log Loss : 1.4412060242341358
for n_estimators = 200 and max depth = 10
Log Loss : 1.7892406351442258
for n_estimators = 500 and max depth = 2
Log Loss : 1.715950314170445
for n_estimators = 500 and max depth = 3
Log Loss : 1.5658682738699774
for n_estimators = 500 and max depth = 5
Log Loss : 1.4445360301518217
for n_estimators = 500 and max depth = 10
Log Loss : 1.8421097596928397
for n_estimators = 1000 and max depth = 2
Log Loss : 1.6834927870864949
for n_estimators = 1000 and max depth = 3
Log Loss : 1.5631973035931377
for n_estimators = 1000 and max depth = 5
Log Loss : 1.4440000702724120

```

```
Log Loss : 1.4449980792724129
```

```
for n_estimators = 1000 and max depth = 10
```

```
Log Loss : 1.85233132619749
```

```
For values of best alpha = 100 The train log loss is: 0.060702709444608406
```

```
For values of best alpha = 100 The cross validation log loss is: 1.417750118424668
```

```
For values of best alpha = 100 The test log loss is: 1.3806278998341923
```

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

In [0]:

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

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

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

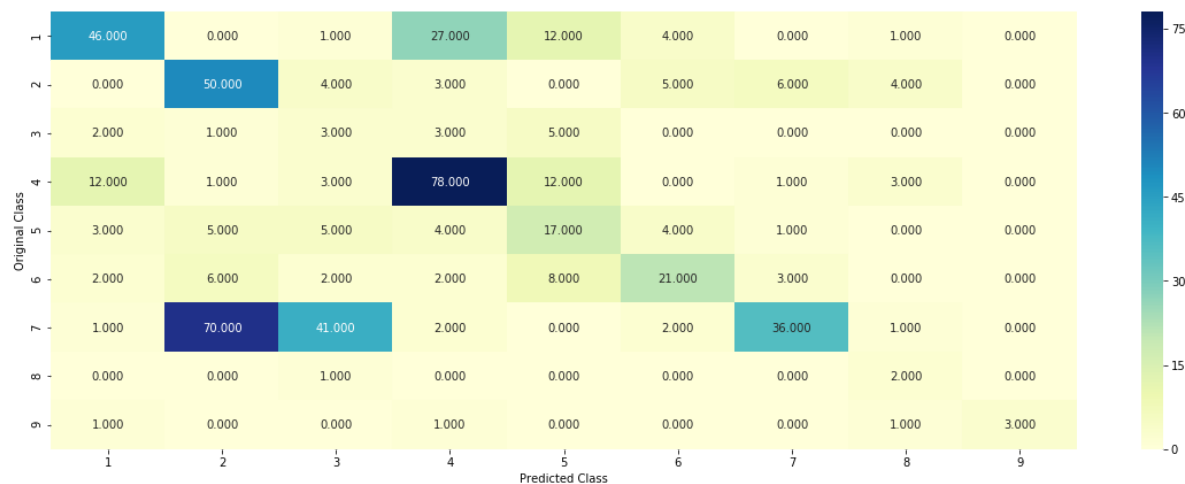
# -----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/random-forest-classifier/
# -----

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

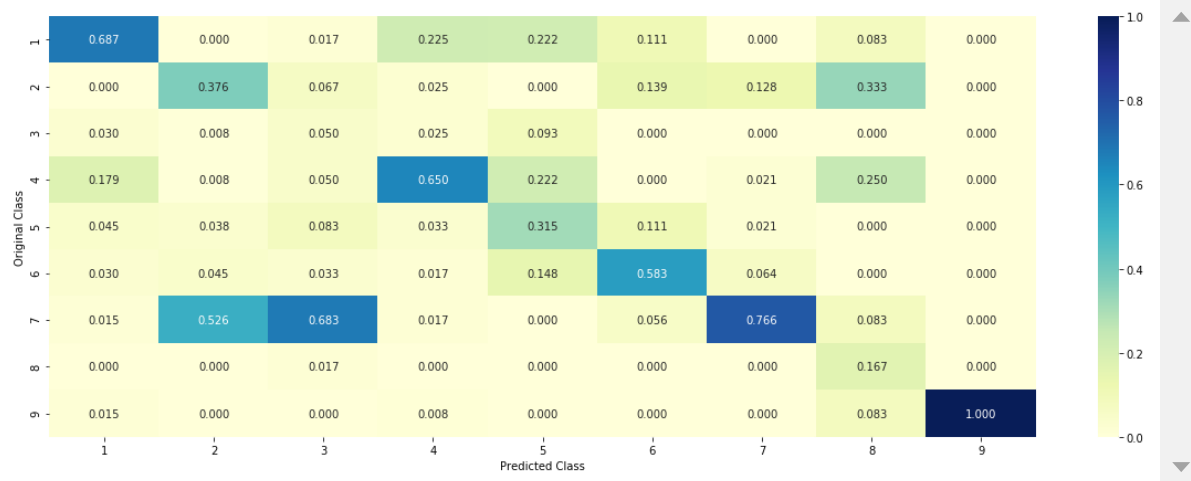
Log loss : 1.4177501184246677

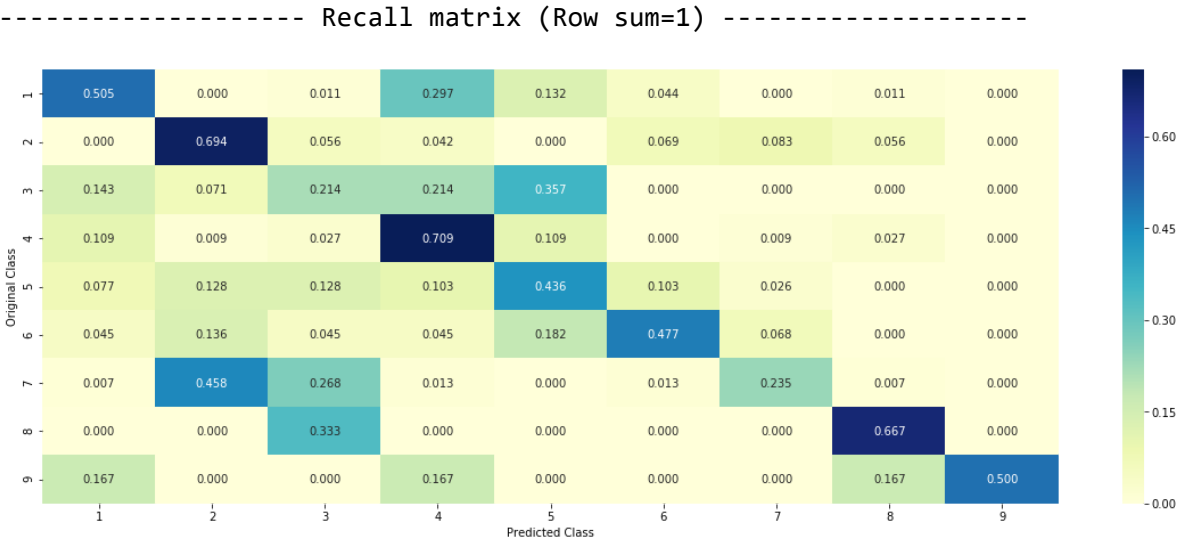
Number of mis-classified points : 0.518796992481203

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



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





4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

In [0]:

```

clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini', max_c
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)), 5))
print("Actual Class :", test_y[test_point_index])
indices = np.argsort(-clf.feature_importances_)
print("-"*50)
for i in indices:
    if i<9:
        print("Gene is important feature")
    elif i<18:
        print("Variation is important feature")
    else:
        print("Text is important feature")

```

```

Predicted Class : 2
Predicted Class Probabilities: [[0.0143 0.5044 0.1471 0.0191 0.0245 0.065
0.1724 0.039 0.0142]]
Actual Class : 7

```

```

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

```

4.5.5.2. Incorrectly Classified point

In [0]:

```

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)), 5))
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 : 7

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

Actual Class : 7

```

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

```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

In [0]:

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

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

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

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

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

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

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

clf1 = SGDClassifier(alpha=0.001, penalty='l2', loss='log', class_weight='balanced', random_state=None)
clf1.fit(train_x_tfidf, train_y)
sig_clf1 = CalibratedClassifierCV(clf1, method="sigmoid")

clf2 = SGDClassifier(alpha=1, penalty='l2', loss='hinge', class_weight='balanced', random_state=None)
clf2.fit(train_x_tfidf, train_y)
sig_clf2 = CalibratedClassifierCV(clf2, method="sigmoid")
```

```

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

sig_clf1.fit(train_x_tfidf, train_y)
print("Logistic Regression : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_proba(cv_x_tfidf))))
sig_clf2.fit(train_x_tfidf, train_y)
print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_proba(cv_x_tfidf))))
sig_clf3.fit(train_x_tfidf, train_y)
print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_tfidf))))
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)
    sclf.fit(train_x_tfidf, 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_tfidf))))
    log_error = log_loss(cv_y, sclf.predict_proba(cv_x_tfidf))
    if best_alpha > log_error:
        best_alpha = log_error

```

```

Logistic Regression : Log Loss: 1.07
Support vector machines : Log Loss: 1.61
Naive Bayes : Log Loss: 1.21

```

```

-----
Stacking Classifier : for the value of alpha: 0.000100 Log Loss: 2.178
Stacking Classifier : for the value of alpha: 0.001000 Log Loss: 2.040
Stacking Classifier : for the value of alpha: 0.010000 Log Loss: 1.540
Stacking Classifier : for the value of alpha: 0.100000 Log Loss: 1.190
Stacking Classifier : for the value of alpha: 1.000000 Log Loss: 1.369
Stacking Classifier : for the value of alpha: 10.000000 Log Loss: 1.764

```

4.7.2 testing the model with the best hyper parameters

In [0]:

```

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

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

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

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

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

```

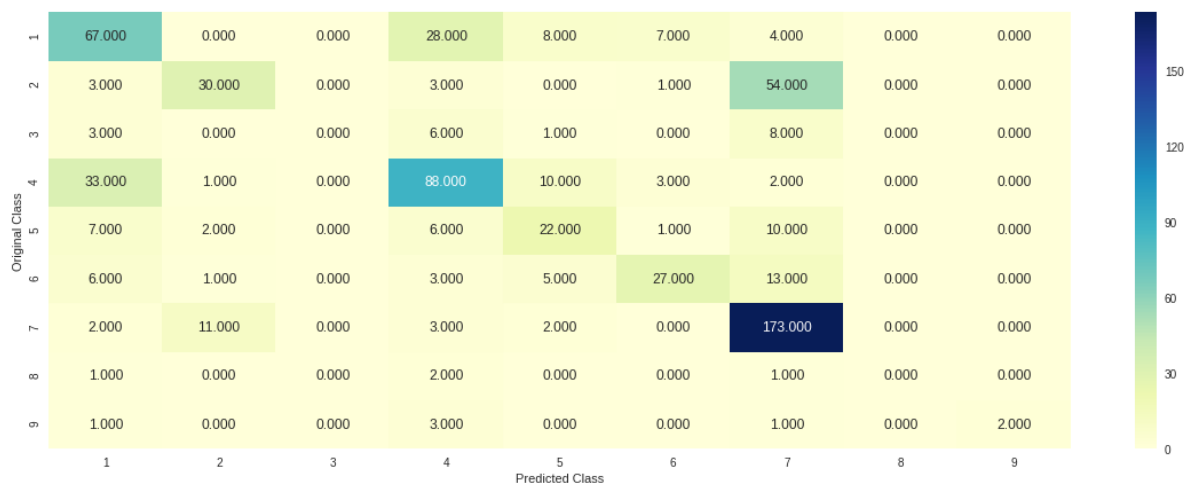
Log loss (train) on the stacking classifier : 0.5674336922867708

Log loss (CV) on the stacking classifier : 1.1899030090349125

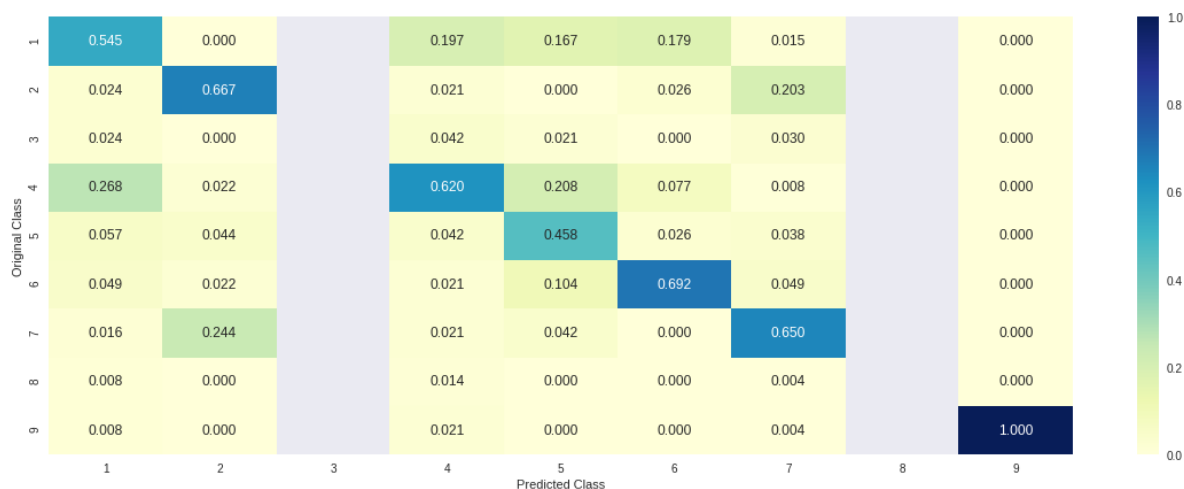
Log loss (test) on the stacking classifier : 1.1631688357227483

Number of missclassified point : 0.3849624060150376

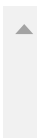
----- Confusion matrix -----

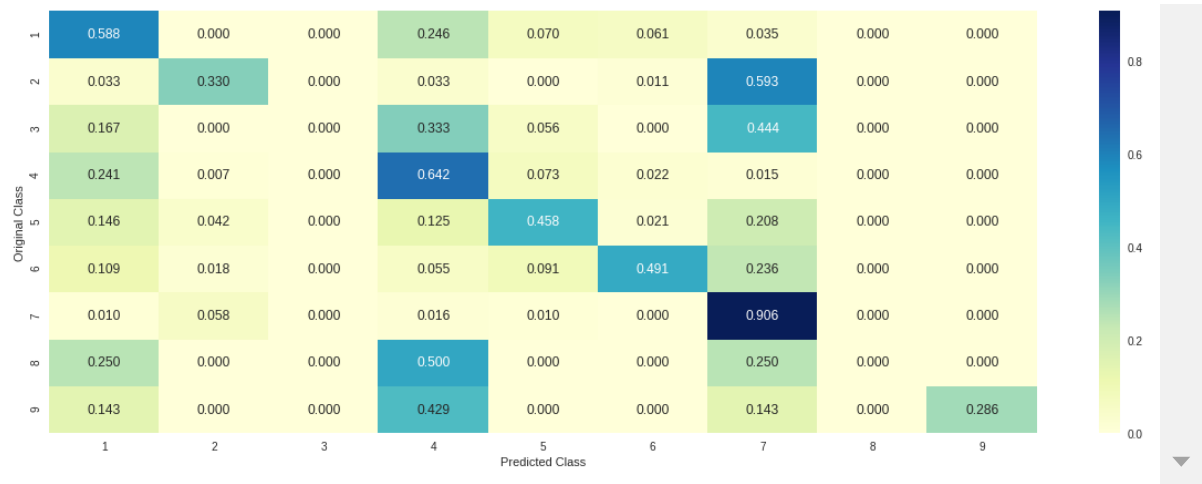


----- Precision matrix (Column Sum=1) -----



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





4.7.3 Maximum Voting classifier

In [0]:

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

Log loss (train) on the VotingClassifier : 0.8358537888416202

Log loss (CV) on the VotingClassifier : 1.2117360597329536

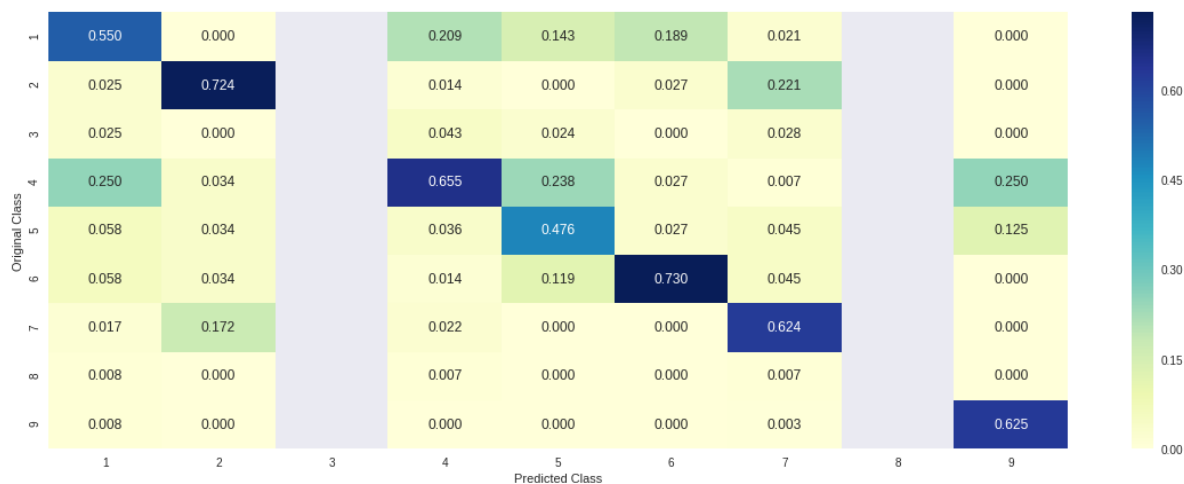
Log loss (test) on the VotingClassifier : 1.206587944764749

Number of missclassified point : 0.3819548872180451

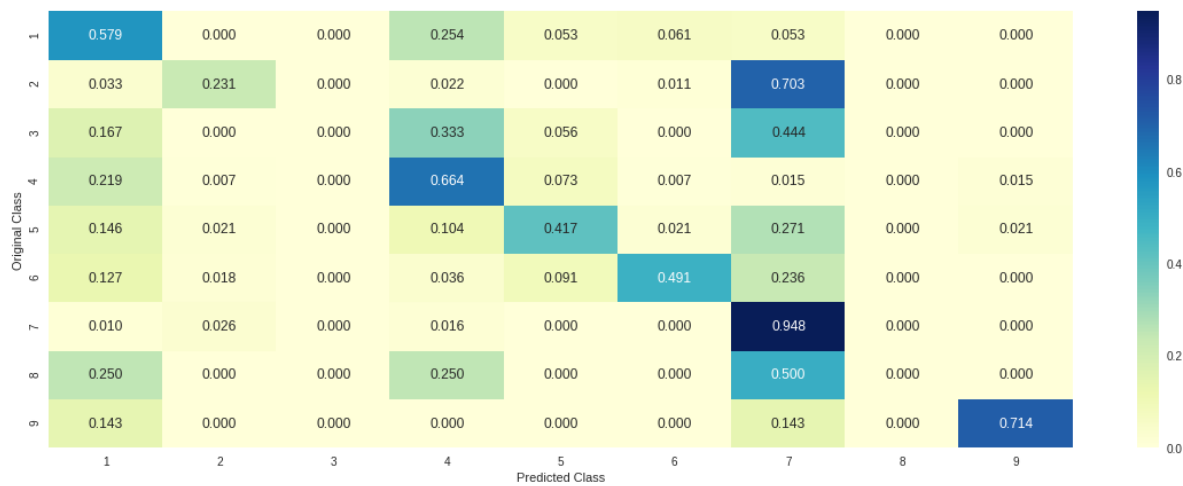
----- Confusion matrix -----



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



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



5. Assignments

1. Apply All the models with tf-idf features (Replace CountVectorizer with TfidfVectorizer and run the same cells)
2. Instead of using all the words in the dataset, use only the top 1000 words based on tf-idf values
3. Apply Logistic regression with CountVectorizer Features, including both unigrams and bigrams
4. Try any of the feature engineering techniques discussed in the course to reduce the CV and test log-loss to a value less than 1.0

Logistic regression with CountVectorizer Features, including both unigrams and bigrams

In [0]:

```
vectorizer = CountVectorizer(min_df=3, ngram_range=(1,2)) # unigrams, bigrams

train_text_ngram = vectorizer.fit_transform(train_df['TEXT'])
test_text_ngram = vectorizer.transform(test_df['TEXT'])
cv_text_ngram = vectorizer.transform(cv_df['TEXT'])
```

In [0]:

```
train_text_features_ngram = vectorizer.get_feature_names() #feature_names
```

In [0]:

```
# DataMatrix
```

```
train_x_ngram = hstack((
    train_gene_var_onehotCoding,
    train_text_ngram
)).tocsr()

test_x_ngram = hstack((
    test_gene_var_onehotCoding,
    test_text_ngram
)).tocsr()

cv_x_ngram = hstack((
    cv_gene_var_onehotCoding,
    cv_text_ngram
)).tocsr()

print("One hot encoding features with Text(unigram, bigram).")
print("(number of data points * number of features) in train data = ", train_x_ngram.shape)
print("(number of data points * number of features) in test data = ", test_x_ngram.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_ngram
```

```
One hot encoding features with Text(unigram, bigram).
(number of data points * number of features) in train data = (2124, 785039)
(number of data points * number of features) in test data = (665, 785039)
(number of data points * number of features) in cross validation data = (53
2, 785039)
```

In [0]:

```

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_ngram, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_ngram, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_ngram)
    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(True)
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_ngram, train_y)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_ngram, train_y)

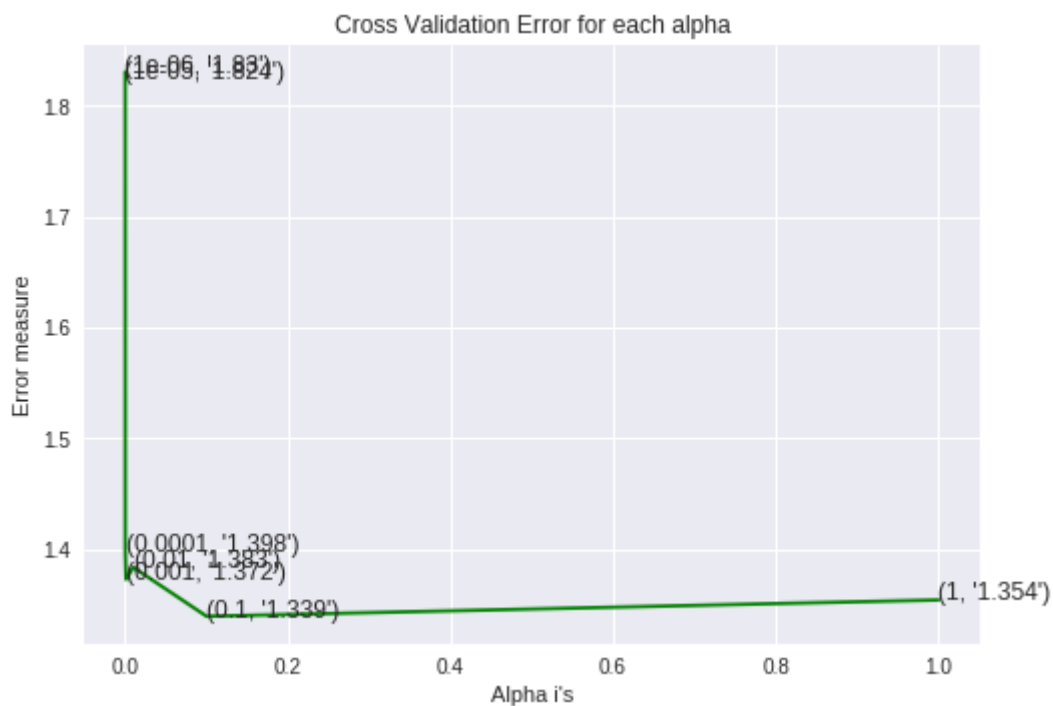
predict_y = sig_clf.predict_proba(train_x_ngram)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_
predict_y = sig_clf.predict_proba(cv_x_ngram)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(test_x_ngram)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 1e-06
Log Loss : 1.8304997567764278
for alpha = 1e-05
Log Loss : 1.8243144666335838
for alpha = 0.0001
Log Loss : 1.3981201849177265
for alpha = 0.001
Log Loss : 1.3724509687254631
for alpha = 0.01
Log Loss : 1.3833253014551083
for alpha = 0.1
Log Loss : 1.3391438814354693
for alpha = 1
Log Loss : 1.3538557445748989

```

For values of best alpha = 0.1 The train log loss is: 1.0635089161511173

For values of best alpha = 0.1 The cross validation log loss is: 1.3391438814354693

For values of best alpha = 0.1 The test log loss is: 1.3115091714735665

Testing the model with best hyper paramters

In [0]:

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

Log loss : 1.3484842674172335

Number of mis-classified points : 0.424812030075188

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



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



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



Feature Engineering

In [0]:

```
# Text (TFIDF encoded)
vectorizer = TfidfVectorizer(ngram_range=(1,4), max_features=5000) # unigram, bigram

train_text_4gram_5000 = vectorizer.fit_transform(train_df['TEXT'])
test_text_4gram_5000 = vectorizer.transform(test_df['TEXT'])
cv_text_4gram_5000 = vectorizer.transform(cv_df['TEXT'])

print(train_text_4gram_5000.shape, y_train.shape)
print(test_text_4gram_5000.shape, y_test.shape)
print(cv_text_4gram_5000.shape, y_cv.shape)
```

```
(2124, 5000) (2124,)
(665, 5000) (665,)
(532, 5000) (532,)
```

In [0]:

```
# Data Matrix

X_tr = hstack((
    train_gene_feature_onehotCoding,
    train_variation_feature_onehotCoding,
    train_text_4gram_5000,
)).tocsr()

X_te = hstack((
    test_gene_feature_onehotCoding,
    test_variation_feature_onehotCoding,
    test_text_4gram_5000,
)).tocsr()

X_cr = hstack((
    cv_gene_feature_onehotCoding,
    cv_variation_feature_onehotCoding,
    cv_text_4gram_5000,
)).tocsr()

print(X_tr.shape, y_train.shape)
print(X_te.shape, y_test.shape)
print(X_cr.shape, y_cv.shape)
```

```
(2124, 7193) (2124,)
(665, 7193) (665,)
(532, 7193) (532,)
```

In [0]:

Logistic Regression

```

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(X_tr, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_tr, y_train)
    sig_clf_probs = sig_clf.predict_proba(X_cr)
    cv_log_error_array.append(log_loss(y_cv, sig_clf_probs, labels=clf.classes_, eps=1e-15))
    print("Log Loss :", log_loss(y_cv, 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(True)
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(X_tr, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_tr, y_train)

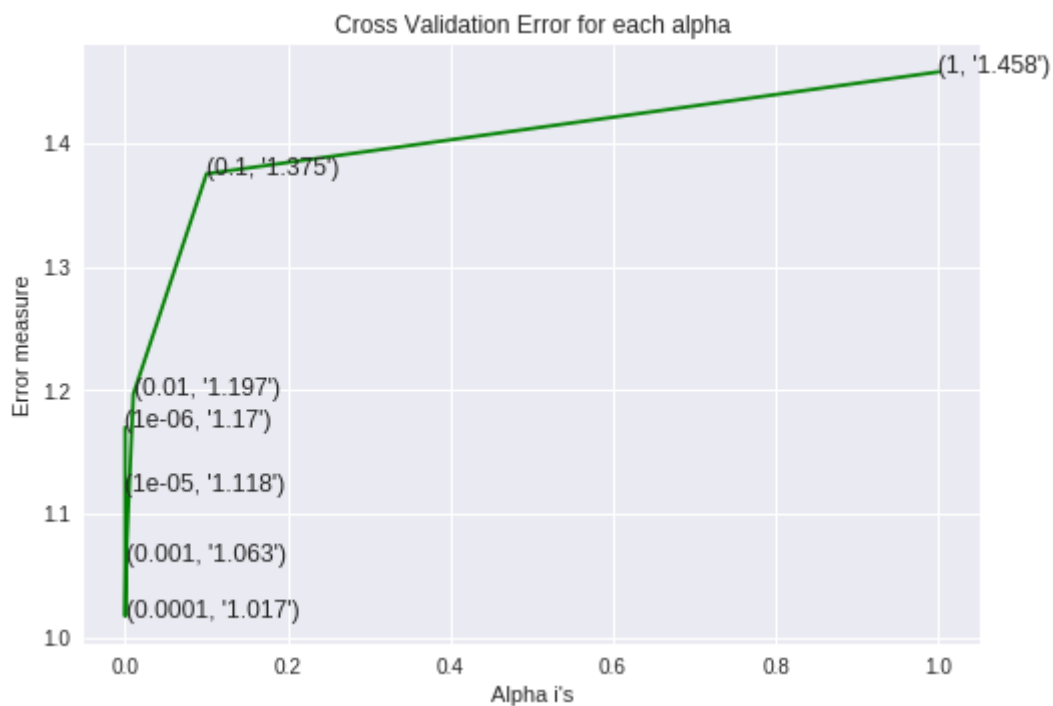
predict_y = sig_clf.predict_proba(X_tr)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_
predict_y = sig_clf.predict_proba(X_cr)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:"
predict_y = sig_clf.predict_proba(X_te)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_

```

```

for alpha = 1e-06
Log Loss : 1.1700275722240303
for alpha = 1e-05
Log Loss : 1.1181365768086642
for alpha = 0.0001
Log Loss : 1.0172010176798834
for alpha = 0.001
Log Loss : 1.0626180097965898
for alpha = 0.01
Log Loss : 1.1968609674968025
for alpha = 0.1
Log Loss : 1.3751422922744503
for alpha = 1
Log Loss : 1.4575491415118715

```



For values of best alpha = 0.0001 The train log loss is: 0.42348335815281435

For values of best alpha = 0.0001 The cross validation log loss is: 1.0172010176798834

For values of best alpha = 0.0001 The test log loss is: 0.9984297436111965

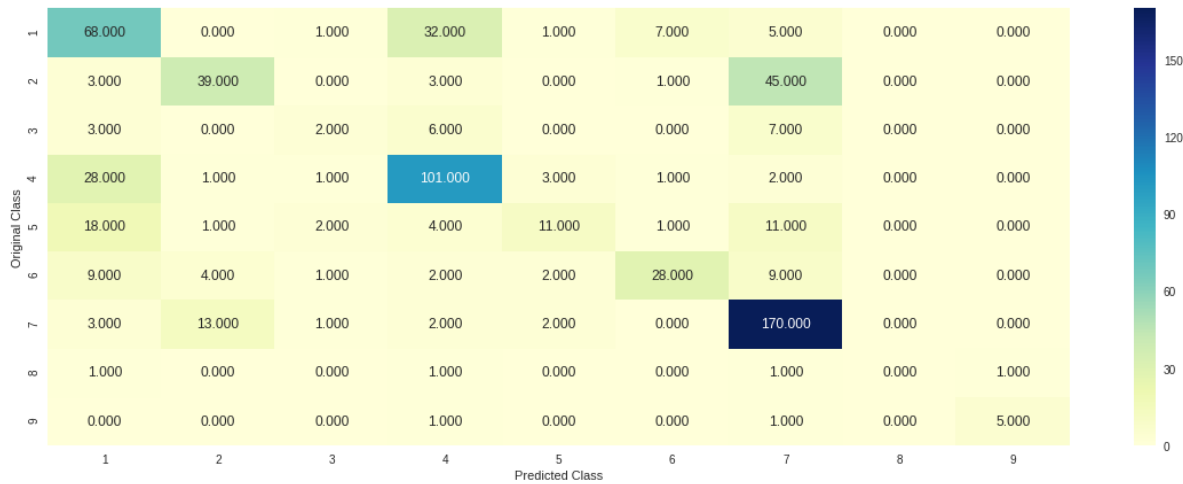
In [0]:

```
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='l2')
predict_and_plot_confusion_matrix(X_tr, y_train, X_te, y_test, clf)
```

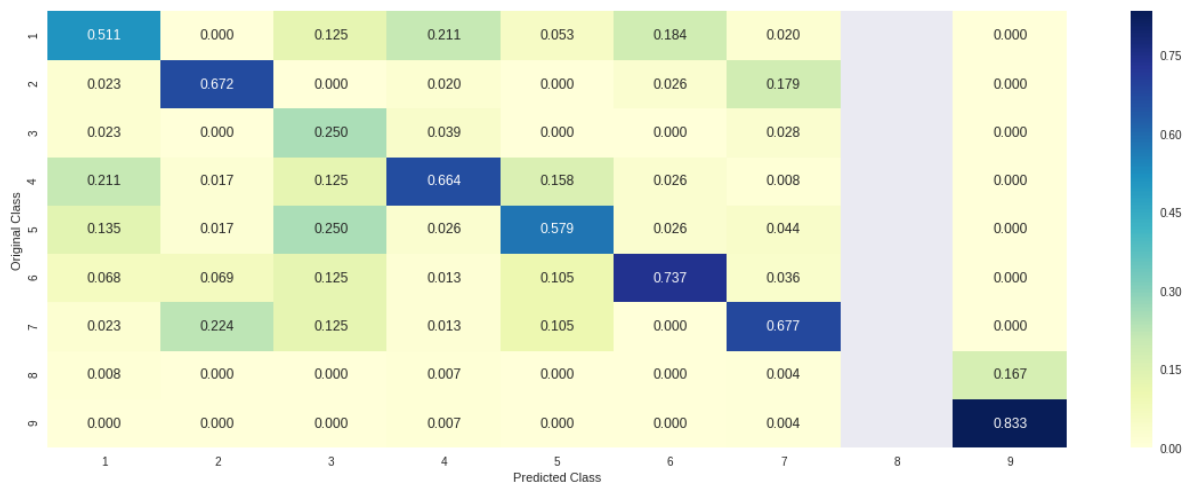
Log loss : 0.9984297436111965

Number of mis-classified points : 0.362406015037594

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



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



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



Conclusion

In [0]:

```

from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ['Vectorizer', 'Model', 'Hyper Parameter', 'Log-Loss']

x.add_row(['TFIDF', 'Multinomial NB', 'alpha=0.0001', '1.213'])
x.add_row(['TFIDF', 'KNN', 'k=5', '1.115'])
x.add_row(['TFIDF', 'Logistic Regression(Class balancing)', 'alpha=0.0001', '1.034'])
x.add_row(['TFIDF', 'Logistic Regression', 'alpha=0.0001', '1.040'])
x.add_row(['TFIDF', 'SVC', 'alpha=0.0001', '1.089'])
x.add_row(['TFIDF', 'Random Forest', 'n_estimators=500 \n max_depth=5', '1.107'])
x.add_row(['TFIDF', 'Random Forest(Response Coding)', 'n_estimators=100 \n max_depth=10', '1.417'])
x.add_row(['BOW', 'Logistic Regression(unigrams, bigrams)', 'alpha=0.1', '1.348'])
x.add_row(['TFIDF', 'Logistic Regression(4 grams)', 'alpha=0.0001', '0.998'])

print(x)

```

```

+-----+-----+-----+-----+
-----+
| Vectorizer |           Model           | Hyper Parameter |
Log-Loss |
+-----+-----+-----+-----+
-----+
|  TFIDF    |           Multinomial NB           |  alpha=0.0001  |
1.213  |
|  TFIDF    |           KNN           |          k=5    |
1.115  |
|  TFIDF    | Logistic Regression(Class balancing) |  alpha=0.0001  |
1.034  |
|  TFIDF    |           Logistic Regression           |  alpha=0.0001  |
1.040  |
|  TFIDF    |           SVC           |  alpha=0.0001  |
1.089  |
|  TFIDF    |           Random Forest           | n_estimators=500 |
1.107  |
|           |           |           max_depth=5 |
|           |           |           |
|  TFIDF    | Random Forest(Response Coding) | n_estimators=100 |
1.417  |
|           |           |           max_depth=10 |
|           |           |           |
|    BOW    | Logistic Regression(unigrams, bigrams) |  alpha=0.1    |
1.348  |
|  TFIDF    |           Logistic Regression(4 grams)           |  alpha=0.0001  |
0.998  |
+-----+-----+-----+-----+
-----+

```

Steps followed for Assignment: Personalized Cancer Diagnosis

- Loaded dataset, basic statistics and text preprocessing.
- Performed EDA with given features Gene , Variation and Text .
- Machine Learning Models
 - Splitted data into Train, Test and CV.
 - One hot encoding for Gene and Variation .
 - Encoded feature Text with TFIDF. Selected top 1000 words using max_features=1000 .

- Applied Multinomial Naive Bayes.
- Applied KNN.
- Applied Logistic Regression with and without class balancing.
- Applied SVC.
- Applied Random Forest.
- Applied Stacking(LR, SVC, NB).
- Applied Logistic Regression with Text BOW encoded(unigram and bigram).
- Feature engineering for reducing log loss: Applied Logistic regression with TFIDF, 4 grams and selected top 5000 words .

Logistic Regression gives the lowest log-loss of 0.998 with Text TFIDF encoded .