PersonalizedCancerDiagnosis

August 8, 2019

Personalized cancer diagnosis

1. Business Problem

1.1. Description
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/
Data: Memorial Sloan Kettering Cancer Center (MSKCC)
Download training_variants.zip and training_text.zip from Kaggle.
Context:
Source: https://www.kaggle.com/c/msk-redefining-cancer-treatment/discussion/35336#198462
Problem statement :
Classify the given genetic variations/mutations based on evidence from text-based clinical literature.
1.2. Source/Useful Links
Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data
2.1.1. Data Overview

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

- We have two data files: one conatins the information about the genetic mutations and the other contains the clinical evidence (text) that human experts/pathologists use to classify the genetic mutations.

- Both these data files are have a common column called ID
- Data file's information:

```
<li>
training_variants (ID , Gene, Variations, Class)
</li>
<li>
training_text (ID, Text)
</li>
```

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 cla
```

2.2.2. Performance Metric

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

Metric(s): * Multi class log-loss * Confusion matrix

2.2.3. Machine Learing Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

```
In [1]: 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
        from prettytable import PrettyTable
```

```
C:\Users\hp\Anaconda3\lib\site-packages\sklearn\externals\six.py:31: DeprecationWarning: The mo
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

```
In [23]: data = pd.read_csv('training_variants') #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[23]:    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
Fields are
<ul>
    <li><b>ID : </b>the id of the row used to link the mutation to the clinical evidence</li>
    <li><b>Gene : </b>the gene where this genetic mutation is located </li>
    <li><b>Variation : </b>the aminoacid change for this mutations </li>
    <li><b>Class :</b> 1-9 the class this genetic mutation has been classified on</li>
</ul>
```

3.1.2. Reading Text Data

```
In [24]: # note the seprator in this file
         data_text =pd.read_csv("training_text",sep="\|\|",engine="python",names=["ID","TEXT"]
         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[24]:    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 [25]: # loading stop words from nltk library
         stop_words = set(stopwords.words('english'))


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

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

                 data_text[column][index] = string
```

```
In [26]: #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 : 213.7354342 seconds
```

```
In [27]: #merging both gene_variations and text data based on ID
         result = pd.merge(data, data_text,on='ID', how='left')
         result.head()

Out[27]:    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

                                                         TEXT
         0  cyclin dependent kinases cdks regulate variety...
         1  abstract background non small cell lung cancer...
         2  abstract background non small cell lung cancer...
         3  recent evidence demonstrated acquired uniparen...
         4  oncogenic mutations monomeric casitas b lineag...
```

```
In [28]: result[result.isnull().any(axis=1)]

Out[28]:         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 [29]: result.loc[result['TEXT'].isnull(),'TEXT'] = result['Gene'] +' '+result['Variation']
```

```
In [30]: result[result['ID']==1277]

Out[30]:         ID    Gene             Variation  Class                         TEXT
         1277  1277  ARID5B  Truncating Mutations      1  ARID5B Truncating Mutations
```

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 [31]: y_true = result['Class'].values
         # replacing multiple spaces with underscore
         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 varai
         X_train, test_df, y_train, y_test = train_test_split(result, y_true, stratify=y_true,
         # split the train data into train and cross validation by maintaining same distributi
         train_df, cv_df, y_train, y_cv = train_test_split(X_train, y_train, stratify=y_train,
```

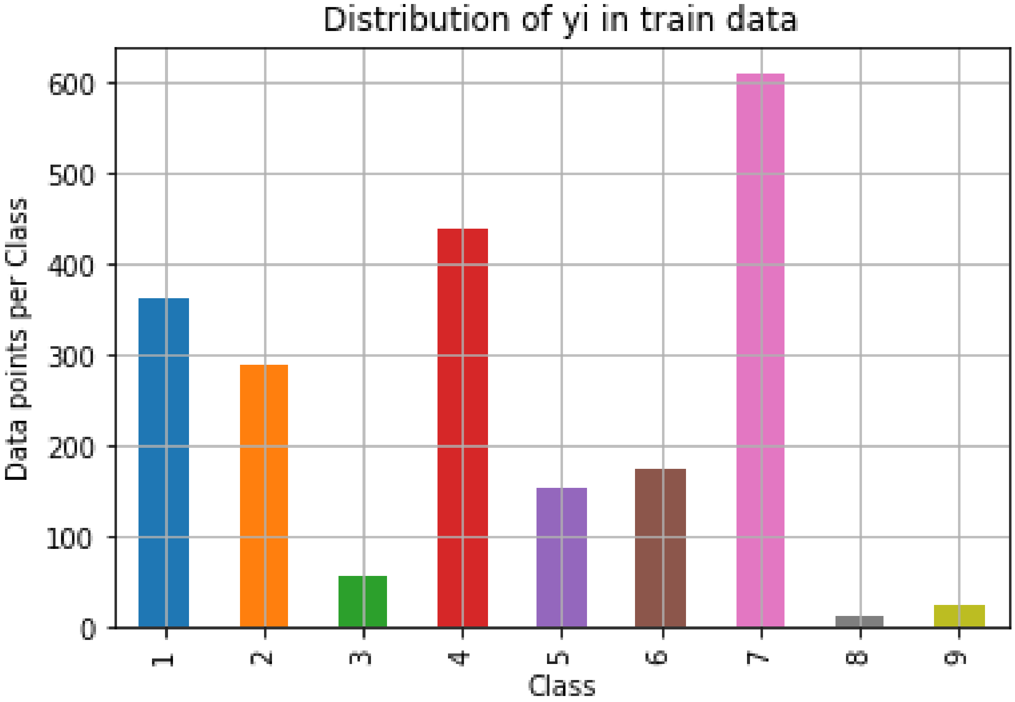
We split the data into train, test and cross validation data sets, preserving the ratio of class distribution in the original data set

```
In [32]: 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
```

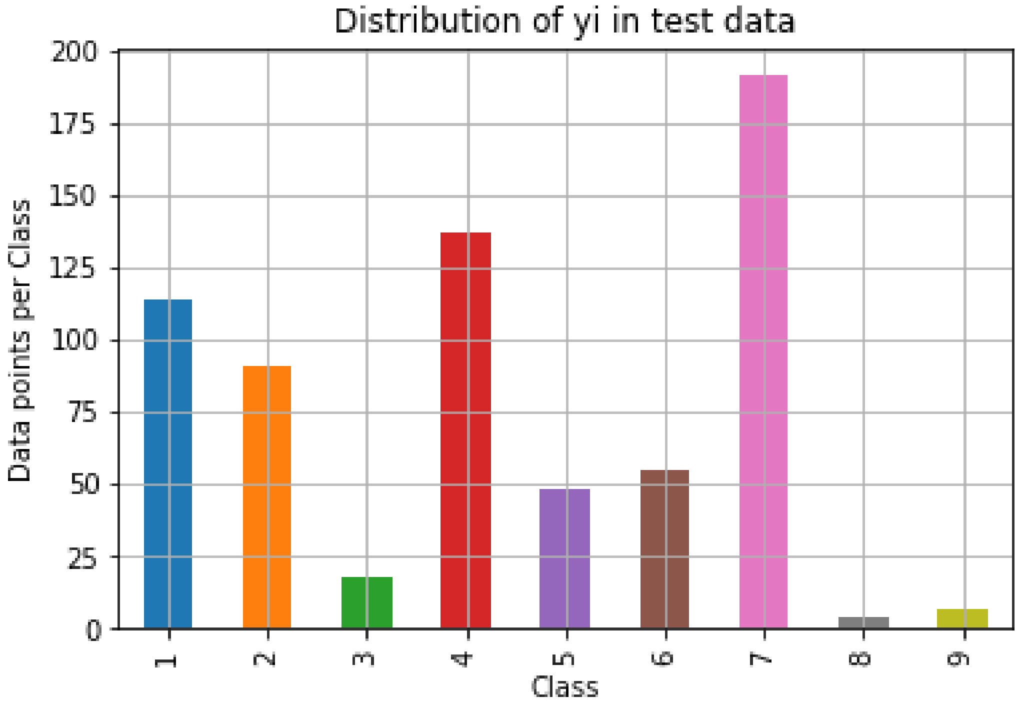
3.1.4.2. Distribution of y_i's in Train, Test and Cross Validation datasets

```
In [33]: # it returns a dict, keys as class labels and values as the number of data points in
         train_class_distribution = train_df['Class'].value_counts().sortlevel()
         test_class_distribution = test_df['Class'].value_counts().sortlevel()
         cv_class_distribution = cv_df['Class'].value_counts().sortlevel()

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

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


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

         # ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.htm
         # -(train_class_distribution.values): the minus sign will give us in decreasing order
         sorted_yi = np.argsort(-test_class_distribution.values)
         for i in sorted_yi:
             print('Number of data points in class', i+1, ':',test_class_distribution.values[i
         print('-'*80)
         my_colors = 'rgbkymc'
         cv_class_distribution.plot(kind='bar')
         plt.xlabel('Class')
         plt.ylabel('Data points per Class')
         plt.title('Distribution of yi in cross validation data')
         plt.grid()
         plt.show()

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

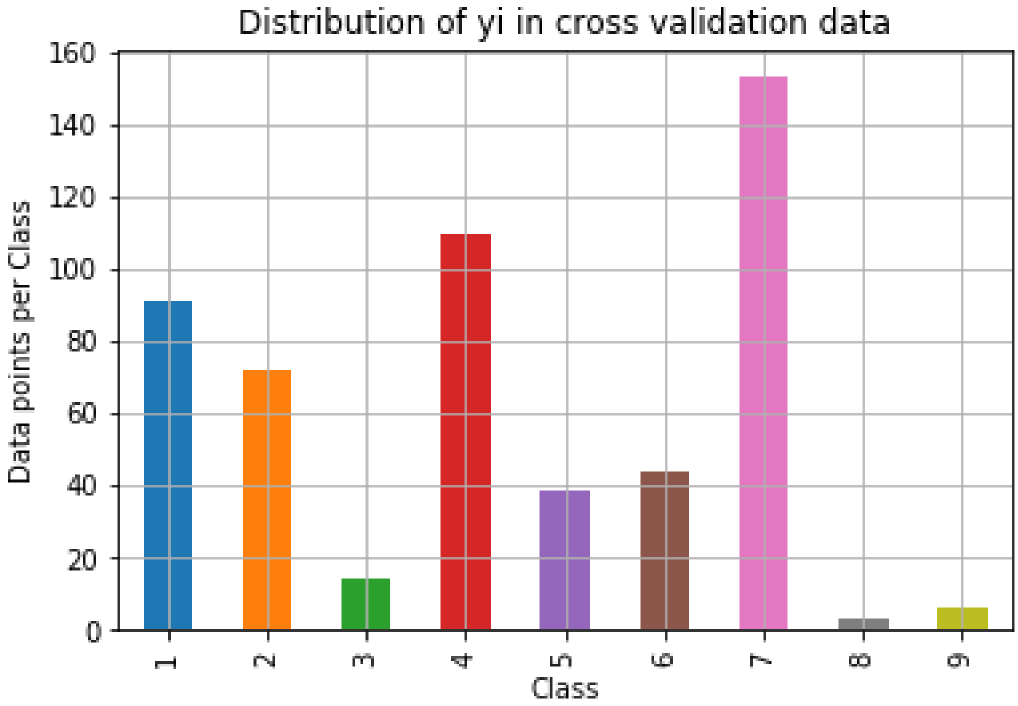


```
Number of data points in class 7 : 609 ( 28.672 %)
Number of data points in class 4 : 439 ( 20.669 %)
Number of data points in class 1 : 363 ( 17.09 %)
Number of data points in class 2 : 289 ( 13.606 %)
Number of data points in class 6 : 176 ( 8.286 %)
Number of data points in class 5 : 155 ( 7.298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

```
--------------------------------------------------------------------------------
```



```
Number of data points in class 7 : 191 ( 28.722 %)
Number of data points in class 4 : 137 ( 20.602 %)
Number of data points in class 1 : 114 ( 17.143 %)
Number of data points in class 2 : 91 ( 13.684 %)
Number of data points in class 6 : 55 ( 8.271 %)
Number of data points in class 5 : 48 ( 7.218 %)
Number of data points in class 3 : 18 ( 2.707 %)
Number of data points in class 9 : 7 ( 1.053 %)
Number of data points in class 8 : 4 ( 0.602 %)
--------------------------------------------------------------------------------
```



```
Number of data points in class 7 : 153 ( 28.759 %)
Number of data points in class 4 : 110 ( 20.677 %)
Number of data points in class 1 : 91 ( 17.105 %)
Number of data points in class 2 : 72 ( 13.534 %)
Number of data points in class 6 : 44 ( 8.271 %)
Number of data points in class 5 : 39 ( 7.331 %)
Number of data points in class 3 : 14 ( 2.632 %)
Number of data points in class 9 : 6 ( 1.128 %)
Number of data points in class 8 : 3 ( 0.564 %)
```

3.2 Prediction using a 'Random' Model

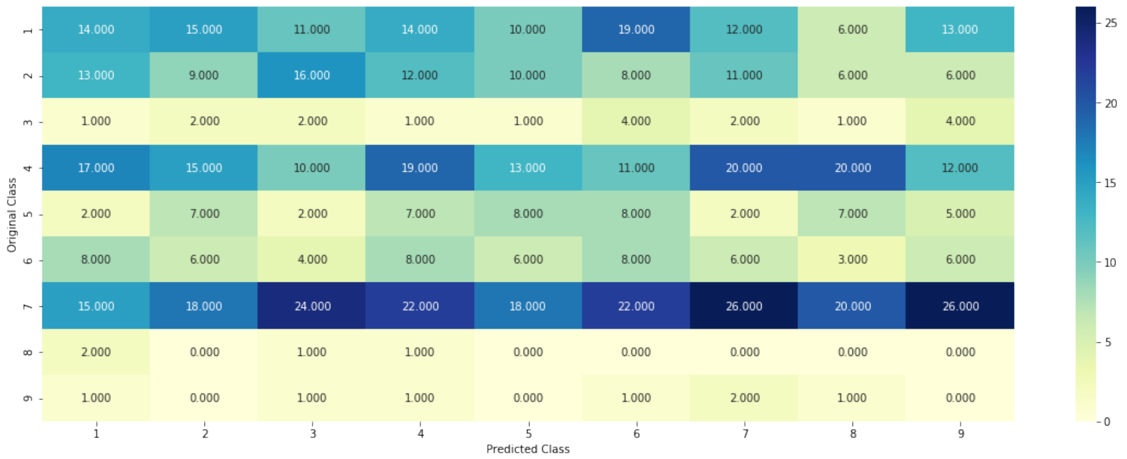
In a 'Random' Model, we generate the NINE class probabilites randomly such that they sum to 1.

```
In [34]: # 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 pred

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

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

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

             B =(C/C.sum(axis=0))
             #divid each element of the confusion matrix with the sum of elements in that row
             # C = [[1, 2],
             #     [3, 4]]
             # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in
             # 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, yticklabe
             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, yticklabe
             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, yticklabe
             plt.xlabel('Predicted Class')
             plt.ylabel('Original Class')
             plt.show()
```

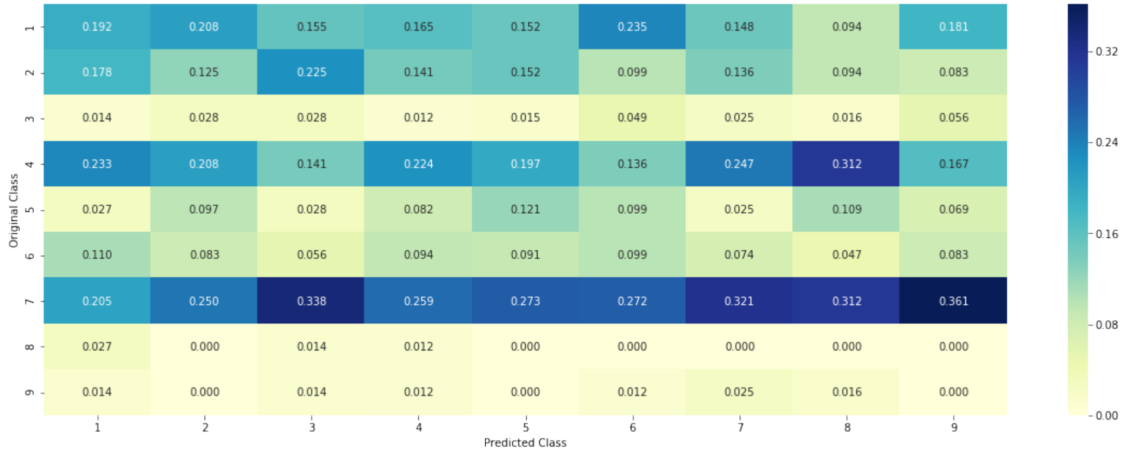
```
In [14]: # we need to generate 9 numbers and the sum of numbers should be 1
         # one solution is to genarate 9 numbers and divide each of the numbers by their sum
         # ref: https://stackoverflow.com/a/18662466/4084039
```

```
test_data_len = test_df.shape[0]
cv_data_len = cv_df.shape[0]

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


# 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

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

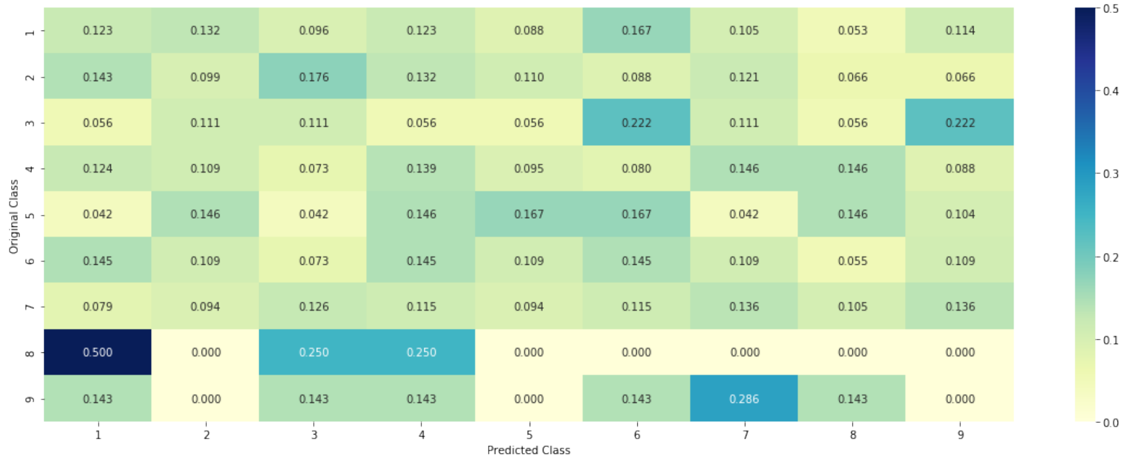
```
Log loss on Cross Validation Data using Random Model 2.4213037900881265
Log loss on Test Data using Random Model 2.4422802991172134
-------------------- Confusion matrix --------------------
```



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



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



3.3 Univariate Analysis

```
In [39]: # code for response coding with Laplace smoothing.
         # alpha : used for laplace smoothing
         # feature: ['gene', 'variation']
         # df: ['train_df', 'test_df', 'cv_df']
         # algorithm
         # ----------
         # Consider all unique values and the number of occurances of given feature in train d
         # build a vector (1*9) , the first element = (number of times it occured in class1 + 
         # gv_dict is like a look up table, for every gene it store a (1*9) representation of
         # for a value of feature in df:
         # if it is in train data:
```

```
# we add the vector that was stored in 'gv_dict' look up table to 'gv_fea'
# if it is not there is train:
# we add [1/9, 1/9, 1/9, 1/9,1/9, 1/9, 1/9, 1/9, 1/9] to 'gv_fea'
# return 'gv_fea'
# ----------------------

# get_gv_fea_dict: Get Gene varaition Feature Dict
def get_gv_fea_dict(alpha, feature, df):
    # value_count: it contains a dict like
    # print(train_df['Gene'].value_counts())
    # output:
    #        {BRCA1      174
    #         TP53       106
    #         EGFR        86
    #         BRCA2       75
    #         PTEN        69
    #         KIT         61
    #         BRAF        60
    #         ERBB2       47
    #         PDGFRA      46
    #         ...}
    # print(train_df['Variation'].value_counts())
    # output:
    # {
    # Truncating_Mutations                     63
    # Deletion                                 43
    # Amplification                            43
    # Fusions                                  22
    # Overexpression                            3
    # E17K                                      3
    # Q61L                                      3
    # S222D                                     2
    # P130S                                     2
    # ...
    # }
    value_count = train_df[feature].value_counts()

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

    # denominator will contain the number of time that particular feature occured in t
    for i, denominator in value_count.items():
        # vec will contain (p(yi==1/Gi) probability of gene/variation belongs to pert
        # vec is 9 diamensional vector
        vec = []
        for k in range(1,10):
            # print(train_df.loc[(train_df['Class']==1) & (train_df['Gene']=='BRCA1')
            #         ID   Gene              Variation  Class
            # 2470  2470  BRCA1                S1715C      1
            # 2486  2486  BRCA1                S1841R      1
            # 2614  2614  BRCA1                   M1R      1
            # 2432  2432  BRCA1                L1657P      1
            # 2567  2567  BRCA1                T1685A      1
            # 2583  2583  BRCA1                E1660G      1
            # 2634  2634  BRCA1                W1718L      1
            # cls_cnt.shape[0] will return the number of rows

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

            # cls_cnt.shape[0](numerator) will contain the number of time that partic
            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, 0.06818181818181818177, 
    #      'TP53': [0.32142857142857145, 0.061224489795918366, 0.061224489795918366, 
    #      'EGFR': [0.056818181818181816, 0.21590909090909091, 0.0625, 0.06818181818181
    #      'BRCA2': [0.13333333333333333, 0.060606060606060608, 0.060606060606060608, 
    #      'PTEN': [0.069182389937106917, 0.062893081761006289, 0.069182389937106917, 
    #      'KIT': [0.066225165562913912, 0.25165562913907286, 0.072847682119205295, 0
    #      'BRAF': [0.066666666666666666, 0.17999999999999999, 0.073333333333333334, 
    #      ...
    #     }
    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 va
    gv_fea = []
    # for every feature values in the given data frame we will check if it is there i
    # 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

(numerator + 10*alpha) / (denominator + 90*alpha)
3.2.1 Univariate Analysis on Gene Feature
Q1. Gene, What type of feature it is ?
Ans. Gene is a categorical variable
Q2. How many categories are there and How they are distributed?

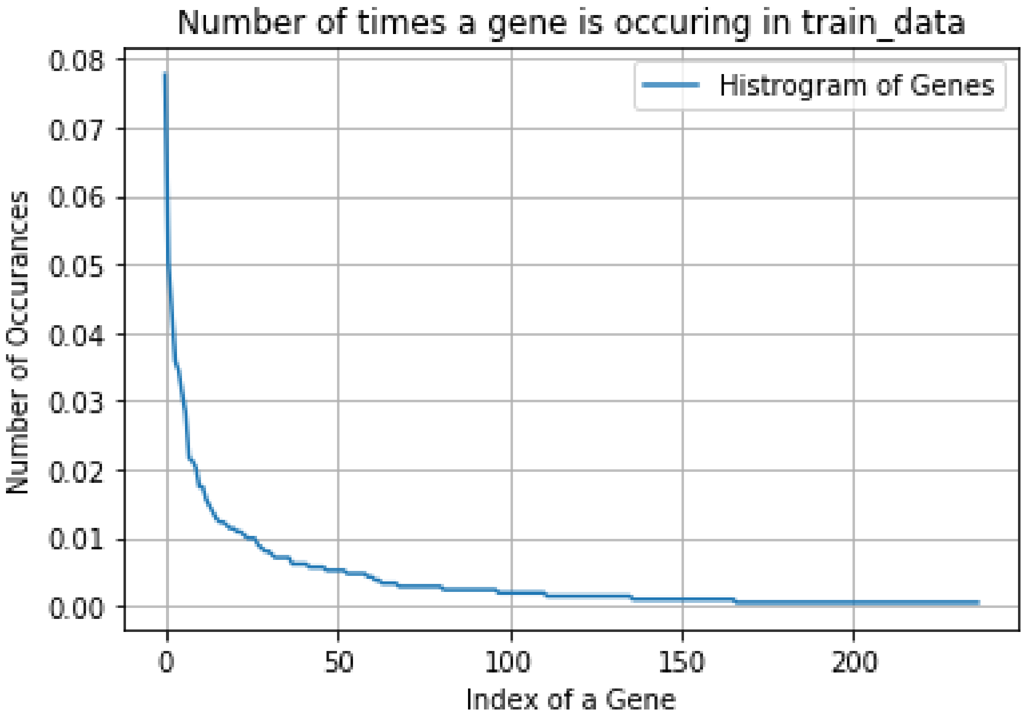
```
In [16]: unique_genes = train_df['Gene'].value_counts()
         print('Number of Unique Genes :', unique_genes.shape[0])
         # the top 10 genes that occured most
         print(unique_genes.head(10))
```

```
Number of Unique Genes : 237
BRCA1     165
TP53      107
EGFR       91
PTEN       76
BRCA2      73
KIT        66
BRAF       60
ALK        46
ERBB2      45
CDKN2A     43
Name: Gene, dtype: int64
```

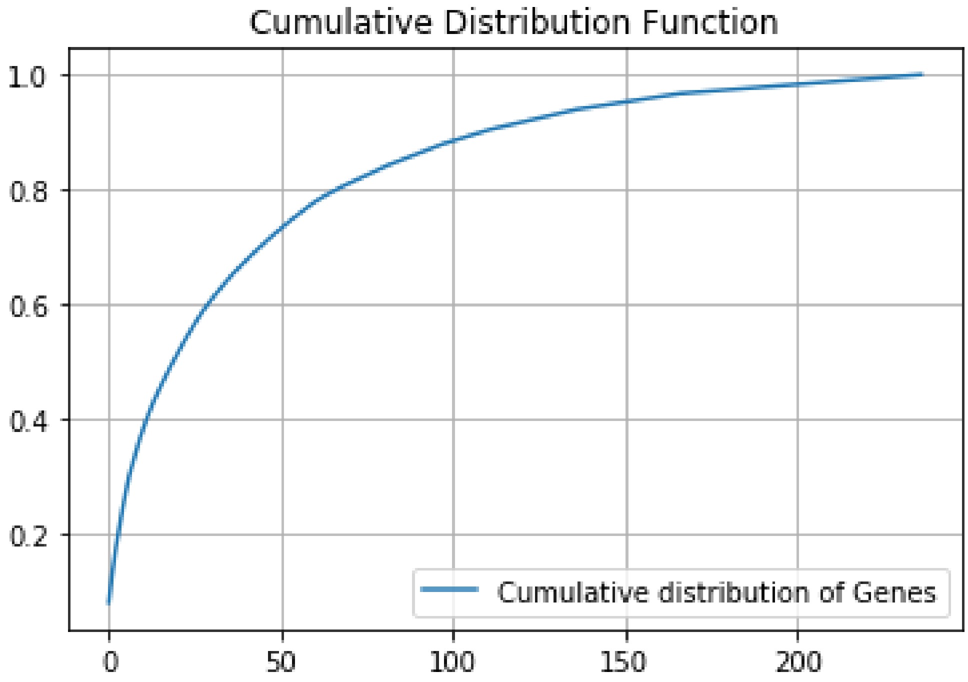
```
In [17]: print("Ans: There are", unique_genes.shape[0] ,"different categories of genes in the
```

```
Ans: There are 237 different categories of genes in the train data, and they are distibuted as
```

```
In [18]: s = sum(unique_genes.values);
         h = unique_genes.values/s;
         plt.plot(h, label="Histrogram of Genes")
         plt.xlabel('Index of a Gene')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.title("Number of times a gene is occuring in train_data")
         plt.grid()
         plt.show()
```



```
In [19]: c = np.cumsum(h)
         plt.plot(c,label='Cumulative distribution of Genes')
         plt.grid()
         plt.title("Cumulative Distribution Function")
         plt.legend()
         plt.show()
```



Q3. How to featurize this Gene feature ?

Ans.there are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

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 [40]: #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 [41]: print("train_gene_feature_responseCoding is converted feature using respone coding me
```

```
train_gene_feature_responseCoding is converted feature using respone coding method. The shape o
```

```
In [43]: # 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 [23]: train_df['Gene'].head()
```

```
Out[23]: 3097    NOTCH2
         2908       NF2
         2948       KDR
         3222     NTRK1
         1190    PIK3CA
         Name: Gene, dtype: object
```

```
In [24]: gene_vectorizer.get_feature_names()
```

```
Out[24]: ['abl1',
          'acvr1',
          'ago2',
          'akt1',
          'akt2',
          'akt3',
          'alk',
          'apc',
          'ar',
          'araf',
          'arid1a',
          'arid1b',
          'arid2',
          'arid5b',
          'asxl2',
          'atm',
          'atr',
          'atrx',
          'aurka',
          'aurkb',
          'axin1',
          'axl',
          'b2m',
          'bap1',
          'bcl10',
          'bcl2l11',
          'bcor',
          'braf',
          'brca1',
          'brca2',
          'brd4',
          'brip1',
          'btk',
          'card11',
'carm1',
'casp8',
'cbl',
'ccnd1',
'ccnd3',
'cdh1',
'cdk12',
'cdk4',
'cdk6',
'cdk8',
'cdkn1a',
'cdkn1b',
'cdkn2a',
'cdkn2b',
'cdkn2c',
'cebpa',
'chek2',
'cic',
'crebbp',
'ctcf',
'ctla4',
'ctnnb1',
'ddr2',
'dicer1',
'dnmt3a',
'dnmt3b',
'dusp4',
'egfr',
'eif1ax',
'elf3',
'ep300',
'epas1',
'epcam',
'erbb2',
'erbb3',
'erbb4',
'ercc2',
'ercc3',
'ercc4',
'erg',
'esr1',
'etv1',
'etv6',
'ewsr1',
'ezh2',
'fanca',
'fancc',
'fat1',
    'fbxw7',
    'fgf3',
    'fgf4',
    'fgfr1',
    'fgfr2',
    'fgfr3',
    'fgfr4',
    'flt1',
    'flt3',
    'foxa1',
    'foxl2',
    'foxo1',
    'foxp1',
    'fubp1',
    'gata3',
    'gli1',
    'gna11',
    'gnaq',
    'gnas',
    'h3f3a',
    'hist1h1c',
    'hla',
    'hnf1a',
    'hras',
    'idh1',
    'idh2',
    'igf1r',
    'ikzf1',
    'jak1',
    'jak2',
    'jun',
    'kdm5a',
    'kdm5c',
    'kdr',
    'keap1',
    'kit',
    'klf4',
    'kmt2a',
    'kmt2c',
    'kmt2d',
    'knstrn',
    'kras',
    'lats1',
    'lats2',
    'map2k1',
    'map2k2',
    'map2k4',
    'map3k1',
'mapk1',
'mdm2',
'med12',
'mef2b',
'met',
'mga',
'mlh1',
'mpl',
'msh2',
'msh6',
'mtor',
'myc',
'myd88',
'myod1',
'ncor1',
'nf1',
'nf2',
'nfe2l2',
'nfkbia',
'nkx2',
'notch1',
'notch2',
'npm1',
'nras',
'nsd1',
'ntrk1',
'ntrk2',
'ntrk3',
'nup93',
'pak1',
'pbrm1',
'pdgfra',
'pdgfrb',
'pik3ca',
'pik3cb',
'pik3cd',
'pik3r1',
'pik3r2',
'pim1',
'pms1',
'pms2',
'pole',
'ppm1d',
'ppp2r1a',
'ppp6c',
'prdm1',
'ptch1',
'pten',
'ptpn11',
'ptprd',
'ptprt',
'rac1',
'rad50',
'rad51b',
'rad51c',
'rad51d',
'rad54l',
'raf1',
'rara',
'rasa1',
'rb1',
'rbm10',
'ret',
'rheb',
'rhoa',
'rit1',
'rnf43',
'ros1',
'runx1',
'rxra',
'sdhb',
'sdhc',
'setd2',
'sf3b1',
'shq1',
'smad2',
'smad3',
'smad4',
'smarca4',
'smarcb1',
'smo',
'sos1',
'sox9',
'spop',
'src',
'srsf2',
'stag2',
'stat3',
'stk11',
'tcf7l2',
'tert',
'tet1',
'tet2',
'tgfbr1',
'tgfbr2',
'tmprss2',
 'tp53',
 'tp53bp1',
 'tsc1',
 'tsc2',
 'u2af1',
 'vhl',
 'whsc1',
 'whsc1l1',
 'xpo1',
 'xrcc2',
 'yap1']
```

```
In [25]: print("train_gene_feature_onehotCoding is converted feature using one-hot encoding me
```

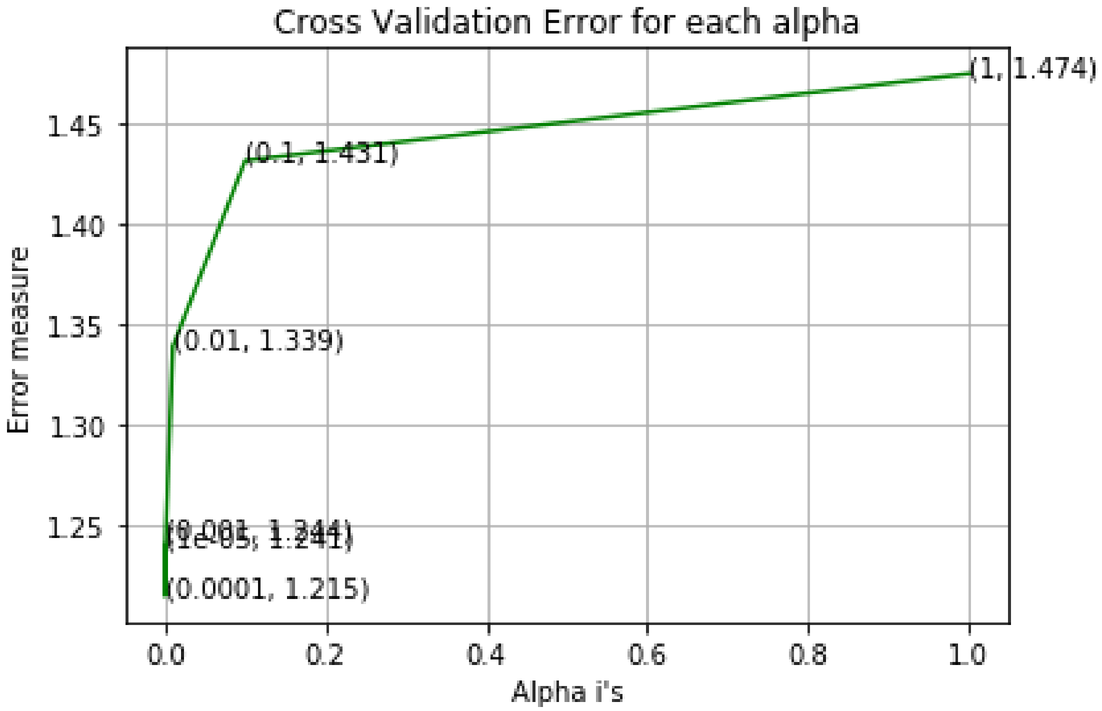
```
train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape 
```

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 [26]: 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
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # 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-1
             print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, l
         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=4
         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_l
         predict_y = sig_clf.predict_proba(cv_gene_feature_onehotCoding)
         print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
         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_l
```

```
For values of alpha =  1e-05 The log loss is: 1.2405402706963011
For values of alpha =  0.0001 The log loss is: 1.215265806779627
For values of alpha =  0.001 The log loss is: 1.2444376661030854
For values of alpha =  0.01 The log loss is: 1.3390511385635417
For values of alpha =  0.1 The log loss is: 1.431226823040672
For values of alpha =  1 The log loss is: 1.4743358345664523
```



```
For values of best alpha =  0.0001 The train log loss is: 0.9857441557176742
For values of best alpha =  0.0001 The cross validation log loss is: 1.215265806779627
For values of best alpha =  0.0001 The test log loss is: 1.1730120275298253
```

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 [27]: print("Q6. How many data points in Test and CV datasets are covered by the ", unique_

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

```
Q6. How many data points in Test and CV datasets are covered by the  237  genes in train datase
Ans
1. In test data 648 out of 665 : 97.44360902255639
2. In cross validation data 517 out of  532 : 97.18045112781954
```

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?

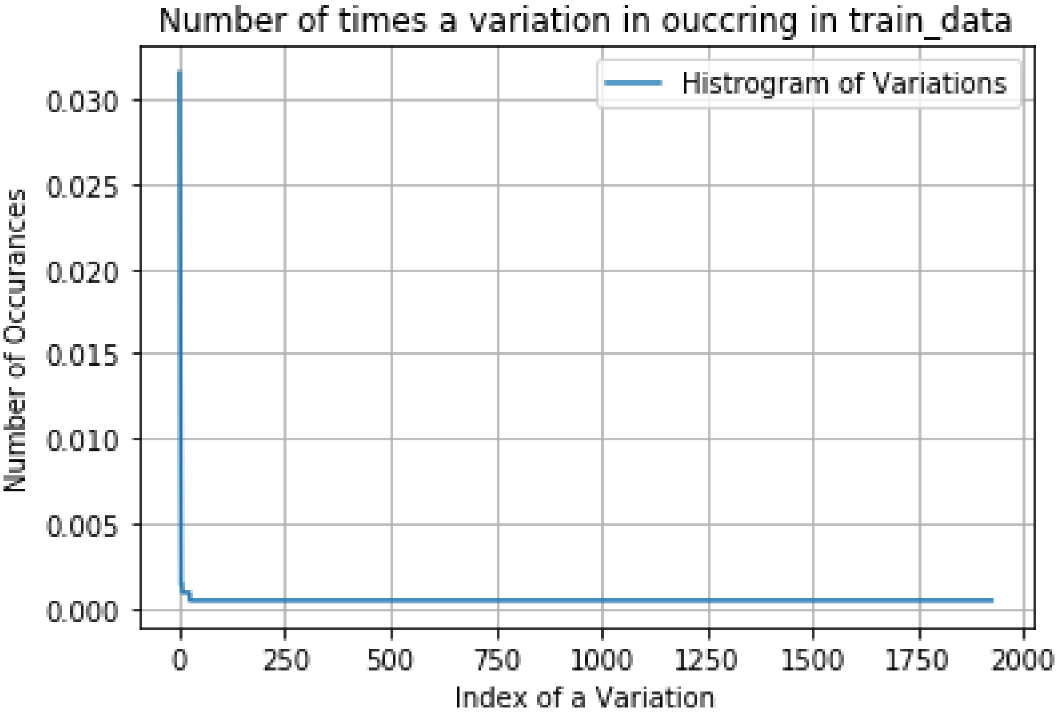
```
In [28]: unique_variations = train_df['Variation'].value_counts()
         print('Number of Unique Variations :', unique_variations.shape[0])
         # the top 10 variations that occured most
         print(unique_variations.head(10))
```

```
Number of Unique Variations : 1925
Truncating_Mutations    67
Deletion                49
Amplification           42
Fusions                 21
Overexpression           3
Q61L                     3
G12V                     3
Q209L                    2
Y64A                     2
E330K                    2
Name: Variation, dtype: int64
```

```
In [29]: print("Ans: There are", unique_variations.shape[0] ,"different categories of variatio
```

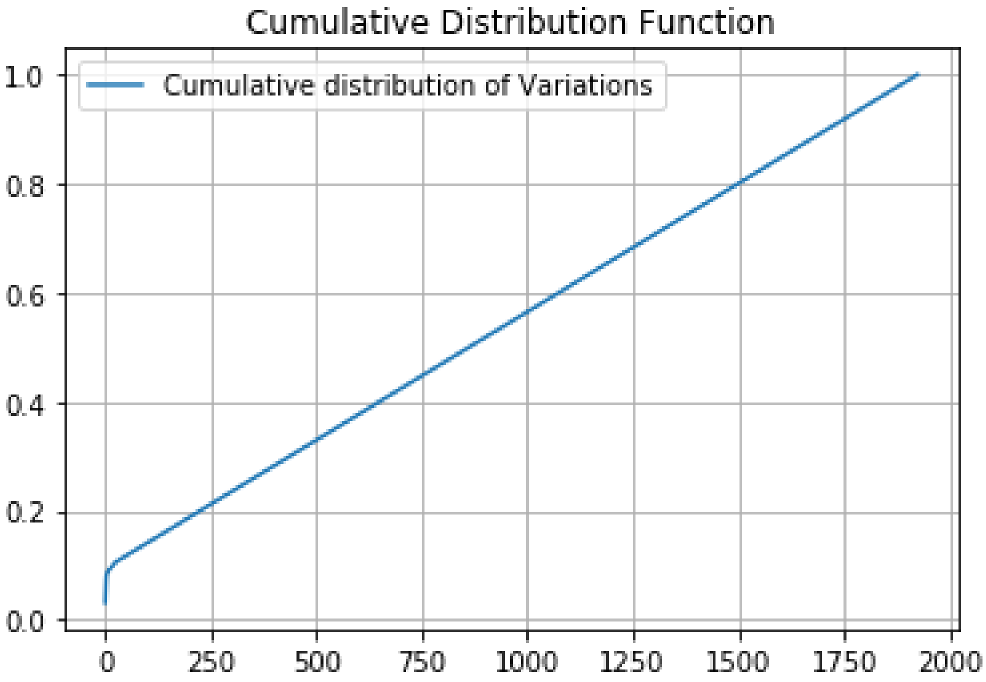
```
Ans: There are 1925 different categories of variations in the train data, and they are distibu
```

```
In [30]: s = sum(unique_variations.values);
         h = unique_variations.values/s;
         plt.plot(h, label="Histrogram of Variations")
         plt.xlabel('Index of a Variation')
         plt.ylabel('Number of Occurances')
         plt.legend()
         plt.title("Number of times a variation in ouccring in train_data")
         plt.grid()
         plt.show()
```



```
In [31]: c = np.cumsum(h)
         print(c)
         plt.plot(c,label='Cumulative distribution of Variations')
         plt.grid()
         plt.legend()
         plt.title("Cumulative Distribution Function")
         plt.show()
```

```
[0.03154426 0.05461394 0.07438795 ... 0.99905838 0.99952919 1.        ]
```



Q9. How to featurize this Variation feature ?

Ans.There are two ways we can featurize this variable check out this video: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/

One hot Encoding

Response coding

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

```
In [50]: # alpha is used for laplace smoothing
         alpha = 1
         # train gene feature
         train_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", t
         # test gene feature
         test_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", te
         # cross validation gene feature
         cv_variation_feature_responseCoding = np.array(get_gv_feature(alpha, "Variation", cv_d
```

```
In [51]: print("train_variation_feature_responseCoding is a converted feature using the respons
```

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

```
In [52]: # one-hot encoding of variation feature.
         variation_vectorizer = CountVectorizer()
         train_variation_feature_onehotCoding = variation_vectorizer.fit_transform(train_df['Va
         test_variation_feature_onehotCoding = variation_vectorizer.transform(test_df['Variatio
         cv_variation_feature_onehotCoding = variation_vectorizer.transform(cv_df['Variation'])
```

```
In [53]: print("train_variation_feature_onehotEncoded is converted feature using the onne-hot
```

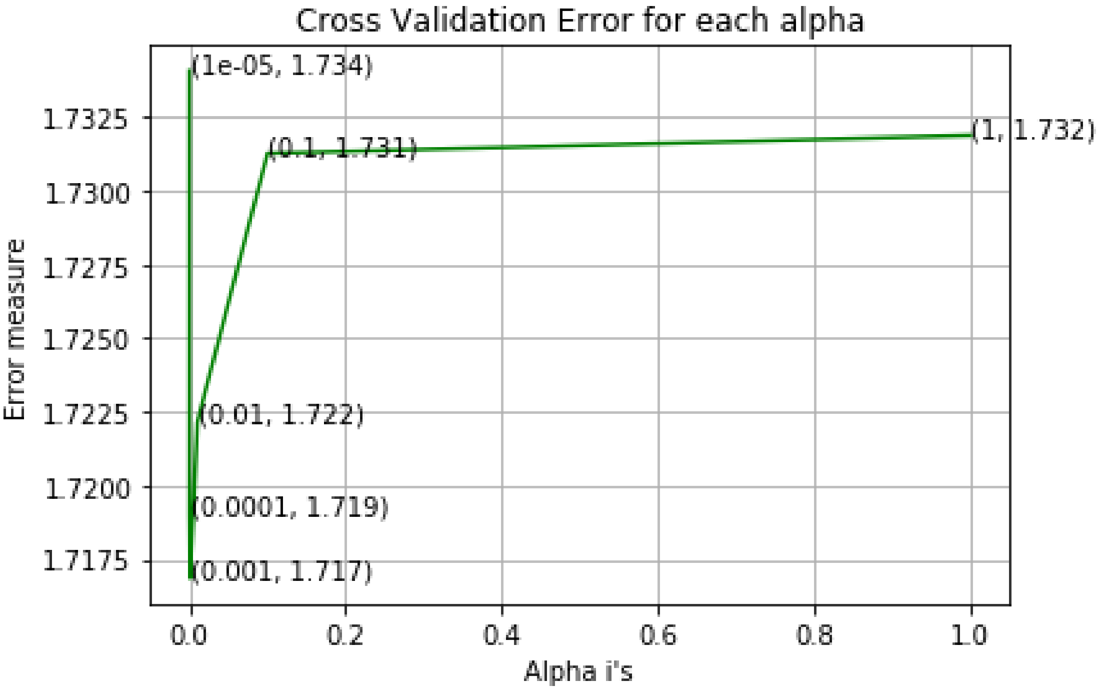
```
train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method.
```

Q10. How good is this Variation feature in predicting y_i?
Let's build a model just like the earlier!

```
In [36]: alpha = [10 ** x for x in range(-5, 1)]

         # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated
         # ------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # 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-1
             print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y, la

         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=4
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_l
predict_y = sig_clf.predict_proba(cv_variation_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log los
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_lo
```

```
For values of alpha =  1e-05 The log loss is: 1.7340741565161613
For values of alpha =  0.0001 The log loss is: 1.7190323097960478
For values of alpha =  0.001 The log loss is: 1.7168671355128318
For values of alpha =  0.01 The log loss is: 1.7221573740572267
For values of alpha =  0.1 The log loss is: 1.7312650289768132
For values of alpha =  1 The log loss is: 1.7318809042776733
```



```
For values of best alpha =  0.001 The train log loss is: 1.113346148145461
For values of best alpha =  0.001 The cross validation log loss is: 1.7168671355128318
For values of best alpha =  0.001 The test log loss is: 1.7209382557584394
```

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 [37]: print("Q12. How many data points are covered by total ", unique_variations.shape[0], '
         test_coverage=test_df[test_df['Variation'].isin(list(set(train_df['Variation'])))].sha
         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_cove
         print('2. In cross validation data',cv_coverage, 'out of ',cv_df.shape[0],":" ,(cv_co
```

```
Q12. How many data points are covered by total  1925  genes in test and cross validation data s
Ans
1. In test data 66 out of 665 : 9.924812030075188
2. In cross validation data 53 out of  532 : 9.962406015037594
```

3.2.3 Univariate Analysis on Text Feature

1. How many unique words are present in train data?
2. How are word frequencies distributed?
3. How to featurize text field?
4. Is the text feature useful in predicitng y_i?
5. Is the text feature stable across train, test and CV datasets?

```
In [63]: # 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 [59]: 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(
                     text_feature_responseCoding[row_index][i] = math.exp(sum_prob/len(row['TE
                     row_index += 1
             return text_feature_responseCoding
```

```
In [137]: # building a CountVectorizer with all the words that occured minimum 3 times in trai
          # only for Logistic Regression, as with TfidfVectorizer Logistic Regression is not p
          text_vectorizer = CountVectorizer(min_df=10, ngram_range=(1,4))
          train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
          # getting all the feature names (words)
          train_text_features= text_vectorizer.get_feature_names()

          # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*n
          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
          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 : 566304
```

```
In [64]: dict_list = []
         # dict_list =[] contains 9 dictoinaries each corresponds to a class
         for i in range(1,10):
             cls_text = train_df[train_df['Class']==i]
             # build a word dict based on the words in that class
             dict_list.append(extract_dictionary_paddle(cls_text))
             # append it to dict_list

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


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

```
In [65]: #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 [66]: # 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_fe
         test_text_feature_responseCoding = (test_text_feature_responseCoding.T/test_text_featu
         cv_text_feature_responseCoding = (cv_text_feature_responseCoding.T/cv_text_feature_res
```

```
In [67]: # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
In [68]: #https://stackoverflow.com/a/2258273/4084039
         sorted_text_fea_dict = dict(sorted(text_fea_dict.items(), key=lambda x: x[1] , reverse
         sorted_text_occur = np.array(list(sorted_text_fea_dict.values()))
```

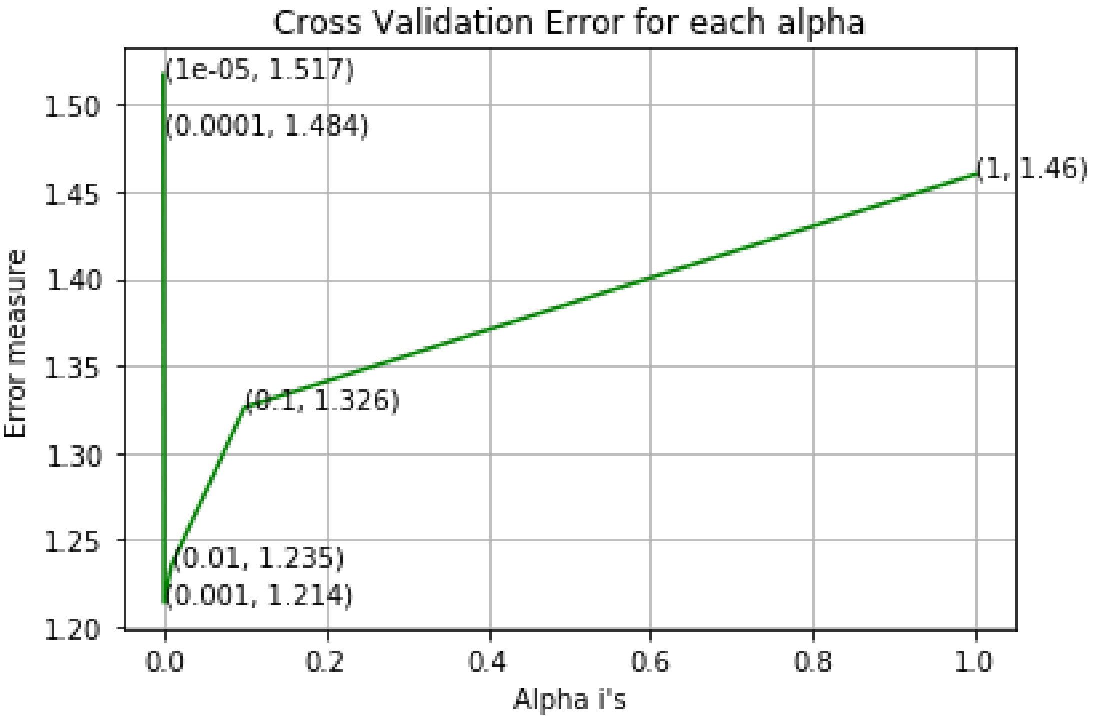
```
In [69]: # Number of words for a given frequency.
         print(Counter(sorted_text_occur))
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

```
In [144]: # Train a Logistic regression+Calibration model using text features whicha re on-hot
          alpha = [10 ** x for x in range(-5, 1)]

          # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # 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-
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_cv, predict_y,

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=
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_
predict_y = sig_clf.predict_proba(cv_text_feature_onehotCoding)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log lo
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_
```

```
For values of alpha =  1e-05 The log loss is: 1.5171638531663776
For values of alpha =  0.0001 The log loss is: 1.4837121247698222
For values of alpha =  0.001 The log loss is: 1.2137737022862025
For values of alpha =  0.01 The log loss is: 1.2352027839760191
For values of alpha =  0.1 The log loss is: 1.3258864827256527
For values of alpha =  1 The log loss is: 1.4600327385737681
```



```
For values of best alpha =  0.001 The train log loss is: 0.866779980295729
For values of best alpha =  0.001 The cross validation log loss is: 1.2137737022862025
For values of best alpha =  0.001 The test log loss is: 1.1902902372181423
```

Q. Is the Text feature stable across all the data sets (Test, Train, Cross validation)?
Ans. Yes, it seems like!

```
In [145]: def get_intersec_text(df):
              df_text_vec = CountVectorizer(min_df=10, max_features=5000)
              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 [146]: 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
99.94 % of word of test data appeared in train data
100.0 % of word of Cross Validation appeared in train data
```

4. Machine Learning Models

```
In [74]: #Data preparation for ML models.

         #Misc. functionns for ML models


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

             # for calculating log_loss we willl provide the array of probabilities belongs to
             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))/tes
             plot_confusion_matrix(test_y, pred_y)

In [75]: 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 [76]: # this function will be used just for naive bayes
         # for the given indices, we will print the name of the features
         # and we will check whether the feature present in the test point text or not
         def get_impfeature_names(indices, text, gene, var, no_features):
             gene_count_vec = CountVectorizer()
             var_count_vec = CountVectorizer()
             text_count_vec = CountVectorizer(min_df=3)

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

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

             word_present = 0
             for i,v in enumerate(indices):
                 if (v < fea1_len):
                     word = gene_vec.get_feature_names()[v]
```

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

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

Stacking the three types of features

```
In [87]: # 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_
         test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
         cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_o

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

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

         cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).t
         cv_y = np.array(list(cv_df['Class']))


         train_gene_var_responseCoding = np.hstack((train_gene_feature_responseCoding,train_var
         test_gene_var_responseCoding = np.hstack((test_gene_feature_responseCoding,test_variat
         cv_gene_var_responseCoding = np.hstack((cv_gene_feature_responseCoding,cv_variation_fe
         train_x_responseCoding = np.hstack((train_gene_var_responseCoding, train_text_feature_
         test_x_responseCoding = np.hstack((test_gene_var_responseCoding, test_text_feature_res
         cv_x_responseCoding = np.hstack((cv_gene_var_responseCoding, cv_text_feature_response(
```

```
In [151]: print("One hot encoding features :")
          print("(number of data points * number of features) in train data = ", train_x_onehot
          print("(number of data points * number of features) in test data = ", test_x_onehotCo
          print("(number of data points * number of features) in cross validation data =", cv_x
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 568491)
(number of data points * number of features) in test data =  (665, 568491)
(number of data points * number of features) in cross validation data = (532, 568491)
```

```
In [152]: print(" Response encoding features :")
          print("(number of data points * number of features) in train data = ", train_x_respon
          print("(number of data points * number of features) in test data = ", test_x_response
          print("(number of data points * number of features) in cross validation data =", cv_x
```

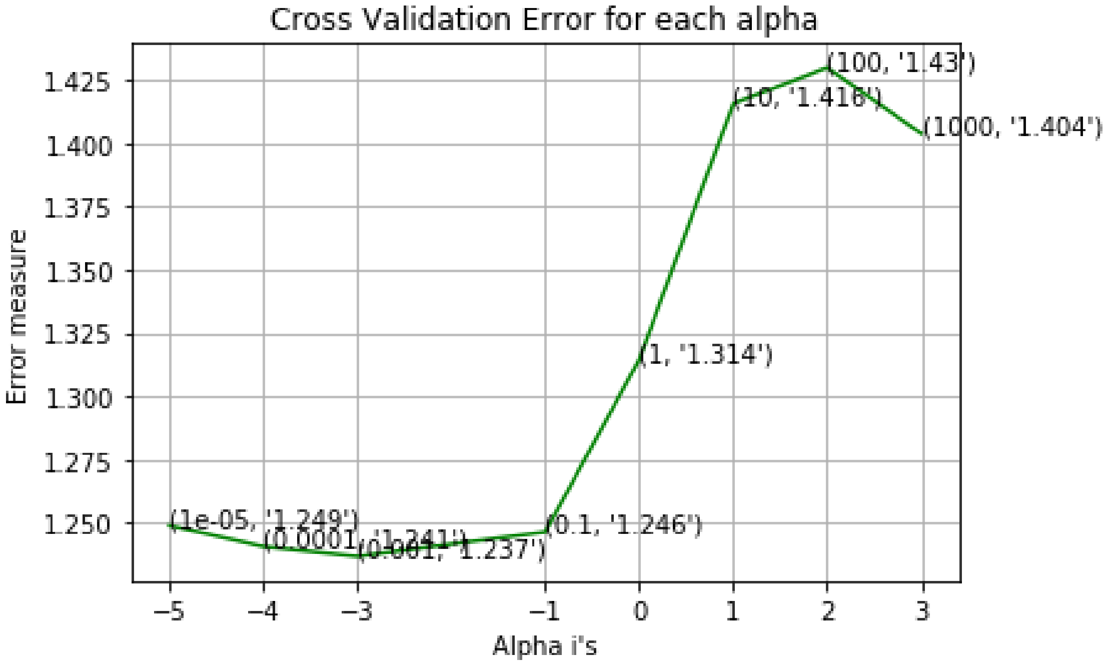
```
 Response encoding features :
(number of data points * number of features) in train data =  (2124, 27)
(number of data points * number of features) in test data =  (665, 27)
(number of data points * number of features) in cross validation data = (532, 27)
```

4.1. Base Line Model
4.1.1. Naive Bayes
4.1.1.1. Hyper parameter tuning

```
In [211]: # find more about Multinomial Naive base function here http://scikit-learn.org/stabl
          # -------------------------
          # 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/lesson
          # -----------------------


          # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
          # ----------------------------
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
          #
          # 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/lesson
          # -----------------------


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

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


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


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

```
          print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
```

```
for alpha = 1e-05
Log Loss : 1.248914701891545
for alpha = 0.0001
Log Loss : 1.240582399570973
for alpha = 0.001
Log Loss : 1.2367232633948446
for alpha = 0.1
Log Loss : 1.2463209817606011
for alpha = 1
Log Loss : 1.314472085390315
for alpha = 10
Log Loss : 1.4156677042183454
for alpha = 100
Log Loss : 1.4299012601700911
for alpha = 1000
Log Loss : 1.403889929135179
```

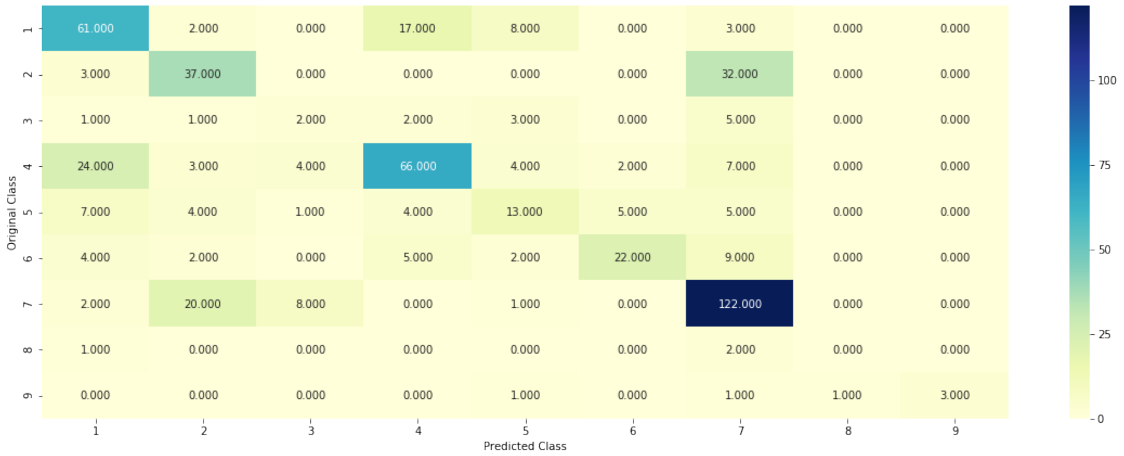


```
For values of best alpha =  0.001 The train log loss is: 0.7509116757627415
For values of best alpha =  0.001 The cross validation log loss is: 1.2367232633948446
For values of best alpha =  0.001 The test log loss is: 1.2680603221795845
```

4.1.1.2. Testing the model with best hyper paramters

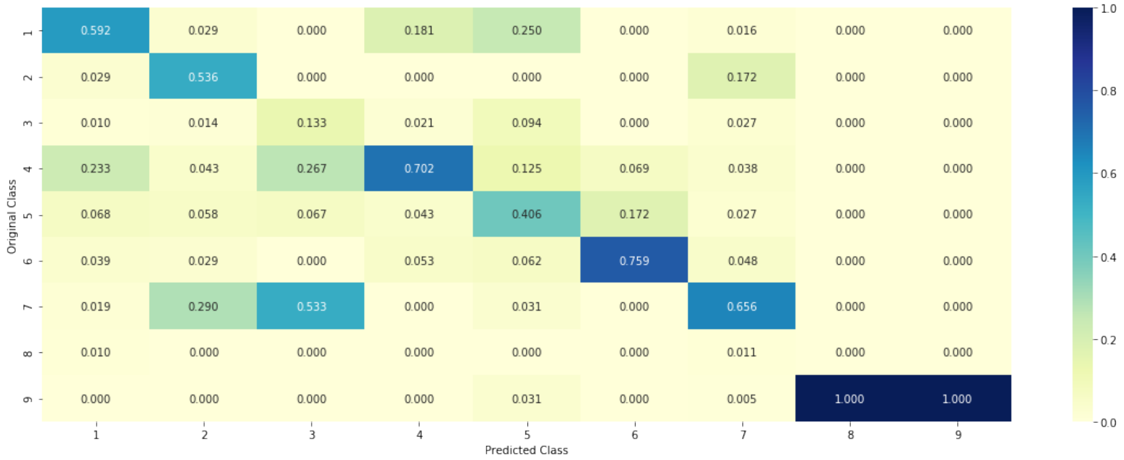
```
In [212]: # find more about Multinomial Naive base function here http://scikit-learn.org/stabl
          # -------------------------
          # default paramters
          # sklearn.naive_bayes.MultinomialNB(alpha=1.0, fit_prior=True, class_prior=None)

          # 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/lesson
          # -----------------------


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

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

```
Log Loss : 1.2367232633948446
Number of missclassified point : 0.38721804511278196
-------------------- Confusion matrix --------------------
```



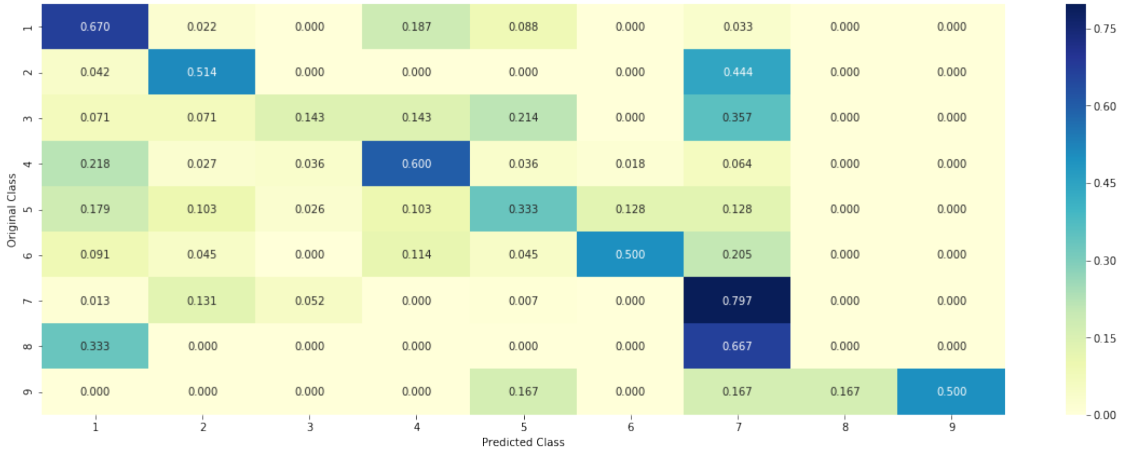
	1	2	3	4	5	6	7	8	9
1	61.000	2.000	0.000	17.000	8.000	0.000	3.000	0.000	0.000
2	3.000	37.000	0.000	0.000	0.000	0.000	32.000	0.000	0.000
3	1.000	1.000	2.000	2.000	3.000	0.000	5.000	0.000	0.000
4	24.000	3.000	4.000	66.000	4.000	2.000	7.000	0.000	0.000
5	7.000	4.000	1.000	4.000	13.000	5.000	5.000	0.000	0.000
6	4.000	2.000	0.000	5.000	2.000	22.000	9.000	0.000	0.000
7	2.000	20.000	8.000	0.000	1.000	0.000	122.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000
9	0.000	0.000	0.000	0.000	1.000	0.000	1.000	1.000	3.000

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



	1	2	3	4	5	6	7	8	9
1	0.592	0.029	0.000	0.181	0.250	0.000	0.016	0.000	0.000
2	0.029	0.536	0.000	0.000	0.000	0.000	0.172	0.000	0.000
3	0.010	0.014	0.133	0.021	0.094	0.000	0.027	0.000	0.000
4	0.233	0.043	0.267	0.702	0.125	0.069	0.038	0.000	0.000
5	0.068	0.058	0.067	0.043	0.406	0.172	0.027	0.000	0.000
6	0.039	0.029	0.000	0.053	0.062	0.759	0.048	0.000	0.000
7	0.019	0.290	0.533	0.000	0.031	0.000	0.656	0.000	0.000
8	0.010	0.000	0.000	0.000	0.000	0.000	0.011	0.000	0.000
9	0.000	0.000	0.000	0.000	0.031	0.000	0.005	1.000	1.000

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



4.1.1.3. Feature Importance, Correctly classified point

```
In [213]: test_point_index = 1
          no_feature = 1000
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehot
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1955 0.0653 0.0139 0.5442 0.042  0.0419 0.0912 0.004  0.002
Actual Class : 4
--------------------------------------------------
58 Text feature [22] present in test data point [True]
88 Text feature [601399] present in test data point [True]
89 Text feature [analysis] present in test data point [True]
109 Text feature [71] present in test data point [True]
115 Text feature [19] present in test data point [True]
116 Text feature [38] present in test data point [True]
142 Text feature [abnormal] present in test data point [True]
217 Text feature [19k] present in test data point [True]
299 Text feature [abolish] present in test data point [True]
301 Text feature [13] present in test data point [True]
338 Text feature [act] present in test data point [True]
516 Text feature [alleles] present in test data point [True]
609 Text feature [30] present in test data point [True]
642 Text feature [12] present in test data point [True]
704 Text feature [39] present in test data point [True]
902 Text feature [activation] present in test data point [True]
```

```
958 Text feature [109] present in test data point [True]
967 Text feature [42] present in test data point [True]
Out of the top  1000  features  18 are present in query point
```

4.1.1.4. Feature Importance, Incorrectly classified point

```
In [214]: test_point_index = 12
          no_feature = 100
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehot
          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['Gen
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.1092 0.1011 0.0229 0.1414 0.2479 0.0648 0.1465 0.1476 0.018
Actual Class : 7
--------------------------------------------------
Out of the top  100  features  0 are present in query point
```

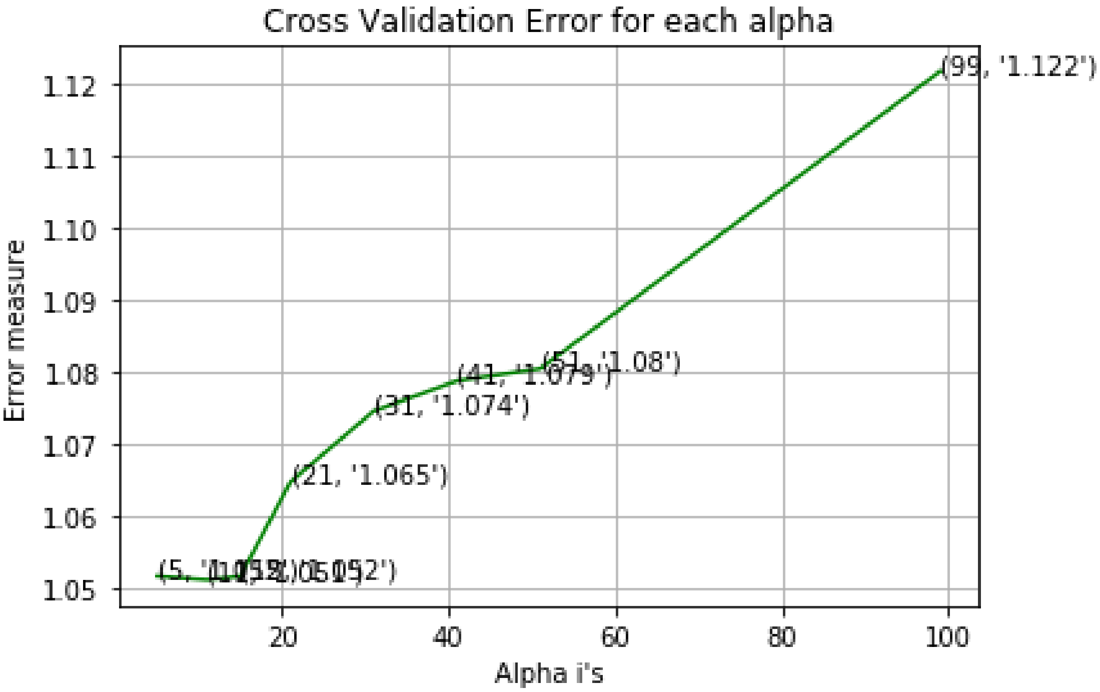
4.2. K Nearest Neighbour Classification

4.2.1. Hyper parameter tuning

```
In [135]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules
          # -------------------------
          # default parameter
          # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
          # 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/lesson


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


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

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


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

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

```
for alpha = 5
Log Loss : 1.0515319987984146
for alpha = 11
```

```
Log Loss : 1.0510055185329903
for alpha = 15
Log Loss : 1.0515250809230197
for alpha = 21
Log Loss : 1.064603570915532
for alpha = 31
Log Loss : 1.0744525807952168
for alpha = 41
Log Loss : 1.078641711475431
for alpha = 51
Log Loss : 1.0803414749199358
for alpha = 99
Log Loss : 1.1216169444807995
```



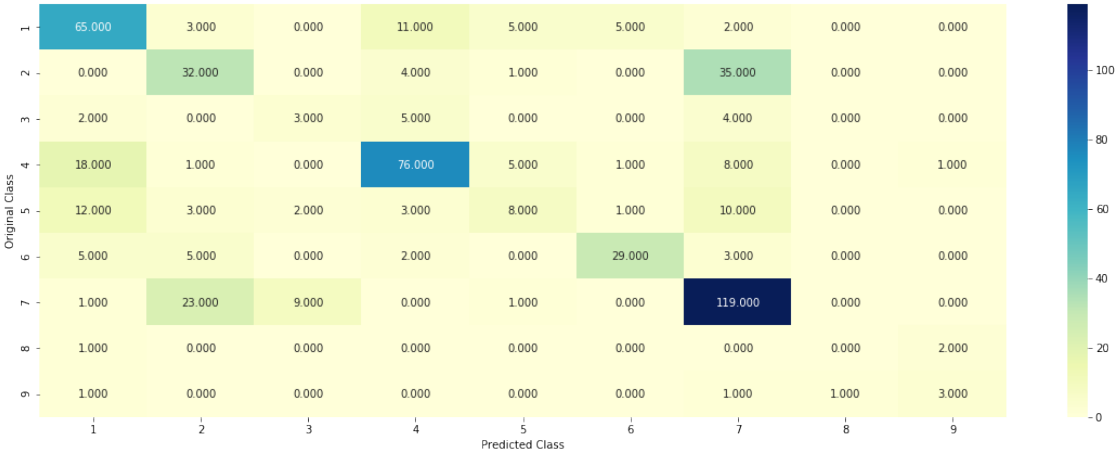
```
For values of best alpha =  11 The train log loss is: 0.6388451784939913
For values of best alpha =  11 The cross validation log loss is: 1.0510055185329903
For values of best alpha =  11 The test log loss is: 1.0873698890528134
```

4.2.2. Testing the model with best hyper paramters

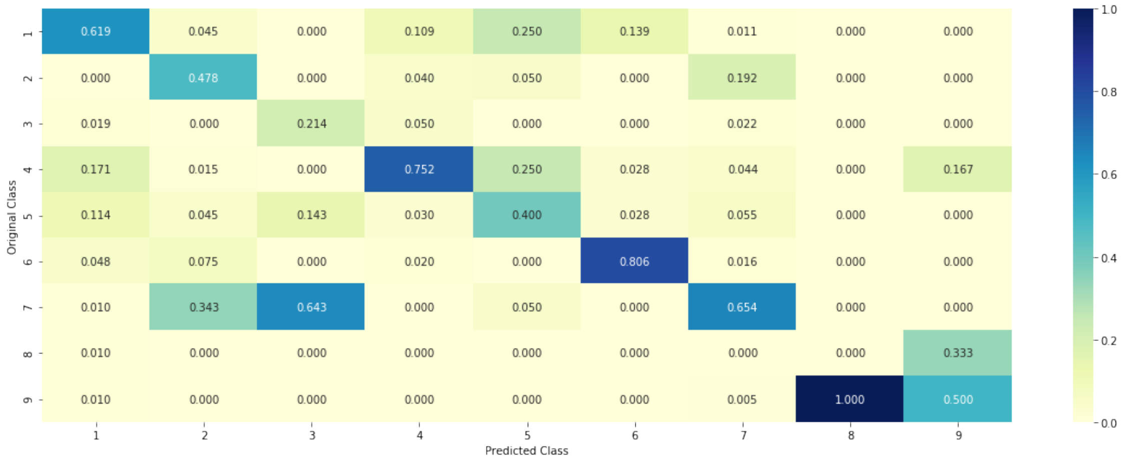
```
In [136]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
          # -------------------------
          # default parameter
```

```
        # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30,
        # 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/lesson
        #-------------------------------------
        clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
        predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodi
```

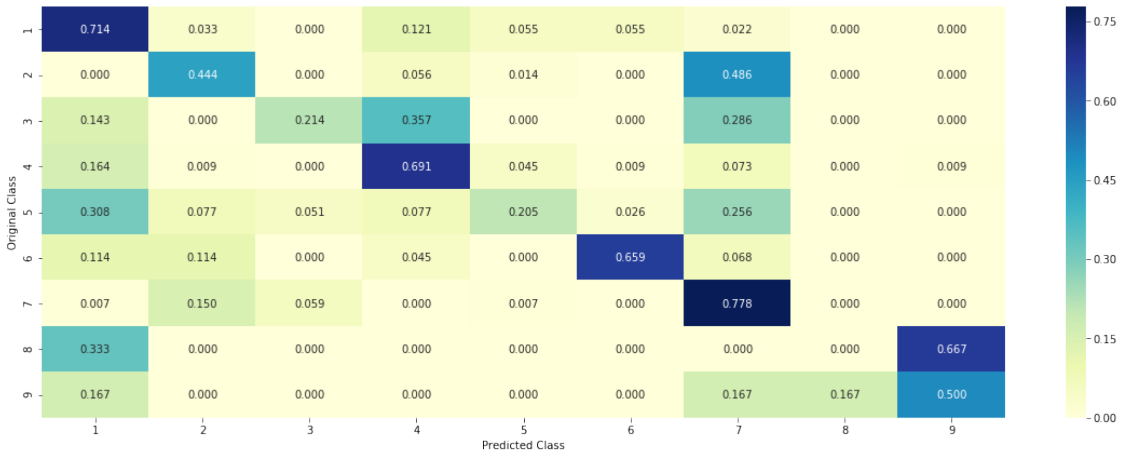
```
Log loss : 1.0510055185329903
Number of mis-classified points : 0.37030075187969924
-------------------- Confusion matrix --------------------
```



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



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



4.2.3.Sample Query point -1

```
In [72]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

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

```
Predicted Class : 1
Actual Class : 4
The  31  nearest neighbours of the test points belongs to classes [4 4 4 4 4 4 1 4 4 4 4 4 4 4
Fequency of nearest points : Counter({4: 29, 1: 2})
```

4.2.4. Sample Query Point-2

```
In [73]: clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         clf.fit(train_x_responseCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_responseCoding, train_y)

         test_point_index = 100

         predicted_cls = sig_clf.predict(test_x_responseCoding[test_point_index].reshape(1,-1)
         print("Predicted Class :", predicted_cls[0])
         print("Actual Class :", test_y[test_point_index])
         neighbors = clf.kneighbors(test_x_responseCoding[test_point_index].reshape(1, -1), al
         print("the k value for knn is",alpha[best_alpha],"and the nearest neighbours of the te
         print("Fequency of nearest points :",Counter(train_y[neighbors[1][0]]))
```

```
Predicted Class : 7
Actual Class : 7
the k value for knn is 31 and the nearest neighbours of the test points belongs to classes [7
Fequency of nearest points : Counter({7: 30, 2: 1})
```

4.3. Logistic Regression

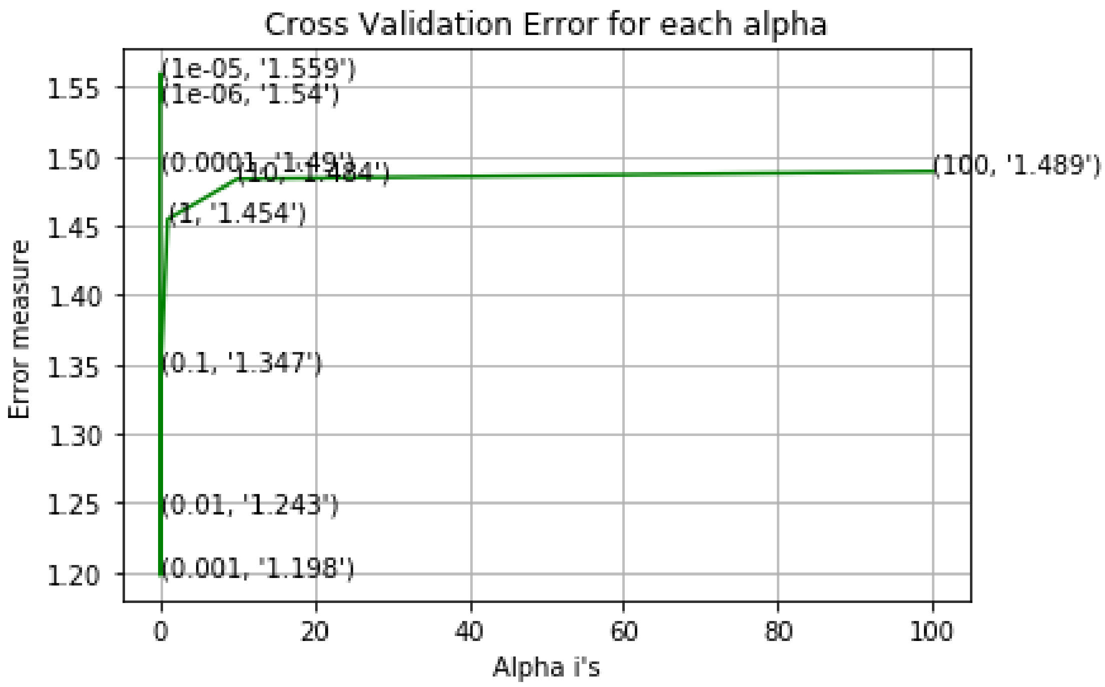
4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

```
In [153]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # -------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # predict(X)        Predict class labels for samples in X.

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


          # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
          # ----------------------------
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv
          #
          # 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', 
              clf.fit(train_x_onehotCoding, train_y)
              sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
              sig_clf.fit(train_x_onehotCoding, train_y)
              sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
              cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps
              # to avoid rounding error while multiplying probabilites we use log-probability 
              print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
    print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_
```

```
for alpha = 1e-06
Log Loss : 1.5401880899720983
for alpha = 1e-05
Log Loss : 1.5594105074598468
for alpha = 0.0001
Log Loss : 1.4901246512602186
for alpha = 0.001
Log Loss : 1.1979639909415651
for alpha = 0.01
Log Loss : 1.2432682249291072
for alpha = 0.1
Log Loss : 1.3472010828158667
for alpha = 1
Log Loss : 1.4543614579199502
for alpha = 10
Log Loss : 1.4835037155722661
for alpha = 100
Log Loss : 1.4889752652859862
```

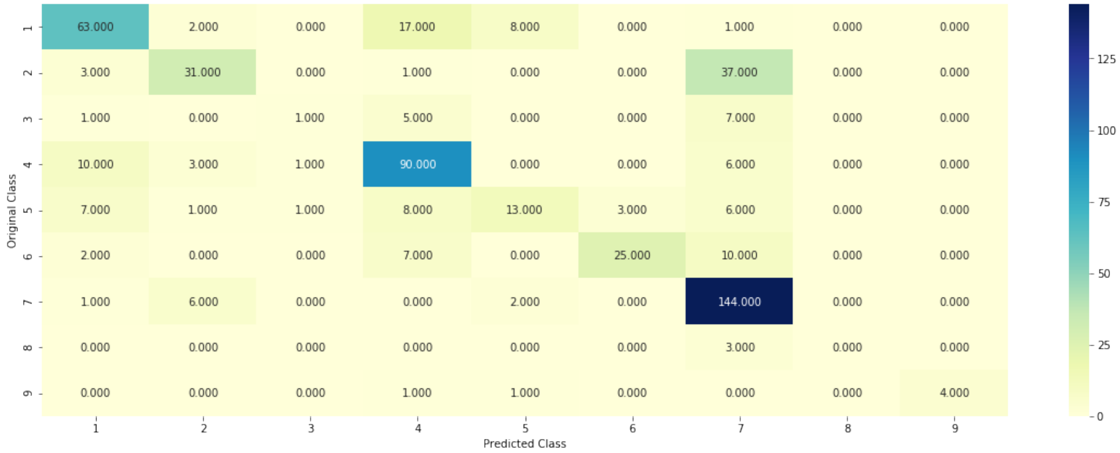


```
For values of best alpha =  0.001 The train log loss is: 0.7916100007675856
For values of best alpha =  0.001 The cross validation log loss is: 1.1979639909415651
For values of best alpha =  0.001 The test log loss is: 1.153974342982127
```

4.3.1.2. Testing the model with best hyper paramters

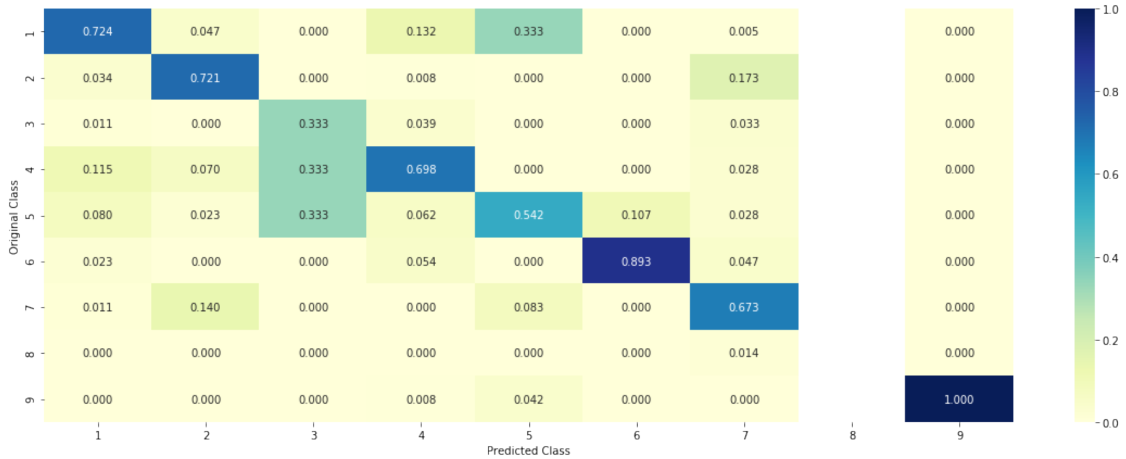
```
In [270]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # predict(X)        Predict class labels for samples in X.

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

```
Log loss : 1.0223579903158366
Number of mis-classified points : 0.3026315789473684
-------------------- Confusion matrix --------------------
```

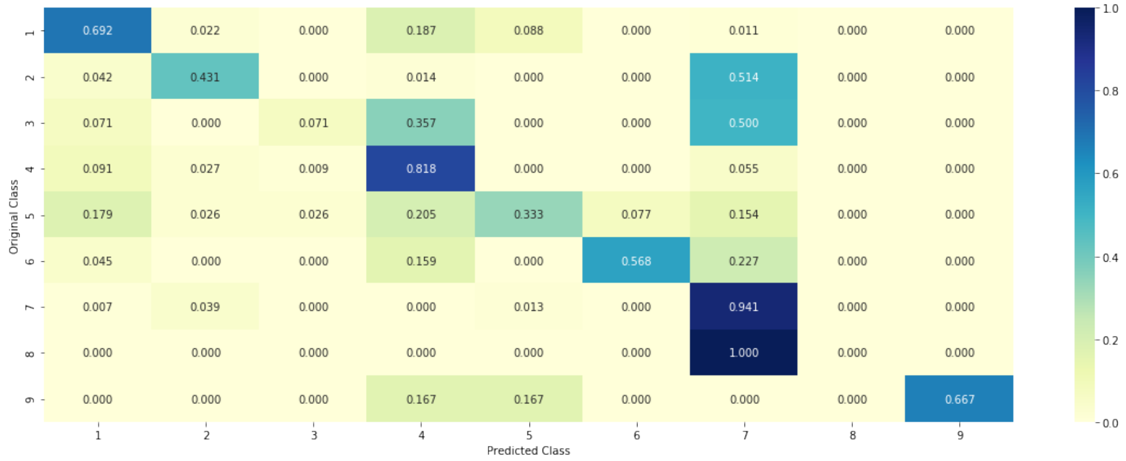


	1	2	3	4	5	6	7	8	9
1	63.000	2.000	0.000	17.000	8.000	0.000	1.000	0.000	0.000
2	3.000	31.000	0.000	1.000	0.000	0.000	37.000	0.000	0.000
3	1.000	0.000	1.000	5.000	0.000	0.000	7.000	0.000	0.000
4	10.000	3.000	1.000	90.000	0.000	0.000	6.000	0.000	0.000
5	7.000	1.000	1.000	8.000	13.000	3.000	6.000	0.000	0.000
6	2.000	0.000	0.000	7.000	0.000	25.000	10.000	0.000	0.000
7	1.000	6.000	0.000	0.000	2.000	0.000	144.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	3.000	0.000	0.000
9	0.000	0.000	0.000	1.000	1.000	0.000	0.000	0.000	4.000

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



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



4.3.1.3. Feature Importance

```
In [271]: def get_imp_feature_names(text, indices, removed_ind = []):
              word_present = 0
              tabulte_list = []
              incresingorder_ind = 0
              for i in indices:
                  if i < train_gene_feature_onehotCoding.shape[1]:
                      tabulte_list.append([incresingorder_ind, "Gene", "Yes"])
                  elif i< 18:
                      tabulte_list.append([incresingorder_ind,"Variation", "Yes"])
                  if ((i > 17) & (i not in removed_ind)) :
                      word = train_text_features[i]
```

```
            yes_no = True if word in text.split() else False
            if yes_no:
                word_present += 1
            tabulte_list.append([incresingorder_ind,train_text_features[i], yes_no])
        incresingorder_ind += 1
    print(word_present, "most importent features are present in our query point")
    print("-"*50)
    print("The features that are most importent of the ",predicted_cls[0]," class:")
    print (tabulate(tabulte_list, headers=["Index",'Feature name', 'Present or Not']
```

4.3.1.3.1. Correctly Classified point

```
In [272]: # from tabulate import tabulate
          clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', 
          clf.fit(train_x_onehotCoding,train_y)
          test_point_index = 1
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.1767 0.017  0.0055 0.7425 0.0094 0.0074 0.038  0.0026 0.0008
Actual Class : 4
--------------------------------------------------
359 Text feature [76] present in test data point [True]
363 Text feature [29] present in test data point [True]
449 Text feature [57] present in test data point [True]
480 Text feature [2b] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.1.3.2. Incorrectly Classified point

```
In [274]: test_point_index = 12
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
          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
```

```
Predicted Class : 5
Predicted Class Probabilities: [[0.0196 0.0203 0.0142 0.2299 0.4456 0.0136 0.008  0.0895 0.1595
```

```
Actual Class : 7
--------------------------------------------------
238 Text feature [100] present in test data point [True]
262 Text feature [2217] present in test data point [True]
380 Text feature [1t01] present in test data point [True]
454 Text feature [2a] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

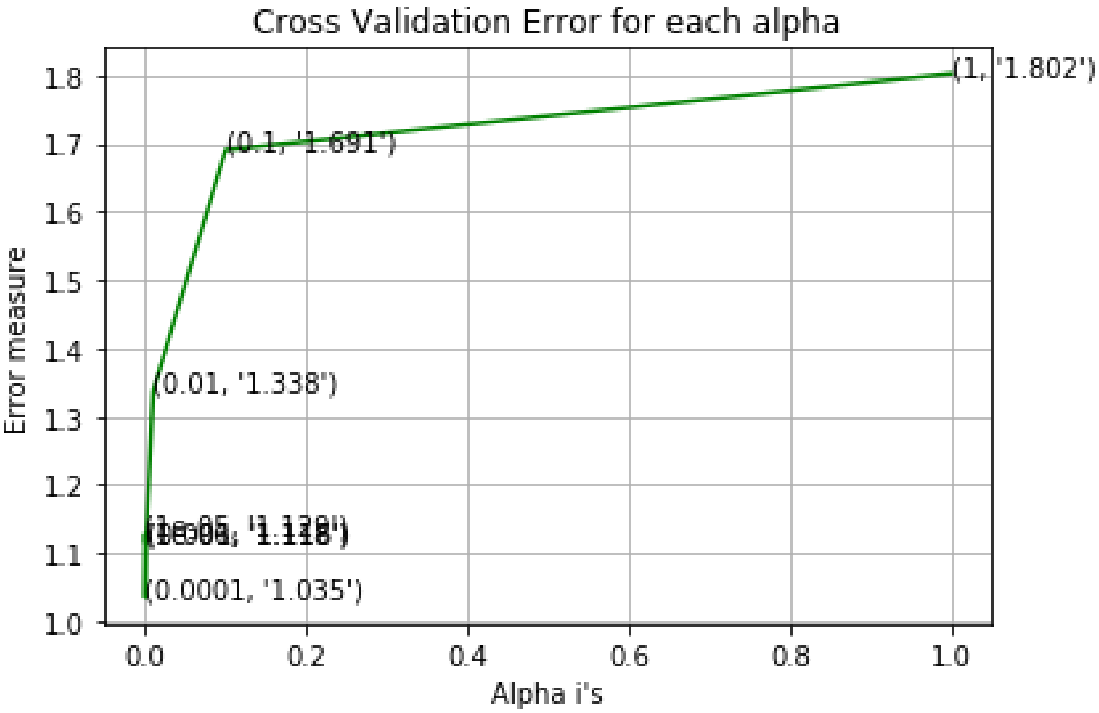
4.3.2. Without Class balancing

4.3.2.1. Hyper paramter tuning

```
In [275]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # predict(X)        Predict class labels for samples in X.

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




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

          alpha = [10 ** x for x in range(-6, 1)]
          cv_log_error_array = []
          for i in alpha:
              print("for alpha =", i)
              clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
```

```
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

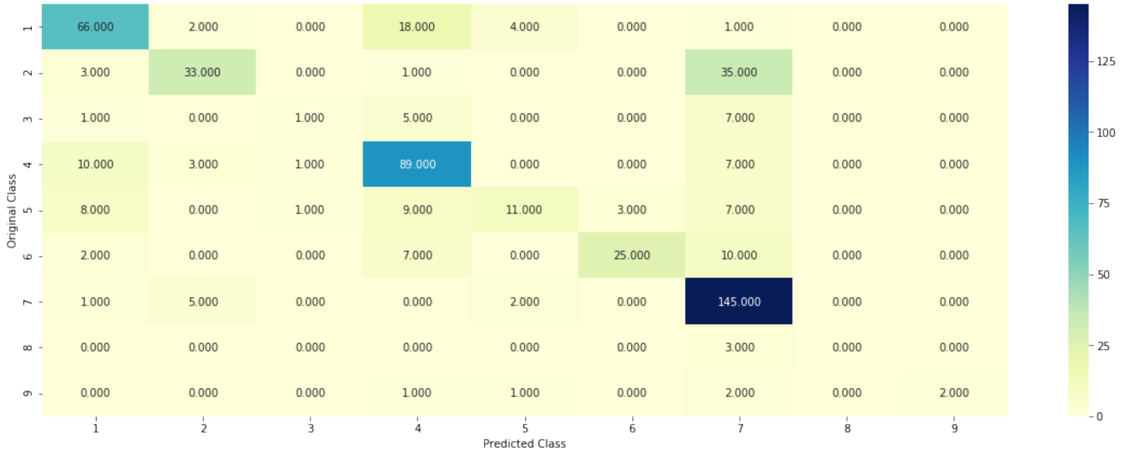
```
for alpha = 1e-06
Log Loss : 1.1178316147139558
for alpha = 1e-05
Log Loss : 1.1286700941375547
for alpha = 0.0001
Log Loss : 1.0354905771417635
for alpha = 0.001
Log Loss : 1.116192796578188
for alpha = 0.01
Log Loss : 1.337809693021511
for alpha = 0.1
Log Loss : 1.6906945613078885
for alpha = 1
Log Loss : 1.8021462124853862
```



```
For values of best alpha =  0.0001 The train log loss is: 0.4171340627067658
For values of best alpha =  0.0001 The cross validation log loss is: 1.0354905771417635
For values of best alpha =  0.0001 The test log loss is: 1.0890151910016732
```

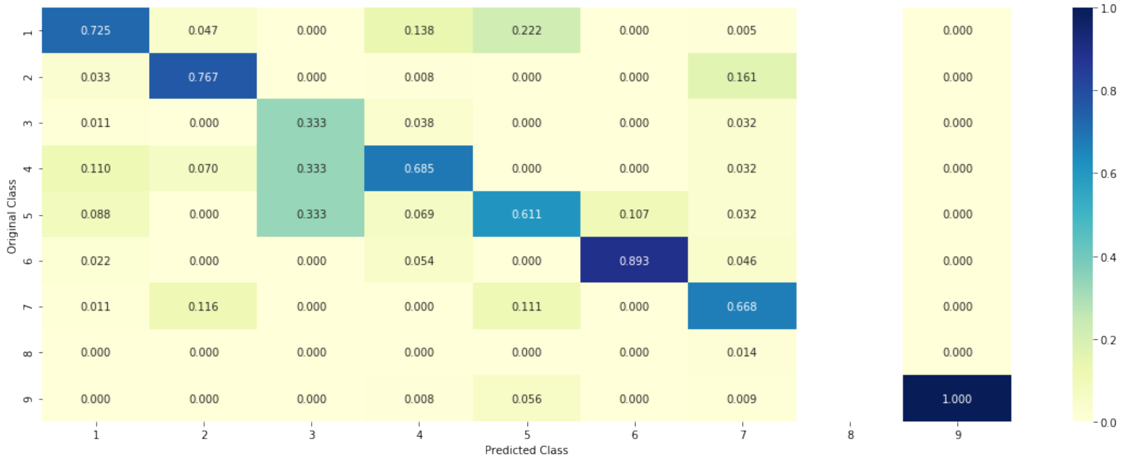
4.3.2.2. Testing model with best hyper parameters

```
In [276]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # predict(X)        Predict class labels for samples in X.

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

          clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
          predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y, cv_x_onehotCoding, 
Log loss : 1.0354905771417635
Number of mis-classified points : 0.3007518796992481
-------------------- Confusion matrix --------------------
```



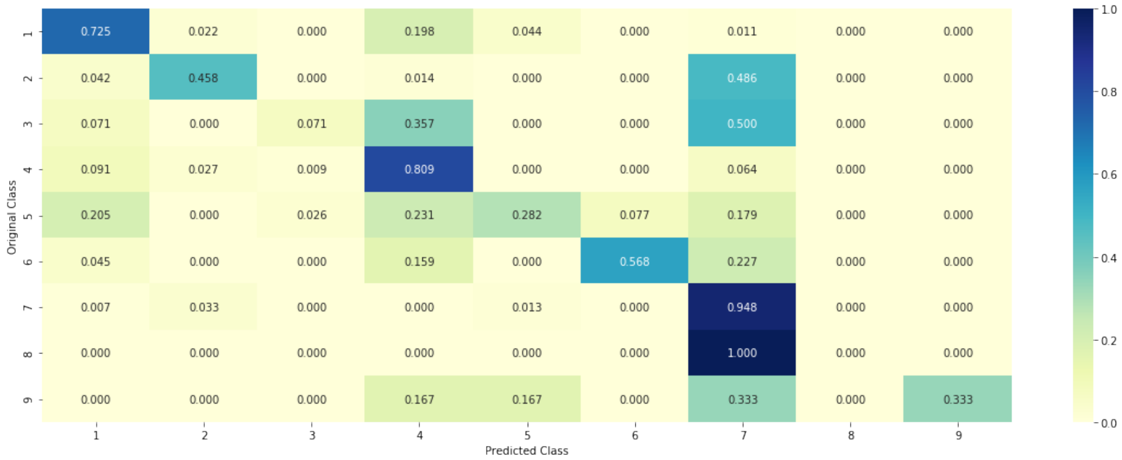
	1	2	3	4	5	6	7	8	9
1	66.000	2.000	0.000	18.000	4.000	0.000	1.000	0.000	0.000
2	3.000	33.000	0.000	1.000	0.000	0.000	35.000	0.000	0.000
3	1.000	0.000	1.000	5.000	0.000	0.000	7.000	0.000	0.000
4	10.000	3.000	1.000	89.000	0.000	0.000	7.000	0.000	0.000
5	8.000	0.000	1.000	9.000	11.000	3.000	7.000	0.000	0.000
6	2.000	0.000	0.000	7.000	0.000	25.000	10.000	0.000	0.000
7	1.000	5.000	0.000	0.000	2.000	0.000	145.000	0.000	0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	3.000	0.000	0.000
9	0.000	0.000	0.000	1.000	1.000	0.000	2.000	0.000	2.000

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



	1	2	3	4	5	6	7	8	9
1	0.725	0.047	0.000	0.138	0.222	0.000	0.005		0.000
2	0.033	0.767	0.000	0.008	0.000	0.000	0.161		0.000
3	0.011	0.000	0.333	0.038	0.000	0.000	0.032		0.000
4	0.110	0.070	0.333	0.685	0.000	0.000	0.032		0.000
5	0.088	0.000	0.333	0.069	0.611	0.107	0.032		0.000
6	0.022	0.000	0.000	0.054	0.000	0.893	0.046		0.000
7	0.011	0.116	0.000	0.000	0.111	0.000	0.668		0.000
8	0.000	0.000	0.000	0.000	0.000	0.000	0.014		0.000
9	0.000	0.000	0.000	0.008	0.056	0.000	0.009		1.000

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



4.3.2.3. Feature Importance, Correctly Classified point

```
In [277]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=
          clf.fit(train_x_onehotCoding,train_y)
          test_point_index = 1
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehot
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[1.751e-01 1.710e-02 5.200e-03 7.410e-01 8.900e-03 7.000e-03 4
  2.300e-03 4.000e-04]]
Actual Class : 4
--------------------------------------------------
370 Text feature [76] present in test data point [True]
397 Text feature [29] present in test data point [True]
464 Text feature [57] present in test data point [True]
484 Text feature [145] present in test data point [True]
Out of the top  500  features  4 are present in query point
```

4.3.2.4. Feature Importance, Inorrectly Classified point

```
In [278]: test_point_index = 11
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehot
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.4698 0.0084 0.0005 0.498  0.0076 0.0027 0.0124 0.0005 0.
Actual Class : 1
--------------------------------------------------
299 Text feature [75] present in test data point [True]
370 Text feature [76] present in test data point [True]
397 Text feature [29] present in test data point [True]
Out of the top  500  features  3 are present in query point
```

4.4. Linear Support Vector Machines

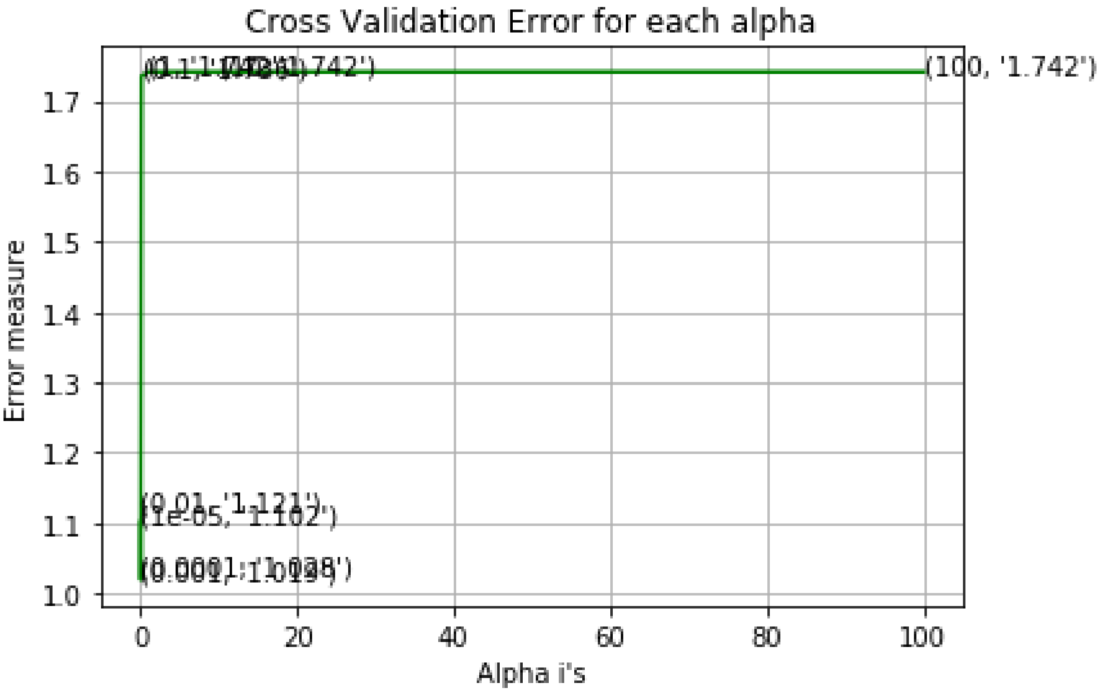
4.4.1. Hyper paramter tuning

```
In [224]: # read more about support vector machines with linear kernals here http://scikit-lea

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

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



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

```python
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
    clf.fit(train_x_onehotCoding, train_y)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_x_onehotCoding, train_y)
    sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
    cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps
    print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for C = 1e-05
Log Loss : 1.102137759816982
for C = 0.0001
Log Loss : 1.0281363076815744
for C = 0.001
Log Loss : 1.0194443230177024
for C = 0.01
Log Loss : 1.120774433971054
for C = 0.1
Log Loss : 1.736006667328727
```

```
for C = 1
Log Loss : 1.741688158603737
for C = 10
Log Loss : 1.741686539288531
for C = 100
Log Loss : 1.7416865873401068
```

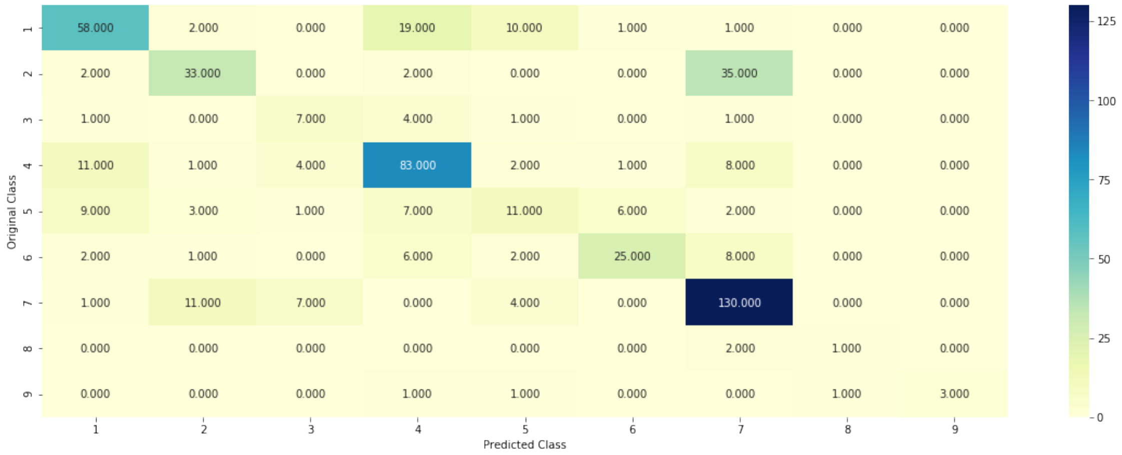


```
For values of best alpha =  0.001 The train log loss is: 0.4756151048578769
For values of best alpha =  0.001 The cross validation log loss is: 1.0194443230177024
For values of best alpha =  0.001 The test log loss is: 1.0810094120789544
```

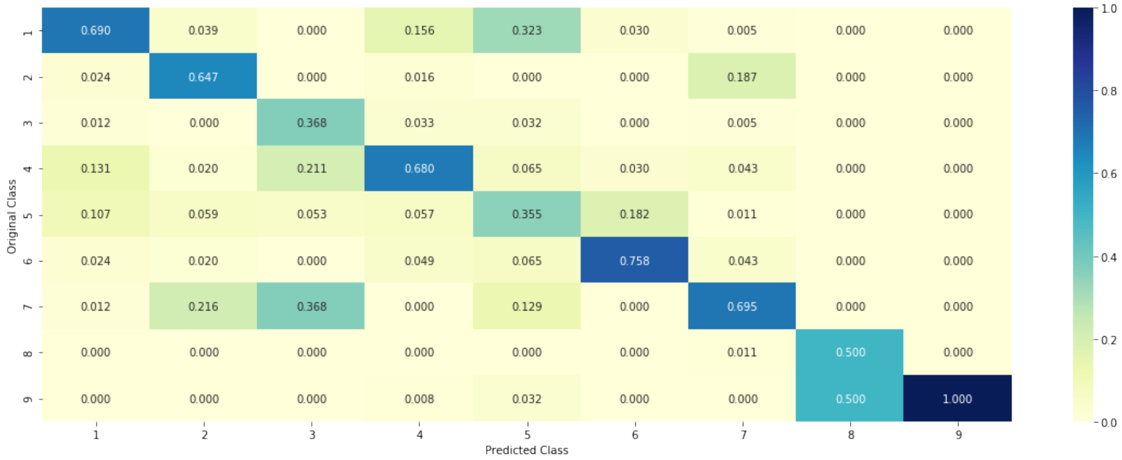
4.4.2. Testing model with best hyper parameters

In [225]:
```
# read more about support vector machines with linear kernals here http://scikit-lea

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

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

# ----------------------------------


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

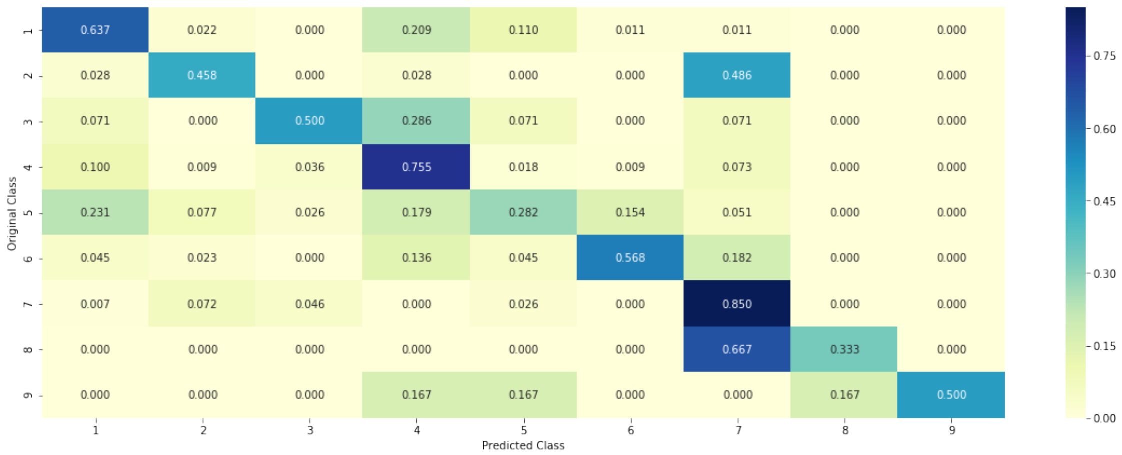
```
Log loss : 1.0194443230177024
Number of mis-classified points : 0.34022556390977443
-------------------- Confusion matrix --------------------
```



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



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



4.3.3. Feature Importance
4.3.3.1. For Correctly classified point

```
In [226]: clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
          clf.fit(train_x_onehotCoding,train_y)
          test_point_index = 1
          # test_point_index = 100
          no_feature = 500
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[4.610e-02 1.870e-02 3.600e-03 8.511e-01 1.370e-02 6.500e-03 5
  1.000e-03 6.000e-04]]
Actual Class : 4
--------------------------------------------------
256 Text feature [6b] present in test data point [True]
329 Text feature [42] present in test data point [True]
377 Text feature [analogous] present in test data point [True]
434 Text feature [address] present in test data point [True]
456 Text feature [analyses] present in test data point [True]
467 Text feature [80] present in test data point [True]
Out of the top  500  features  6 are present in query point
```

4.3.3.2. For Incorrectly classified point

```
In [227]: test_point_index = 11
          no_feature = 1000
          predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
          print("Predicted Class :", predicted_cls[0])
          print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotC
          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
```

```
Predicted Class : 4
Predicted Class Probabilities: [[0.2319 0.0116 0.0033 0.5717 0.0243 0.009  0.1468 0.0006 0.0008
Actual Class : 1
--------------------------------------------------
168 Text feature [99] present in test data point [True]
191 Text feature [addressed] present in test data point [True]
265 Text feature [accessible] present in test data point [True]
329 Text feature [42] present in test data point [True]
343 Text feature [affinity42] present in test data point [True]
377 Text feature [analogous] present in test data point [True]
434 Text feature [address] present in test data point [True]
456 Text feature [analyses] present in test data point [True]
554 Text feature [1989] present in test data point [True]
561 Text feature [agents] present in test data point [True]
614 Text feature [abundant] present in test data point [True]
623 Text feature [55] present in test data point [True]
653 Text feature [affinities] present in test data point [True]
678 Text feature [afterwards] present in test data point [True]
682 Text feature [appears] present in test data point [True]
709 Text feature [activating] present in test data point [True]
710 Text feature [29] present in test data point [True]
799 Text feature [11] present in test data point [True]
828 Text feature [23] present in test data point [True]
862 Text feature [33] present in test data point [True]
Out of the top  1000  features  20 are present in query point
```

4.5 Random Forest Classifier

4.5.1. Hyper paramter tuning (With One hot Encoding)

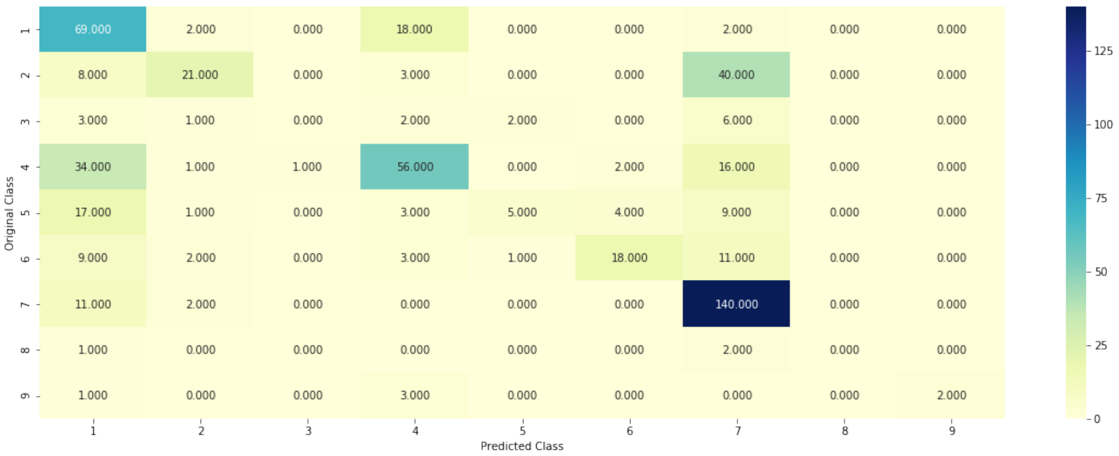
```
In [91]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=N
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)
         # Some of methods of RandomForestClassifier()
         # fit(X, y, [sample_weight])        Fit the SVM model according to the given training
         # 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/
         # --------------------------------


         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # 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, ra
                 clf.fit(train_x_onehotCoding, train_y)
                 sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
                 sig_clf.fit(train_x_onehotCoding, train_y)
                 sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
                 cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_,
                 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_e
         plt.grid()
         plt.title("Cross Validation Error for each alpha")
         plt.xlabel("Alpha i's")
         plt.ylabel("Error measure")
         plt.show()
         '''

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

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

```
for n_estimators = 100 and max depth =  5
Log Loss : 1.2463453963006337
for n_estimators = 100 and max depth =  10
Log Loss : 1.2559771619712576
for n_estimators = 200 and max depth =  5
Log Loss : 1.2361175912958131
for n_estimators = 200 and max depth =  10
Log Loss : 1.2452368980466992
for n_estimators = 500 and max depth =  5
Log Loss : 1.2305268439400538
for n_estimators = 500 and max depth =  10
Log Loss : 1.2406945614312292
for n_estimators = 1000 and max depth =  5
Log Loss : 1.2308964085439669
for n_estimators = 1000 and max depth =  10
Log Loss : 1.2386235785189585
for n_estimators = 2000 and max depth =  5
Log Loss : 1.2290083778730985
for n_estimators = 2000 and max depth =  10
Log Loss : 1.2372246234447666
For values of best estimator =  2000 The train log loss is: 0.8624403124225934
For values of best estimator =  2000 The cross validation log loss is: 1.2290083778730985
For values of best estimator =  2000 The test log loss is: 1.222301769822958
```

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

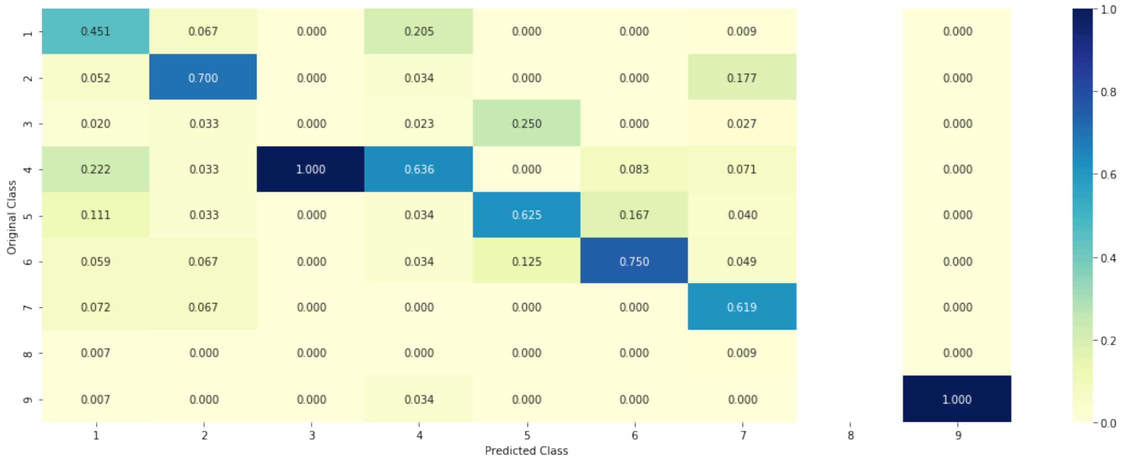
```
In [92]: # --------------------------------
         # default parameters
         # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
         # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_nodes=
         # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=No
         # class_weight=None)

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

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

```
Log loss : 1.2290083778730985
Number of mis-classified points : 0.41541353383458646
-------------------- Confusion matrix --------------------
```

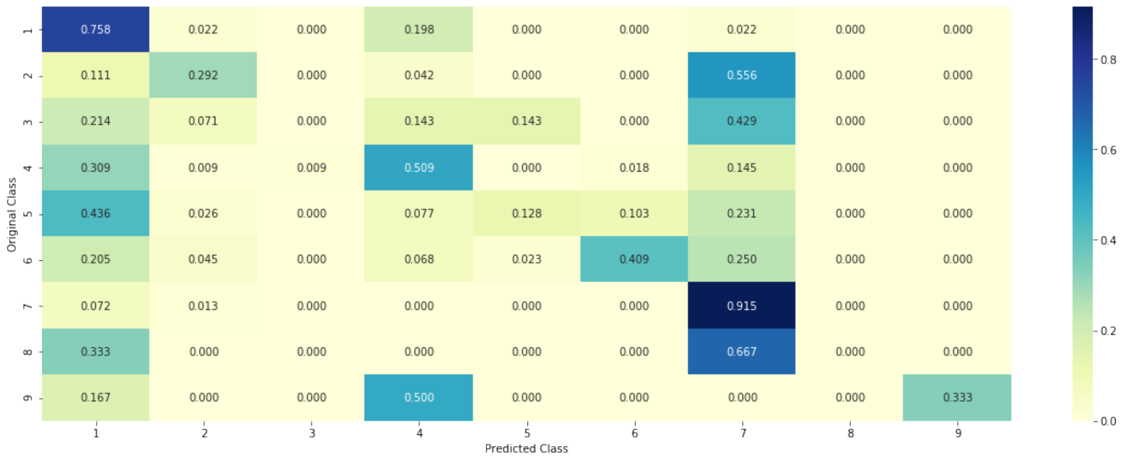


	1	2	3	4	5	6	7	8	9
1	69.000	2.000	0.000	18.000	0.000	0.000	2.000	0.000	0.000
2	8.000	21.000	0.000	3.000	0.000	0.000	40.000	0.000	0.000
3	3.000	1.000	0.000	2.000	2.000	0.000	6.000	0.000	0.000
4	34.000	1.000	1.000	56.000	0.000	2.000	16.000	0.000	0.000
5	17.000	1.000	0.000	3.000	5.000	4.000	9.000	0.000	0.000
6	9.000	2.000	0.000	3.000	1.000	18.000	11.000	0.000	0.000
7	11.000	2.000	0.000	0.000	0.000	0.000	140.000	0.000	0.000
8	1.000	0.000	0.000	0.000	0.000	0.000	2.000	0.000	0.000
9	1.000	0.000	0.000	3.000	0.000	0.000	0.000	0.000	2.000

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



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



4.5.3. Feature Importance

4.5.3.1. Correctly Classified point

```
In [98]: # test_point_index = 10
         clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/2)], criterion='gini',
         clf.fit(train_x_onehotCoding, train_y)
         sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
         sig_clf.fit(train_x_onehotCoding, train_y)

         test_point_index = 6
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         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
```

```
Predicted Class : 7
Predicted Class Probabilities: [[2.600e-02 2.585e-01 2.240e-02 1.850e-02 3.530e-02 3.230e-02 6
  3.300e-03 5.000e-04]]
Actual Class : 7
--------------------------------------------------
46 Text feature [10] present in test data point [True]
82 Text feature [11] present in test data point [True]
118 Text feature [1166] present in test data point [True]
399 Text feature [12] present in test data point [True]
513 Text feature [0013] present in test data point [True]
587 Text feature [100k] present in test data point [True]
639 Text feature [1038] present in test data point [True]
666 Text feature [1253] present in test data point [True]
709 Text feature [1235] present in test data point [True]
758 Text feature [100] present in test data point [True]
805 Text feature [123] present in test data point [True]
866 Text feature [1271] present in test data point [True]
873 Text feature [13] present in test data point [True]
Out of the top  1000  features  13 are present in query point
```

4.5.3.2. Inorrectly Classified point

```
In [96]: test_point_index = 100
         no_feature = 1000
         predicted_cls = sig_clf.predict(test_x_onehotCoding[test_point_index])
         print("Predicted Class :", predicted_cls[0])
         print("Predicted Class Probabilities:", np.round(sig_clf.predict_proba(test_x_onehotCo
         print("Actuall Class :", test_y[test_point_index])
         indices = np.argsort(-clf.feature_importances_)
         print("-"*50)
         get_impfeature_names(indices[:no_feature], test_df['TEXT'].iloc[test_point_index],test
```

```
Predicted Class : 1
Predicted Class Probabilities: [[0.3595 0.1361 0.0202 0.1153 0.0558 0.052  0.2084 0.0224 0.030
Actuall Class : 7
--------------------------------------------------
46 Text feature [10] present in test data point [True]
82 Text feature [11] present in test data point [True]
160 Text feature [1011] present in test data point [True]
174 Text feature [000] present in test data point [True]
337 Text feature [11b] present in test data point [True]
399 Text feature [12] present in test data point [True]
```

```
404 Text feature [105] present in test data point [True]
559 Text feature [11a] present in test data point [True]
578 Text feature [112] present in test data point [True]
597 Text feature [106] present in test data point [True]
668 Text feature [10ng] present in test data point [True]
873 Text feature [13] present in test data point [True]
915 Text feature [120] present in test data point [True]
963 Text feature [104] present in test data point [True]
Out of the top  1000  features  14 are present in query point
```

4.5.3. Hyper paramter tuning (With Response Coding)

```
In [228]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth=
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # 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/lesson
          # --------------------------------


          # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modu
          # ----------------------------
          # default paramters
          # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv
          #
          # 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, 
        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_,
        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_
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'
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 i
predict_y = sig_clf.predict_proba(cv_x_responseCoding)
print('For values of best alpha = ', alpha[int(best_alpha/4)], "The cross validation
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
```

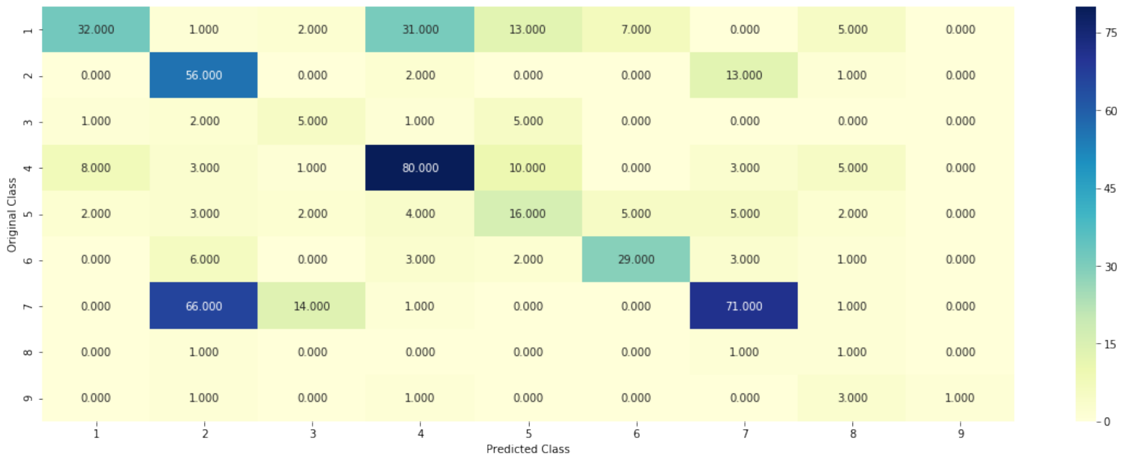
```
for n_estimators = 10 and max depth =  2
Log Loss : 2.192496933743523
for n_estimators = 10 and max depth =  3
Log Loss : 1.6243040950960994
for n_estimators = 10 and max depth =  5
Log Loss : 1.5308266607435403
for n_estimators = 10 and max depth =  10
Log Loss : 1.8192520740442522
for n_estimators = 50 and max depth =  2
```

```
Log Loss : 1.7701639900960602
for n_estimators = 50 and max depth =  3
Log Loss : 1.4504106723838643
for n_estimators = 50 and max depth =  5
Log Loss : 1.3837874669499097
for n_estimators = 50 and max depth =  10
Log Loss : 1.762333496374847
for n_estimators = 100 and max depth =  2
Log Loss : 1.5915780383091052
for n_estimators = 100 and max depth =  3
Log Loss : 1.4602516790636246
for n_estimators = 100 and max depth =  5
Log Loss : 1.3295532814629036
for n_estimators = 100 and max depth =  10
Log Loss : 1.7423825982098107
for n_estimators = 200 and max depth =  2
Log Loss : 1.652003390988076
for n_estimators = 200 and max depth =  3
Log Loss : 1.5106825460726994
for n_estimators = 200 and max depth =  5
Log Loss : 1.4288996539674041
for n_estimators = 200 and max depth =  10
Log Loss : 1.7356548570365746
for n_estimators = 500 and max depth =  2
Log Loss : 1.7384544423268617
for n_estimators = 500 and max depth =  3
Log Loss : 1.575111086481199
for n_estimators = 500 and max depth =  5
Log Loss : 1.43679218091445
for n_estimators = 500 and max depth =  10
Log Loss : 1.7576244318324115
for n_estimators = 1000 and max depth =  2
Log Loss : 1.7127404674883175
for n_estimators = 1000 and max depth =  3
Log Loss : 1.6025057684578634
for n_estimators = 1000 and max depth =  5
Log Loss : 1.4407835424113442
for n_estimators = 1000 and max depth =  10
Log Loss : 1.7223095722099377
For values of best alpha =  100 The train log loss is: 0.06171722023141785
For values of best alpha =  100 The cross validation log loss is: 1.3295532814629036
For values of best alpha =  100 The test log loss is: 1.339510557001445
```

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

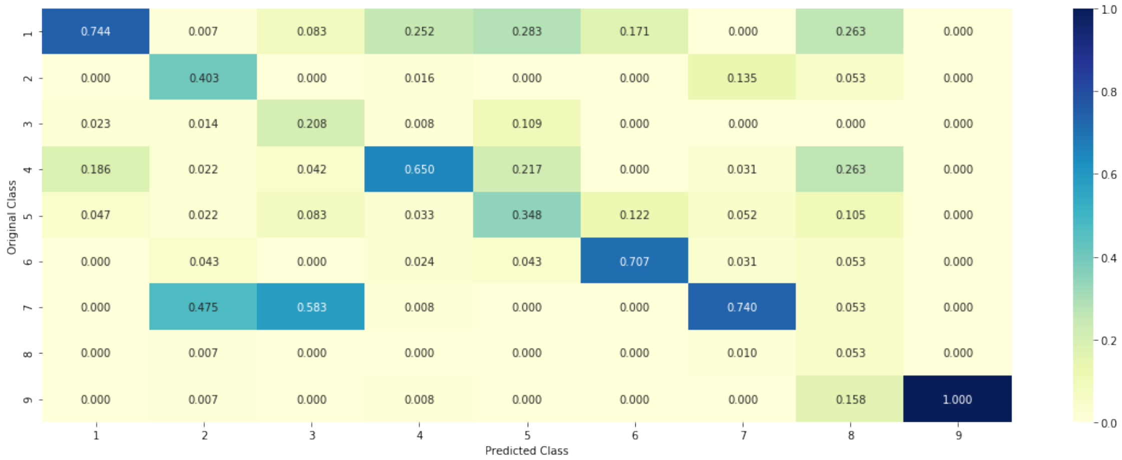
```
In [182]: # --------------------------------
          # default parameters
          # sklearn.ensemble.RandomForestClassifier(n_estimators=10, criterion=gini, max_depth
          # min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=auto, max_leaf_node
          # min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=N
          # class_weight=None)

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # 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/lesson
          # --------------------------------

          clf = RandomForestClassifier(max_depth=max_depth[int(best_alpha%4)], n_estimators=al
          predict_and_plot_confusion_matrix(train_x_responseCoding, train_y,cv_x_responseCodin
```

```
Log loss : 1.3295532814629036
Number of mis-classified points : 0.45300751879699247
-------------------- Confusion matrix --------------------
```

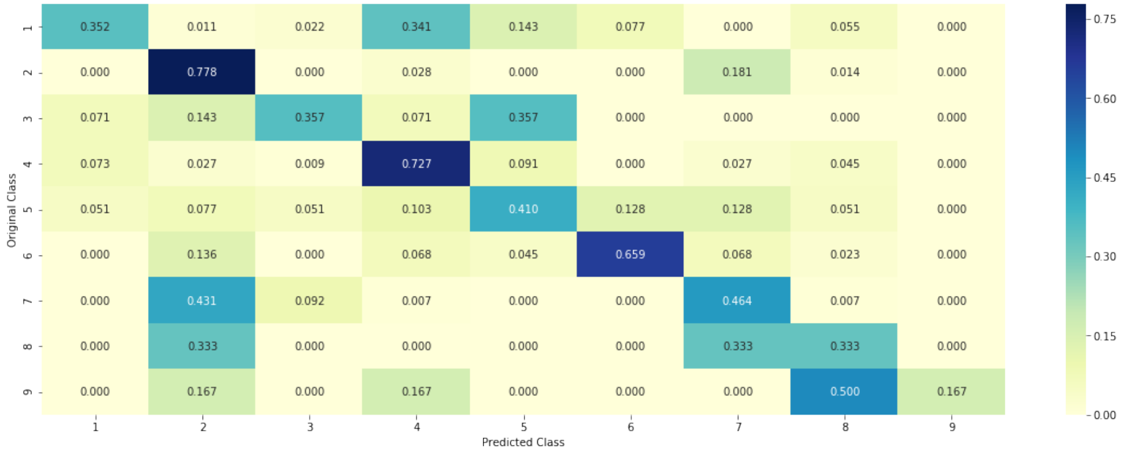


	1	2	3	4	5	6	7	8	9
1	32.000	1.000	2.000	31.000	13.000	7.000	0.000	5.000	0.000
2	0.000	56.000	0.000	2.000	0.000	0.000	13.000	1.000	0.000
3	1.000	2.000	5.000	1.000	5.000	0.000	0.000	0.000	0.000
4	8.000	3.000	1.000	80.000	10.000	0.000	3.000	5.000	0.000
5	2.000	3.000	2.000	4.000	16.000	5.000	5.000	2.000	0.000
6	0.000	6.000	0.000	3.000	2.000	29.000	3.000	1.000	0.000
7	0.000	66.000	14.000	1.000	0.000	0.000	71.000	1.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
9	0.000	1.000	0.000	1.000	0.000	0.000	0.000	3.000	1.000

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



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



4.5.5. Feature Importance

4.5.5.1. Correctly Classified point

```
In [183]: clf = RandomForestClassifier(n_estimators=alpha[int(best_alpha/4)], criterion='gini'
          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_respons
          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 : 4
Predicted Class Probabilities: [[0.1603 0.0328 0.1939 0.4421 0.0422 0.0448 0.0123 0.0514 0.020
Actual Class : 4
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.5.5.2. Incorrectly Classified point

```
In [184]: test_point_index = 6
          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_respons
          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: [[0.0092 0.4578 0.0987 0.0184 0.0318 0.0357 0.3229 0.0167 0.008
Actual Class : 7
--------------------------------------------------
Variation is important feature
Variation is important feature
Variation is important feature
Variation is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Gene is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Variation is important feature
Gene is important feature
Text is important feature
Gene is important feature
Variation is important feature
Variation is important feature
Text is important feature
Text is important feature
Text is important feature
Gene is important feature
Gene is important feature
Gene is important feature
```

4.7 Stack the models

4.7.1 testing with hyper parameter tuning

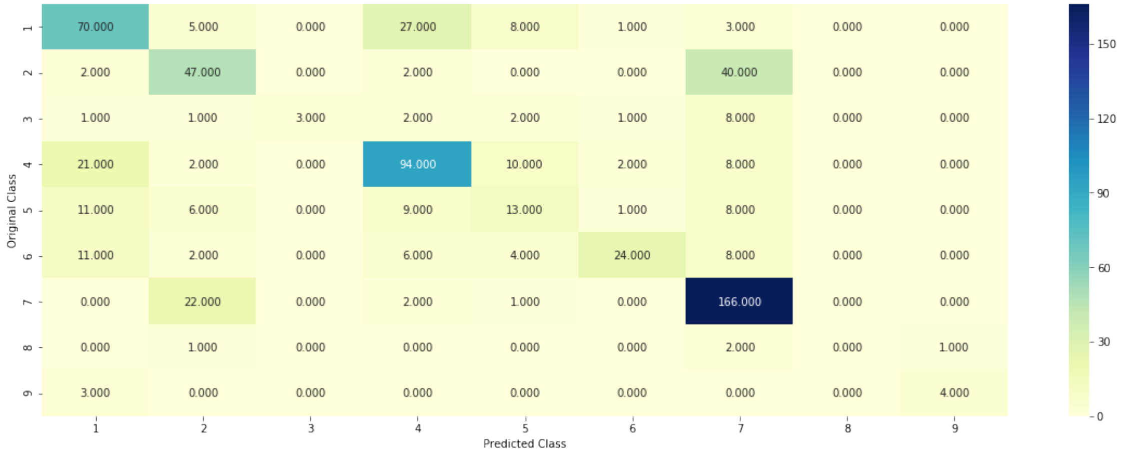
```
In [229]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generate
          # ------------------------------
          # default parameters
          # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=T
          # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=o
          # 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 G
          # predict(X)        Predict class labels for samples in X.

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


          # read more about support vector machines with linear kernals here http://scikit-lea
          # --------------------------------
          # default parameters
          # SVC(C=1.0, kernel=rbf, degree=3, gamma=auto, coef0=0.0, shrinking=True, probabilit
          # cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_s

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


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

          # Some of methods of RandomForestClassifier()
          # fit(X, y, [sample_weight])        Fit the SVM model according to the given trainin
          # 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/lesson
          # --------------------------------


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

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


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

          sig_clf1.fit(train_x_onehotCoding, train_y)
          print("Logistic Regression :  Log Loss: %0.2f" % (log_loss(cv_y, sig_clf1.predict_pro
          sig_clf2.fit(train_x_onehotCoding, train_y)
          print("Support vector machines : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf2.predict_
          sig_clf3.fit(train_x_onehotCoding, train_y)
          print("Naive Bayes : Log Loss: %0.2f" % (log_loss(cv_y, sig_clf3.predict_proba(cv_x_
          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_class
              sclf.fit(train_x_onehotCoding, train_y)
              print("Stacking Classifer : for the value of alpha: %f Log Loss: %0.3f" % (i, log
              log_error =log_loss(cv_y, sclf.predict_proba(cv_x_onehotCoding))
              if best_alpha > log_error:
                  best_alpha = log_error
```

```
Logistic Regression :  Log Loss: 1.02
Support vector machines : Log Loss: 1.74
Naive Bayes : Log Loss: 1.24
--------------------------------------------------
Stacking Classifer : for the value of alpha: 0.000100 Log Loss: 2.177
Stacking Classifer : for the value of alpha: 0.001000 Log Loss: 2.025
Stacking Classifer : for the value of alpha: 0.010000 Log Loss: 1.468
Stacking Classifer : for the value of alpha: 0.100000 Log Loss: 1.104
Stacking Classifer : for the value of alpha: 1.000000 Log Loss: 1.207
Stacking Classifer : for the value of alpha: 10.000000 Log Loss: 1.435
```

4.7.2 testing the model with the best hyper parameters

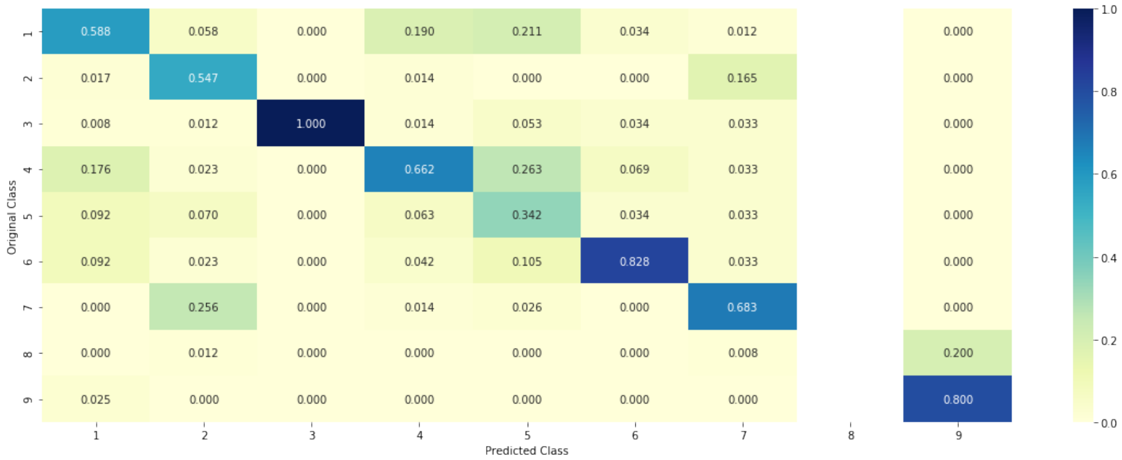
```
In [230]: lr = LogisticRegression(C=0.1)
          sclf = StackingClassifier(classifiers=[sig_clf1, sig_clf2, sig_clf3], meta_classifie
          sclf.fit(train_x_onehotCoding, train_y)

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

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

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

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

```
Log loss (train) on the stacking classifier : 0.6480965976927581
Log loss (CV) on the stacking classifier : 1.1039478366110875
Log loss (test) on the stacking classifier : 1.1349307092092904
Number of missclassified point : 0.3669172932330827
-------------------- Confusion matrix --------------------
```

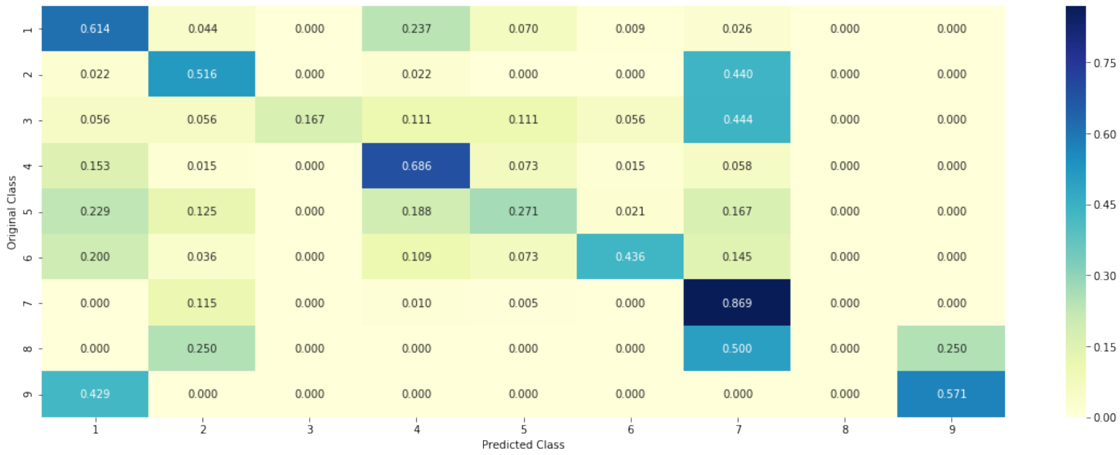


	1	2	3	4	5	6	7	8	9
1	70.000	5.000	0.000	27.000	8.000	1.000	3.000	0.000	0.000
2	2.000	47.000	0.000	2.000	0.000	0.000	40.000	0.000	0.000
3	1.000	1.000	3.000	2.000	2.000	1.000	8.000	0.000	0.000
4	21.000	2.000	0.000	94.000	10.000	2.000	8.000	0.000	0.000
5	11.000	6.000	0.000	9.000	13.000	1.000	8.000	0.000	0.000
6	11.000	2.000	0.000	6.000	4.000	24.000	8.000	0.000	0.000
7	0.000	22.000	0.000	2.000	1.000	0.000	166.000	0.000	0.000
8	0.000	1.000	0.000	0.000	0.000	0.000	2.000	0.000	1.000
9	3.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	4.000

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



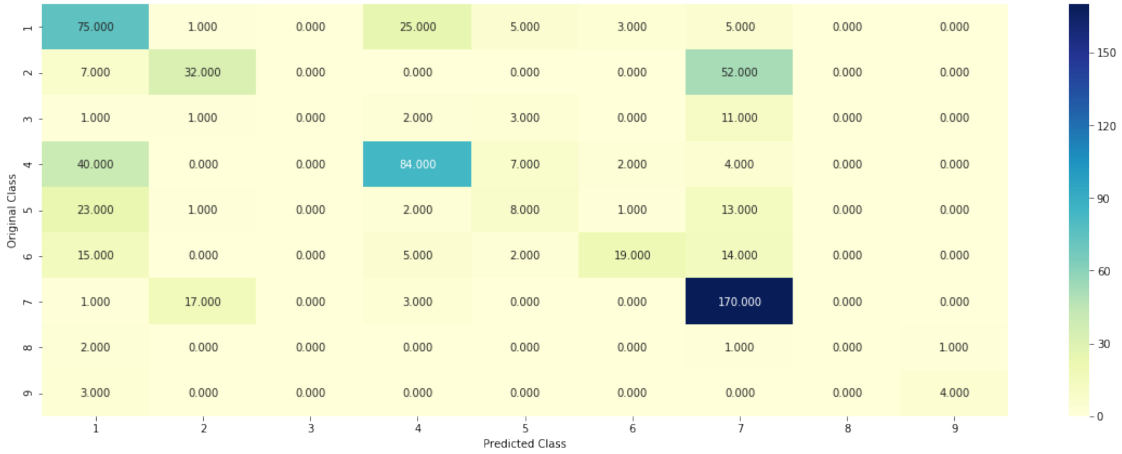
```
-------------------- Recall matrix (Row sum=1) --------------------
```



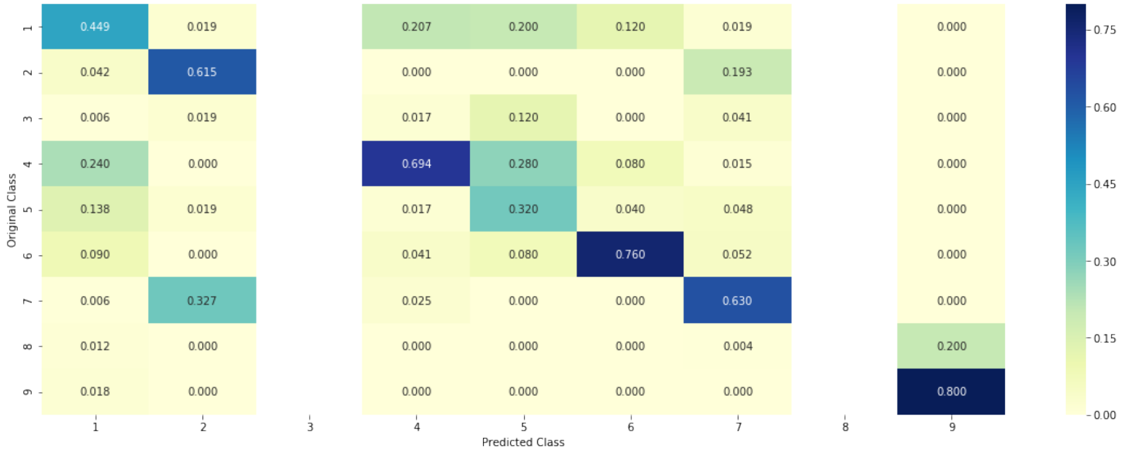
4.7.3 Maximum Voting classifier

```
In [106]: #Refer:http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClass
          from sklearn.ensemble import VotingClassifier
          vclf = VotingClassifier(estimators=[('lr', sig_clf1), ('svc', sig_clf2), ('rf', sig_c
          vclf.fit(train_x_onehotCoding, train_y)
          print("Log loss (train) on the VotingClassifier :", log_loss(train_y, vclf.predict_pr
          print("Log loss (CV) on the VotingClassifier :", log_loss(cv_y, vclf.predict_proba(cv
          print("Log loss (test) on the VotingClassifier :", log_loss(test_y, vclf.predict_prob
          print("Number of missclassified point :", np.count_nonzero((vclf.predict(test_x_oneho
          plot_confusion_matrix(test_y=test_y, predict_y=vclf.predict(test_x_onehotCoding))
```

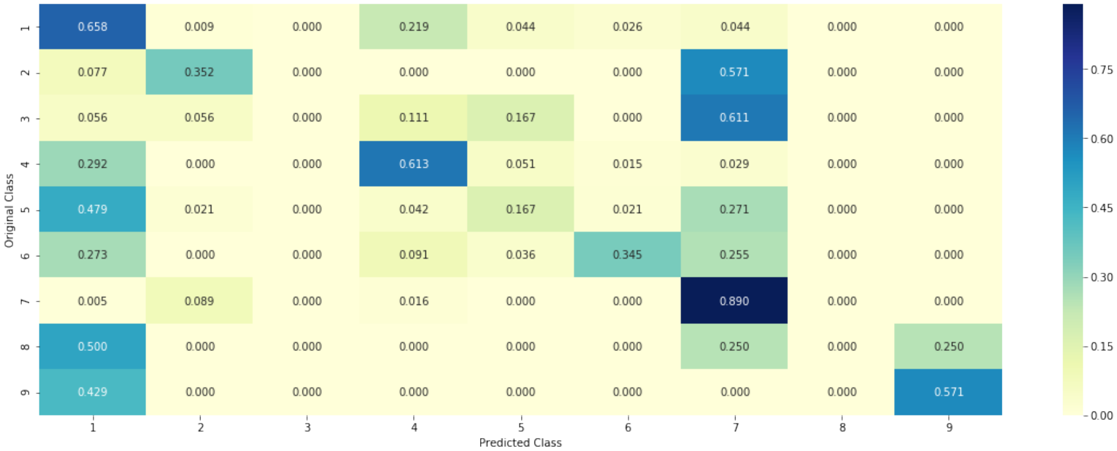
```
Log loss (train) on the VotingClassifier : 0.8228478056328926
Log loss (CV) on the VotingClassifier : 1.2484195699960718
Log loss (test) on the VotingClassifier : 1.2298293116433938
Number of missclassified point : 0.4105263157894737
-------------------- Confusion matrix --------------------
```



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



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



5. Assignments

```
<li> Apply All the models with tf-idf features (Replace CountVectorizer with tfidfVectorizer an
<li> Instead of using all the words in the dataset, use only the top 1000 words based of tf-idf
<li>Apply Logistic regression with CountVectorizer Features, including both unigrams and bigram
<li> Try any of the feature engineering techniques discussed in the course to reduce the CV and
```

0.0.1 1, 2

This is the performance of various models with respect to the tfidf features. I observed that few models were overfitting initially(when only 1000 top features were taken) so to resolve that issue I tried to run those models again with increased number of features.

- Performance table for tf-idf vectorizer with top 1000 features

```
In [17]: ######################### performance using top 1000 tfidf features #################

         pt = PrettyTable()
         pt.field_names = ["S.No.","Model","Best alpha", "Train", "Cross Validation","Test", "M
         pt.add_row(["1","Naive Bayes", 0.001, 0.513, 1.254, 1.226, 41])
         pt.add_row(["2","KNN", "k=31", 0.807, 1.092, 1.087, 37])
         pt.add_row(["3","LR with class balancing", 0.0001, 0.440, 1.049, 1.051, 34])
         pt.add_row(["4","LR without class balancing", 0.0001, 0.432, 1.090, 1.091, 33])
         pt.add_row(["5","Linear SVM", 0.0001, 0.397, 1.060, 1.089, 34])
         pt.add_row(["6","RF OneHot encoding", 2000, 0.862, 1.222, 1.222, 41])
         pt.add_row(["7","RF Response coding", 100, 0.061, 1.329, 1.339, 45])
         pt.add_row(["8","Stacking Models LR + SVM + NB", 0.1, 0.532, 1.229, 1.190, 40])
         pt.add_row(["9","Maximum Voting Classifier", 0.1, 0.822, 1.248, 1.229, 41])
         print(pt)
```

```
+--------+--------------------------------+------------+-------+------------------+-------+------
| S.No. |             Model              | Best alpha | Train | Cross Validation |  Test | Miscl
```

```
+-------+-------------------------------+------------+-------+------------------+-------+-----
|   1   |          Naive Bayes          |   0.001    | 0.513 |      1.254       | 1.226 |
|   2   |              KNN              |    k=31    | 0.807 |      1.092       | 1.087 |
|   3   |    LR with class balancing    |   0.0001   |  0.44 |      1.049       | 1.051 |
|   4   |   LR without class balancing  |   0.0001   | 0.432 |       1.09       | 1.091 |
|   5   |           Linear SVM          |   0.0001   | 0.397 |       1.06       | 1.089 |
|   6   |       RF OneHot encoding      |    2000    | 0.862 |      1.222       | 1.222 |
|   7   |       RF Response coding      |    100     | 0.061 |      1.329       | 1.339 |
|   8   | Stacking Models LR + SVM + NB |    0.1     | 0.532 |      1.229       |  1.19 |
|   9   |   Maximum Voting Classifier   |    0.1     | 0.822 |      1.248       | 1.229 |
+-------+-------------------------------+------------+-------+------------------+-------+-----
```

- Performance table for tf-idf vectorizer with 2000 features

```
In [19]: ################################ performace using 2000 top tfidf features ###########

         pt = PrettyTable()
         pt.field_names = ["S.No.","Model","Best alpha", "Train", "Cross Validation","Test", "
         pt.add_row(["1","Naive Bayes", 0.0001, 0.554, 1.273, 1.263, 42])
         pt.add_row(["2","LR with class balancing", 0.0001, 0.427, 1.035, 1.038, 31])
         pt.add_row(["3","LR without class balancing", 0.0001, 0.429, 1.073, 1.074, 32])
         pt.add_row(["4","Linear SVM", 0.001, 0.505, 1.055, 1.073, 34])
         pt.add_row(["5","RF Response coding", 100, 0.061, 1.329, 1.339, 45])
         pt.add_row(["6","Stacking Models LR + SVM + NB", 0.1, 0.587, 1.212, 1.189, 40])
         print(pt)
```

```
+-------+-------------------------------+------------+-------+------------------+-------+-----
| S.No. |             Model             | Best alpha | Train | Cross Validation |  Test | Miscl
+-------+-------------------------------+------------+-------+------------------+-------+-----
|   1   |          Naive Bayes          |   0.0001   | 0.554 |      1.273       | 1.263 |
|   2   |    LR with class balancing    |   0.0001   | 0.427 |      1.035       | 1.038 |
|   3   |   LR without class balancing  |   0.0001   | 0.429 |      1.073       | 1.074 |
|   4   |           Linear SVM          |   0.001    | 0.505 |      1.055       | 1.073 |
|   5   |       RF Response coding      |    100     | 0.061 |      1.329       | 1.339 |
|   6   | Stacking Models LR + SVM + NB |    0.1     | 0.587 |      1.212       | 1.189 |
+-------+-------------------------------+------------+-------+------------------+-------+-----
```

- Performance table for tf-idf vectorizer with 4000 features

```
In [20]: ############################ performance using 4000 top tfidf features ##############

         pt = PrettyTable()
         pt.field_names = ["S.No.","Model","Best alpha", "Train", "Cross Validation","Test", "
         pt.add_row(["1","Naive Bayes", 0.00001, 0.603, 1.245, 1.273, 39])
         pt.add_row(["2","LR with class balancing", 0.0001, 0.422, 1.025, 1.036, 31])
         pt.add_row(["3","LR without class balancing", 0.0001, 0.420, 1.066, 1.065, 31])
         pt.add_row(["4","Linear SVM", 0.0001, 0.415, 1.023, 1.082, 31])
```

```
        pt.add_row(["6","Stacking Models LR + SVM + NB", 0.1, 0.635, 1.157, 1.160, 38])
        print(pt)
```

```
+--------+-------------------------------+------------+-------+-------------------+-------+-----
| S.No. |             Model             | Best alpha | Train | Cross Validation |  Test | Miscl
+--------+-------------------------------+------------+-------+-------------------+-------+-----
|   1   |          Naive Bayes          |   1e-05    | 0.603 |       1.245       | 1.273 |
|   2   |    LR with class balancing    |   0.0001   | 0.422 |       1.025       | 1.036 |
|   3   |  LR without class balancing   |   0.0001   |  0.42 |       1.066       | 1.065 |
|   4   |           Linear SVM          |   0.0001   | 0.415 |       1.023       | 1.082 |
|   6   | Stacking Models LR + SVM + NB |    0.1     | 0.635 |       1.157       |  1.16 |
+--------+-------------------------------+------------+-------+-------------------+-------+-----
```

- Performance table for tf-idf vectorizer with 8000 features

```
In [21]: ########################### performace table using top 8000 tfidf features ##########

        pt = PrettyTable()
        pt.field_names = ["S.No.","Model","Best alpha", "Train", "Cross Validation","Test", "M
        pt.add_row(["1","Naive Bayes", 0.001, 0.750, 1.236, 1.268, 38])
        pt.add_row(["2","LR with class balancing", 0.0001, 0.428, 1.028, 1.046, 31])
        pt.add_row(["3","LR without class balancing", 0.0001, 0.419, 1.056, 1.075, 31])
        pt.add_row(["4","Linear SVM", 0.001, 0.475, 1.019, 1.081, 34])
        pt.add_row(["6","Stacking Models LR + SVM + NB", 0.1, 0.648, 1.103, 1.134, 37])
        print(pt)
```

```
+--------+-------------------------------+------------+-------+-------------------+-------+-----
| S.No. |             Model             | Best alpha | Train | Cross Validation |  Test | Miscl
+--------+-------------------------------+------------+-------+-------------------+-------+-----
|   1   |          Naive Bayes          |   0.001    |  0.75 |       1.236       | 1.268 |
|   2   |    LR with class balancing    |   0.0001   | 0.428 |       1.028       | 1.046 |
|   3   |  LR without class balancing   |   0.0001   | 0.419 |       1.056       | 1.075 |
|   4   |           Linear SVM          |   0.001    | 0.475 |       1.019       | 1.081 |
|   6   | Stacking Models LR + SVM + NB |    0.1     | 0.648 |       1.103       | 1.134 |
+--------+-------------------------------+------------+-------+-------------------+-------+-----
```

3. Logistic Regression with CountVectorizer features Bigram

```
In [58]: # building a CountVectorizer with all the words that occured minimum 10 times in trai
        # only for Logistic Regression, as with TfidfVectorizer Logistic Regression is not pe
        text_vectorizer = CountVectorizer(min_df=10, ngram_range=(1, 2))
        train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
        # getting all the feature names (words)
        train_text_features= text_vectorizer.get_feature_names()

        # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*nu
        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 i
        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 : 219597
```

```
In [59]: # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

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

```
In [60]: # 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_
         test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
         cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_

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

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

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

```
In [61]: print("One hot encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_onehot
         print("(number of data points * number of features) in test data = ", test_x_onehotCod
         print("(number of data points * number of features) in cross validation data =", cv_x_
```

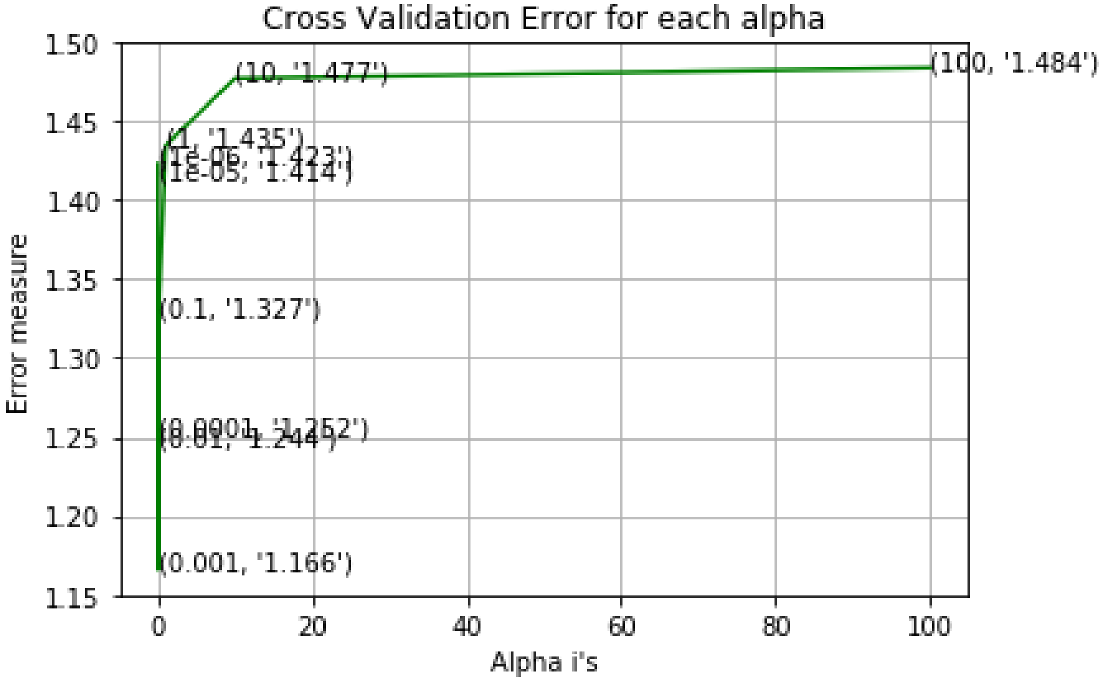
```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 221784)
(number of data points * number of features) in test data =  (665, 221784)
(number of data points * number of features) in cross validation data = (532, 221784)
```

Logistic Regression

```
In [62]: # read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
         # -------------------------------
         # default parameters
         # SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
         # shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
         # 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 Gr
         # predict(X)        Predict class labels for samples in X.

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



         # find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
         # ----------------------------
         # default paramters
         # sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
         #
         # 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', ra
             clf.fit(train_x_onehotCoding, train_y)
             sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
        # to avoid rounding error while multiplying probabilites we use log-probability e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

```
for alpha = 1e-06
Log Loss : 1.422987132578381
for alpha = 1e-05
Log Loss : 1.4143285387178606
for alpha = 0.0001
Log Loss : 1.251932992218001
for alpha = 0.001
Log Loss : 1.165735457465465
for alpha = 0.01
Log Loss : 1.2443011413725826
for alpha = 0.1
Log Loss : 1.3267430155937652
for alpha = 1
Log Loss : 1.4346156274978699
for alpha = 10
Log Loss : 1.4772648918502151
for alpha = 100
Log Loss : 1.4837941217305397
```

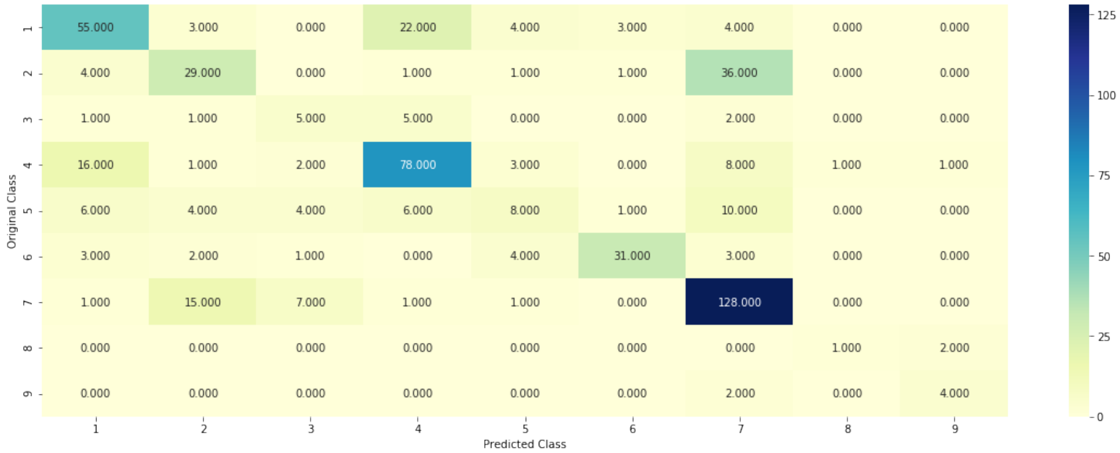


```
For values of best alpha =  0.001 The train log loss is: 0.5962160022873862
For values of best alpha =  0.001 The cross validation log loss is: 1.165735457465465
For values of best alpha =  0.001 The test log loss is: 1.126932847173988
```

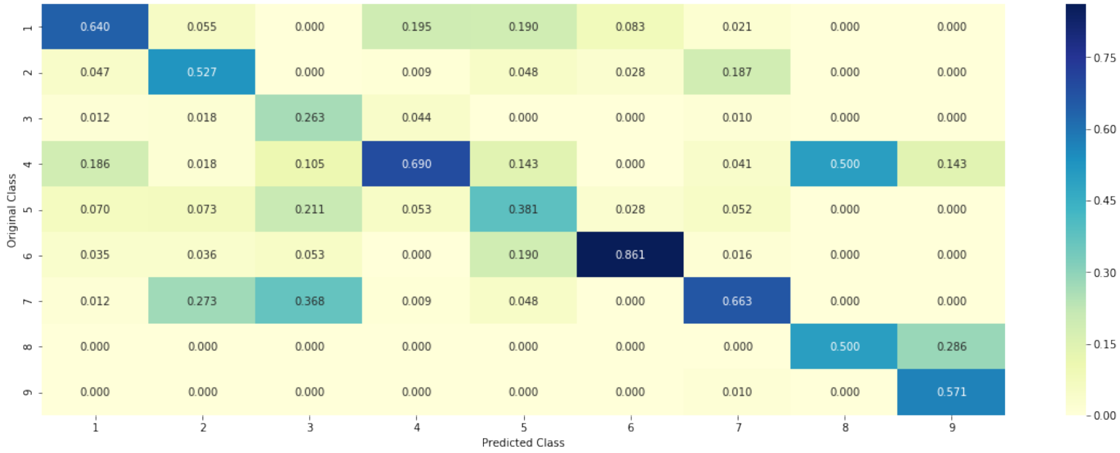
Testing

```
In [63]: # read more about support vector machines with linear kernals here http://scikit-lear

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

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


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

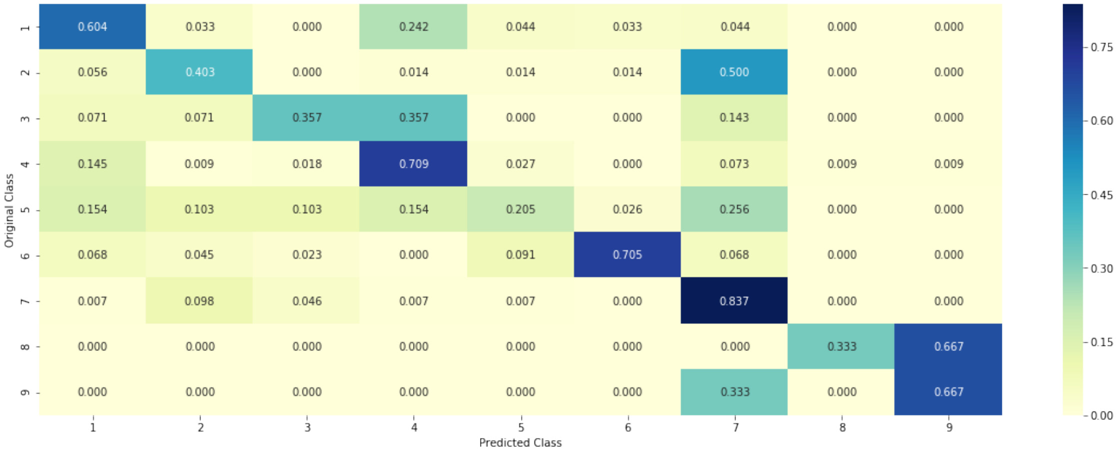
```
Log loss : 1.2108878723679422
Number of mis-classified points : 0.362781954887218O4
-------------------- Confusion matrix --------------------
```



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



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



0.0.2 4. Feature Engineerting

4.1. Using 4gram with KNN classifier

```
In [35]: # building a CountVectorizer with all the words that occured minimum 3 times in train
         # only for Logistic Regression, as with TfidfVectorizer Logistic Regression is not pe
         text_vectorizer = TfidfVectorizer(min_df=10, ngram_range=(1, 4))
         train_text_feature_onehotCoding = text_vectorizer.fit_transform(train_df['TEXT'])
         # getting all the feature names (words)
         train_text_features= text_vectorizer.get_feature_names()

         # train_text_feature_onehotCoding.sum(axis=0).A1 will sum every row and returns (1*nu
         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 i
         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 : 624053
```

```
In [36]: # don't forget to normalize every feature
         train_text_feature_onehotCoding = normalize(train_text_feature_onehotCoding, axis=0)

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

         # we use the same vectorizer that was trained on train data
```

```
          cv_text_feature_onehotCoding = text_vectorizer.transform(cv_df['TEXT'])
          # don't forget to normalize every feature
          cv_text_feature_onehotCoding = normalize(cv_text_feature_onehotCoding, axis=0)
```

```
In [46]: # 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_
         test_gene_var_onehotCoding = hstack((test_gene_feature_onehotCoding,test_variation_fea
         cv_gene_var_onehotCoding = hstack((cv_gene_feature_onehotCoding,cv_variation_feature_o

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

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

         cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).t
         cv_y = np.array(list(cv_df['Class']))
```

```
In [47]: print("One hot encoding features :")
         print("(number of data points * number of features) in train data = ", train_x_onehotC
         print("(number of data points * number of features) in test data = ", test_x_onehotCod
         print("(number of data points * number of features) in cross validation data =", cv_x_
```

```
One hot encoding features :
(number of data points * number of features) in train data =  (2124, 626258)
(number of data points * number of features) in test data =  (665, 626258)
(number of data points * number of features) in cross validation data = (532, 626258)
```

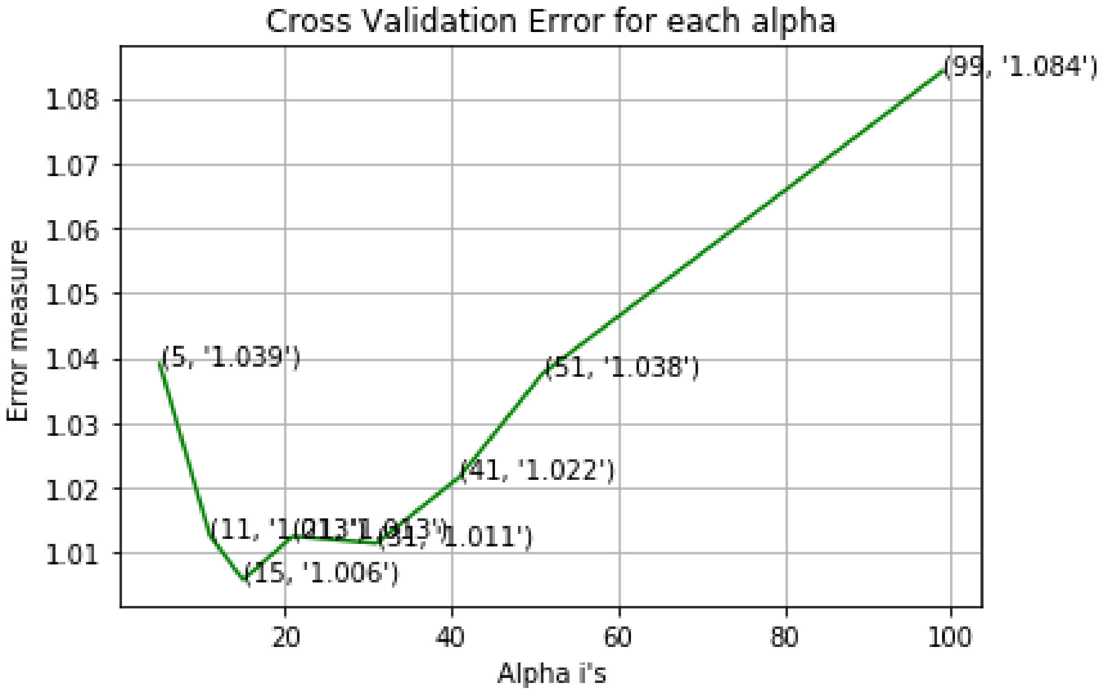
KNN Classifier

```
In [72]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/g
         # -------------------------
         # 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
```

```python
# predict_proba(X):Return probability estimates for the test data X.
#--------------------------------------
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/
#--------------------------------------


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


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

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


best_alpha = np.argmin(cv_log_error_array)
clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
clf.fit(train_x_responseCoding, train_y)
```

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

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

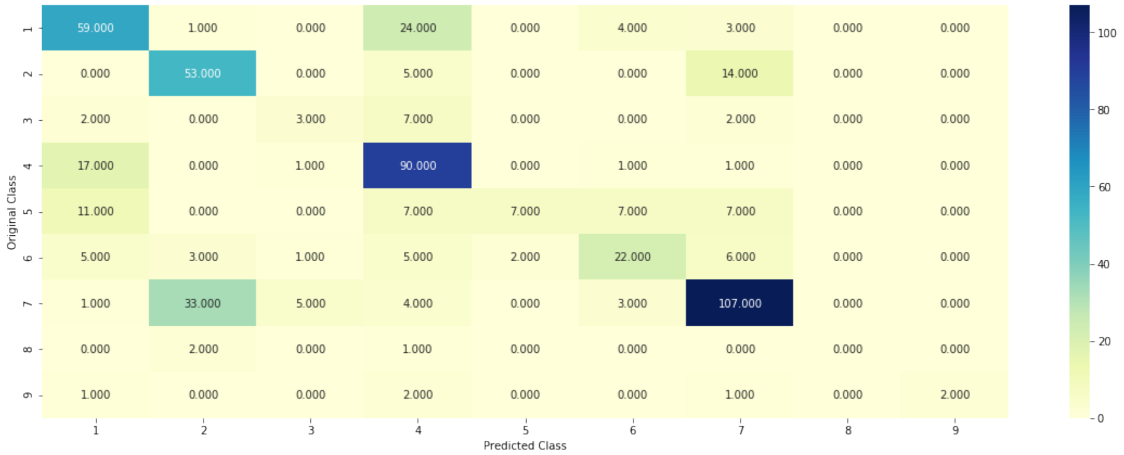
```
for alpha = 5
Log Loss : 1.0390009156532347
for alpha = 11
Log Loss : 1.012617055336673
for alpha = 15
Log Loss : 1.005755249592562
for alpha = 21
Log Loss : 1.0125019249245404
for alpha = 31
Log Loss : 1.0113895533748627
for alpha = 41
Log Loss : 1.0216166278237597
for alpha = 51
Log Loss : 1.0375597608462803
for alpha = 99
Log Loss : 1.0841212293675873
```



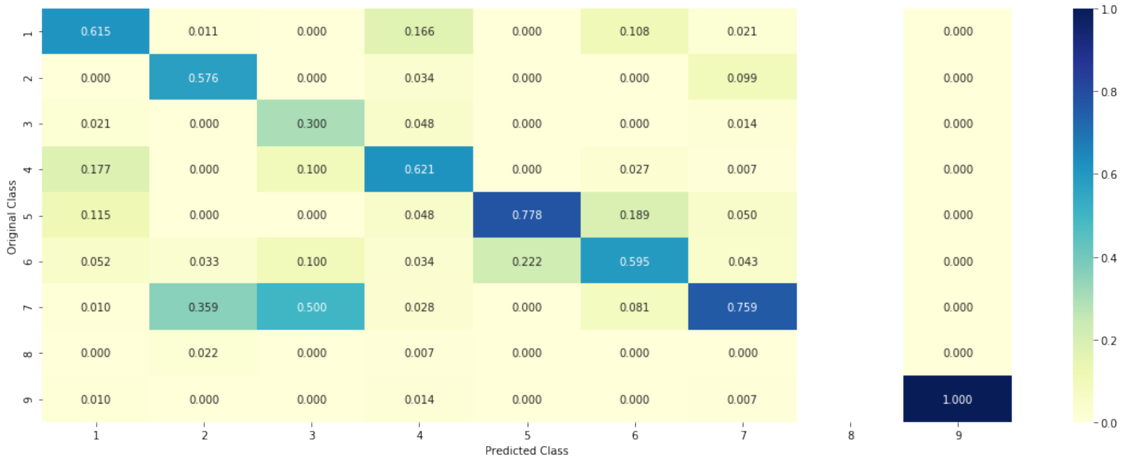
```
For values of best alpha =  15 The train log loss is: 0.6854622250743824
For values of best alpha =  15 The cross validation log loss is: 1.005755249592562
For values of best alpha =  15 The test log loss is: 1.0514381315733239
```

```
In [77]: # find more about KNeighborsClassifier() here http://scikit-learn.org/stable/modules/
         # -------------------------
         # default parameter
         # KNeighborsClassifier(n_neighbors=5, weights=uniform, algorithm=auto, leaf_size=30, 
         # 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
         #-------------------------------------
         clf = KNeighborsClassifier(n_neighbors=alpha[best_alpha])
         predict_and_plot_confusion_matrix(train_x_responseCoding, train_y, cv_x_responseCodin
```

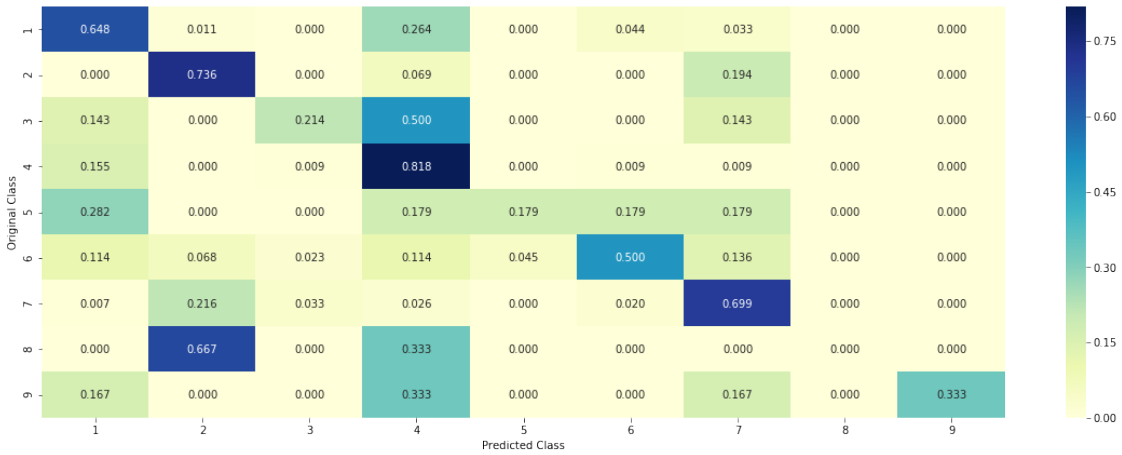
```
Log loss : 1.005755249592562
Number of mis-classified points : 0.35526315789473684
-------------------- Confusion matrix --------------------
```



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



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



4.2. Combining gene and variation feature together to form a new feature.

```
In [80]: # another try at feature engineering combining gene and variation feature:
         gene_and_variation = []

         for gene in result['Gene'].values:
             gene_and_variation.append(gene)

         for variation in result['Variation'].values:
             gene_and_variation.append(variation)

In [82]: len(gene_and_variation)
```

Out[82]: 6642

In [ ]:
```
TfidfVectorizer
```

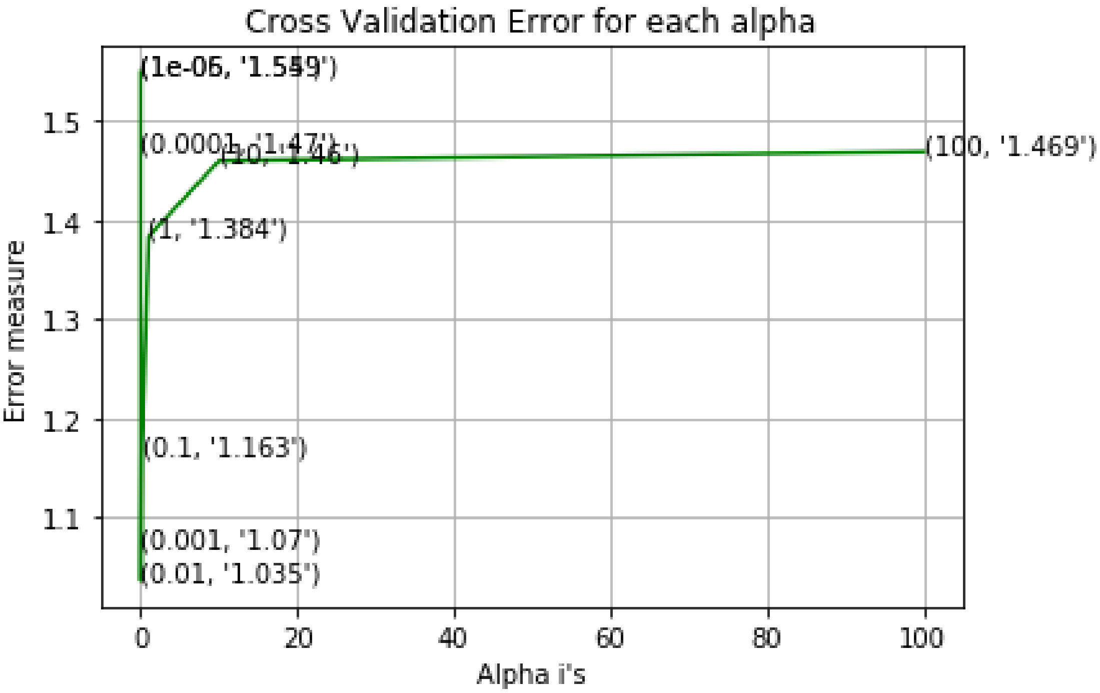
In [86]:
```
tfidf_vect = TfidfVectorizer(max_features=1000)
text2 = tfidf_vect.fit_transform(gene_and_variation)
gene_variation_features = tfidf_vect.get_feature_names()

train_text = tfidf_vect.transform(train_df['TEXT'])
test_text = tfidf_vect.transform(test_df['TEXT'])
cv_text = tfidf_vect.transform(cv_df['TEXT'])
```

In [88]:
```
# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/
# ------------------------------
# default parameters
# SGDClassifier(loss=hinge, penalty=l2, alpha=0.0001, l1_ratio=0.15, fit_intercept=Tr
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate=op
# 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 Gr
# predict(X)        Predict class labels for samples in X.

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



# find more about CalibratedClassifierCV here at http://scikit-learn.org/stable/modul
# ----------------------------
# default paramters
# sklearn.calibration.CalibratedClassifierCV(base_estimator=None, method=sigmoid, cv=
#
# 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', ra
    clf.fit(train_x_onehotCoding, train_y)
```

```
        sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
        sig_clf.fit(train_x_onehotCoding, train_y)
        sig_clf_probs = sig_clf.predict_proba(cv_x_onehotCoding)
        cv_log_error_array.append(log_loss(cv_y, sig_clf_probs, labels=clf.classes_, eps=
        # to avoid rounding error while multiplying probabilites we use log-probability e
        print("Log Loss :",log_loss(cv_y, sig_clf_probs))

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


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

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

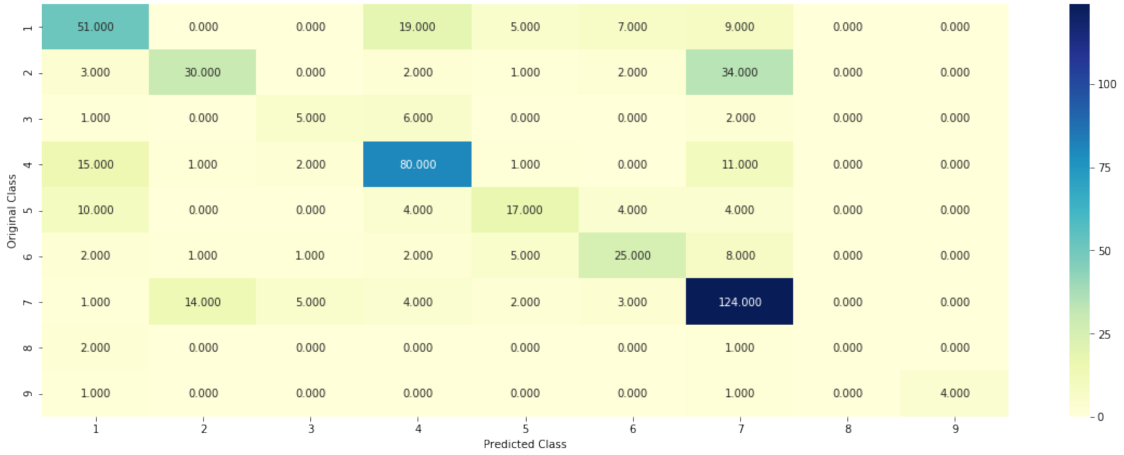
```
for alpha = 1e-06
Log Loss : 1.5500327531294866
for alpha = 1e-05
Log Loss : 1.5491397644770104
for alpha = 0.0001
Log Loss : 1.4696277660171502
for alpha = 0.001
Log Loss : 1.0702157238038494
for alpha = 0.01
Log Loss : 1.0353218164628526
for alpha = 0.1
Log Loss : 1.1629426133629324
for alpha = 1
Log Loss : 1.3836456423525918
for alpha = 10
Log Loss : 1.4602728393028062
for alpha = 100
```

```
Log Loss : 1.4694394300032734
```

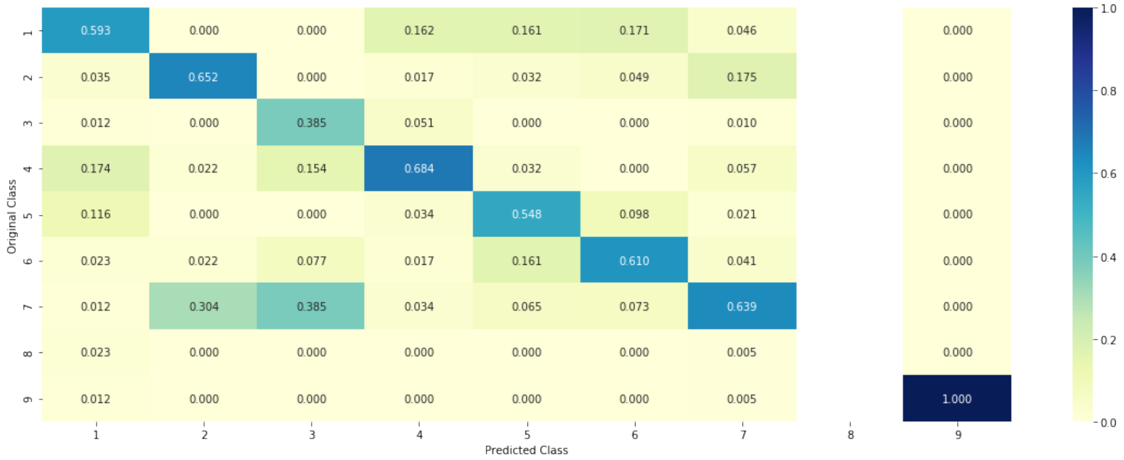


```
For values of best alpha =  0.01 The train log loss is: 0.6714468298210302
For values of best alpha =  0.01 The cross validation log loss is: 1.0353218164628526
For values of best alpha =  0.01 The test log loss is: 1.1430088823601718
```

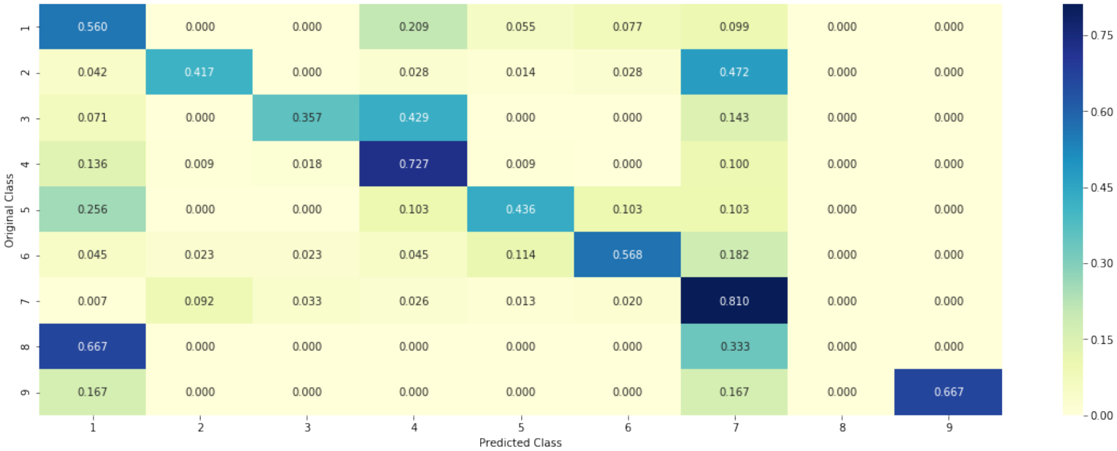
```
In [89]: # read more about support vector machines with linear kernals here http://scikit-lear

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

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


         # clf = SVC(C=alpha[best_alpha],kernel='linear',probability=True, class_weight='balan
         clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='hinge', random_state
         predict_and_plot_confusion_matrix(train_x_onehotCoding, train_y,cv_x_onehotCoding,cv_
Log loss : 1.1015006133661447
Number of mis-classified points : 0.3684210526315789
-------------------- Confusion matrix --------------------
```



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



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



This shows the results we got after completeing 4th task of feature engineering. I have two types of feature engineering.

```
In [12]: from prettytable import PrettyTable

         pt = PrettyTable()
         number   = [1,2]
         name     = ["KNN", "SGDClassifier with Log Loss"]
         feature  = ['4gram', 'combining gene with variation']
         tr_loss  = ["0.6854622250743824", '0.6714468298210302']
         te_loss  = ["1.005755249592562", '1.0353218164628526']


         #Initialize Prettytable
         pt = PrettyTable()
         pt.add_column("Index", number)
         pt.add_column("Model", name)
         pt.add_column("Feature Engineering", feature)
         pt.add_column("Train Log Loss", tr_loss)
         pt.add_column("Test Log Loss", te_loss)
         print(pt)
```

```
+-------+-----------------------------+-------------------------------+--------------------+---
| Index |            Model            |      Feature Engineering      |   Train Log Loss   |
+-------+-----------------------------+-------------------------------+--------------------+---
|   1   |             KNN             |             4gram             | 0.6854622250743824 | 1
|   2   | SGDClassifier with Log Loss | combining gene with variation | 0.6714468298210302 | 1
+-------+-----------------------------+-------------------------------+--------------------+---
```