

Personalized cancer diagnosis

1. Business Problem

1.1. Description

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

Data: Memorial Sloan Kettering Cancer Center (MSKCC)

Download training_variants.zip and training_text.zip from Kaggle.

Context:

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

Problem statement :

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

1.2. Source/Useful Links

Some articles and reference blogs about the problem statement

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

1.3. Real-world/Business objectives and constraints.

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

2. Machine Learning Problem Formulation

2.1. Data

2.1.1. Data Overview

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

2.1.2. Example Data Point

training_variants

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

training_text

ID, Text
0|Cyclin-dependent kinases (CDKs) regulate a variety of fundamental cellular processes. CDK10 stands out as one of the last orphan CDKs for which no activating cyclin has been identified and no kinase activity revealed. Previous work has shown that CDK10 silencing increases ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2)-driven activation of the MAPK pathway, which confers tamoxifen resistance to breast cancer cells. The precise mechanisms by which CDK10 modulates ETS2 activity, and more generally the functions of CDK10, remain elusive. Here we demonstrate that CDK10 is a cyclin-dependent kinase by identifying cyclin M as an activating cyclin. Cyclin M, an orphan cyclin, is the product of FAM58A, whose mutations cause STAR syndrome, a human developmental anomaly whose features include toe syndactyly, telecanthus, and anogenital and renal malformations. We show that STAR syndrome-associated cyclin M mutants are unable to interact with CDK10. Cyclin M silencing phenocopies CDK10 silencing in increasing c-Raf and in conferring tamoxifen resistance to breast cancer cells. CDK10/cyclin M phosphorylates ETS2 in vitro, and in cells it positively controls ETS2 degradation by the proteasome. ETS2 protein levels are increased in cells derived from a STAR patient, and this increase is attributable to decreased cyclin M levels. Altogether, our results reveal an additional regulatory mechanism for ETS2, which plays key roles in cancer and development. They also shed light on the molecular mechanisms underlying STAR syndrome. Cyclin-dependent kinases (CDKs) play a pivotal role in the control of a number of fundamental cellular processes (1). The human genome contains 21 genes encoding proteins that can be considered as members of the CDK family owing to their sequence similarity with bona fide CDKs, those known to be activated by cyclins (2). Although discovered almost 20 y ago (3, 4), CDK10 remains one of the two CDKs without an identified cyclin partner. This knowledge gap has largely impeded the exploration of its biological functions. CDK10 can act as a positive cell cycle regulator in some cells (5, 6) or as a tumor suppressor in others (7, 8). CDK10 interacts with the ETS2 (v-ets erythroblastosis virus E26 oncogene homolog 2) transcription factor and inhibits its transcriptional activity through an unknown mechanism (9). CDK10 knockdown derepresses ETS2, which increases the expression of the c-Raf protein kinase, activates the MAPK pathway, and induces resistance of MCF7 cells to tamoxifen (6). ...

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

2.2.1. Type of Machine Learning Problem

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

2.2.2. Performance Metric

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

Metric(s):

- Multi class log-loss
- Confusion matrix

2.2.3. Machine Learning Objectives and Constraints

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

Constraints:

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

2.3. Train, CV and Test Datasets

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

3. Exploratory Data Analysis

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

from mlxtend.classifier import StackingClassifier

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

3.1. Reading Data

3.1.1. Reading Gene and Variation Data

In [2]:

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

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

Out[2]:

	ID	Gene	Variation	Class

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

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

Fields are

- **ID** : the id of the row used to link the mutation to the clinical evidence
- **Gene** : the gene where this genetic mutation is located
- **Variation** : the aminoacid change for this mutations
- **Class** : 1-9 the class this genetic mutation has been classified on

3.1.2. Reading Text Data

In [3]:

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

Number of data points : 3321

Number of features : 2

Features : ['ID' 'TEXT']

Out[3]:

	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 [4]:

```
# 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 [5]:

```
#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 : 264.8572542 seconds
```

In [6]:

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

Out[6]:

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

In [7]:

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

Out[7]:

	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 [8]:

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

In [9]:

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

Out[9]:

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

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

3.1.4. Test, Train and Cross Validation Split

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

In [10]:

```
y_true = result['Class'].values
result.Gene = result.Gene.str.replace('\s+', '_')
result.Variation = result.Variation.str.replace('\s+', '_')
# split the data into test and train by maintaining same distribution of output variable 'y_true'
[stratify=y_true]
X_1, X_test, y_1, y_test = train_test_split(result, y_true, stratify=y_true, test_size=0.2)
# split the train data into train and cross validation by maintaining same distribution of output
variable 'y_train' [stratify=y_train]
X_train, X_cv, y_train, y_cv = train_test_split(X_1, y_1, stratify=y_1, test_size=0.2)
```

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

In [11]:

```
print('Number of data points in train data:', X_train.shape[0])
print('Number of data points in test data:', X_test.shape[0])
print('Number of data points in cross validation data:', X_cv.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 [12]:

```
#https://stackoverflow.com/questions/50802475/valueerror-invalid-rgba-argument-rgbkymc
# it returns a dict, keys as class labels and values as the number of data points in that class
train_class_distribution = X_train['Class'].value_counts().sort_index()
test_class_distribution = X_test['Class'].value_counts().sort_index()
cv_class_distribution = X_cv['Class'].value_counts().sort_index()

paired_colors = plt.cm.Paired(range(len(train_class_distribution)))
train_class_distribution.plot(kind='bar', color=paired_colors)
plt.xlabel('Class')
plt.ylabel('Data points per Class')
plt.title('Distribution of yi in train data')
plt.grid()
plt.show()

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

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

# ref: argsort https://docs.scipy.org/doc/numpy/reference/generated/numpy.argsort.html
# -(train_class_distribution.values): the minus sign will give us in decreasing order
sorted_vi = np.argsort(-test_class_distribution.values)
```

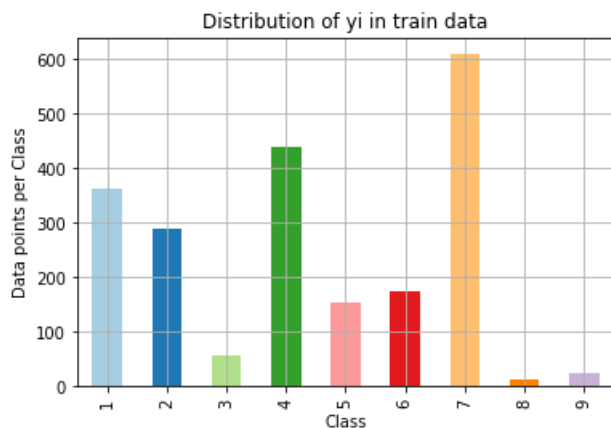
```

for i in sorted_yi:
    print('Number of data points in class', i+1, ':', test_class_distribution.values[i], '(', np.round(
nd((test_class_distribution.values[i]/X_test.shape[0]*100), 3), '%)')

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

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

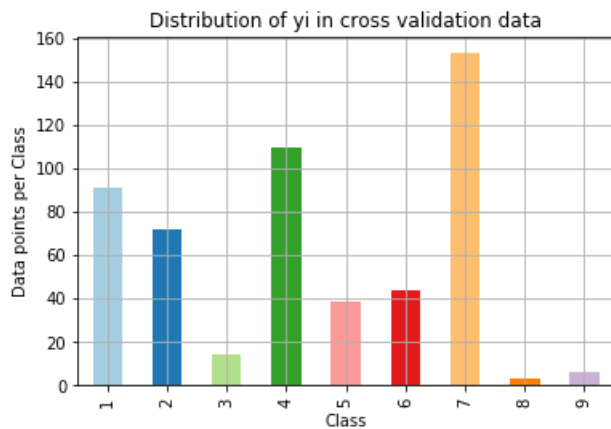
```



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



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



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

3.2 Prediction using a 'Random' Model

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

In [13]:

```

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

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

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

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

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

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



```

plt.xlabel('Predicted Class')
plt.ylabel('Original Class')
plt.show()

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

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

```

In [14]:

```

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

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

# Test-Set error.
#we create a output array that has exactly same as the test data
test_predicted_y = np.zeros((test_data_len,9))
for i in range(test_data_len):
    rand_probs = np.random.rand(1,9)
    test_predicted_y[i] = ((rand_probs/sum(sum(rand_probs))))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test,test_predicted_y, eps=1e-15))

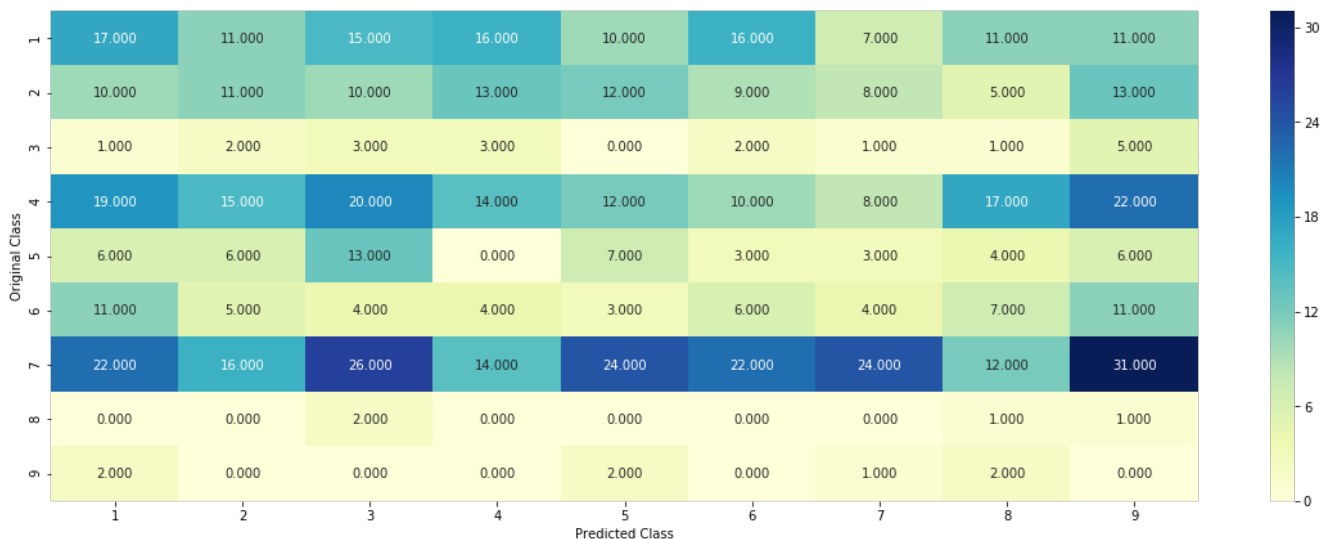
predicted_y = np.argmax(test_predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y+1)

```

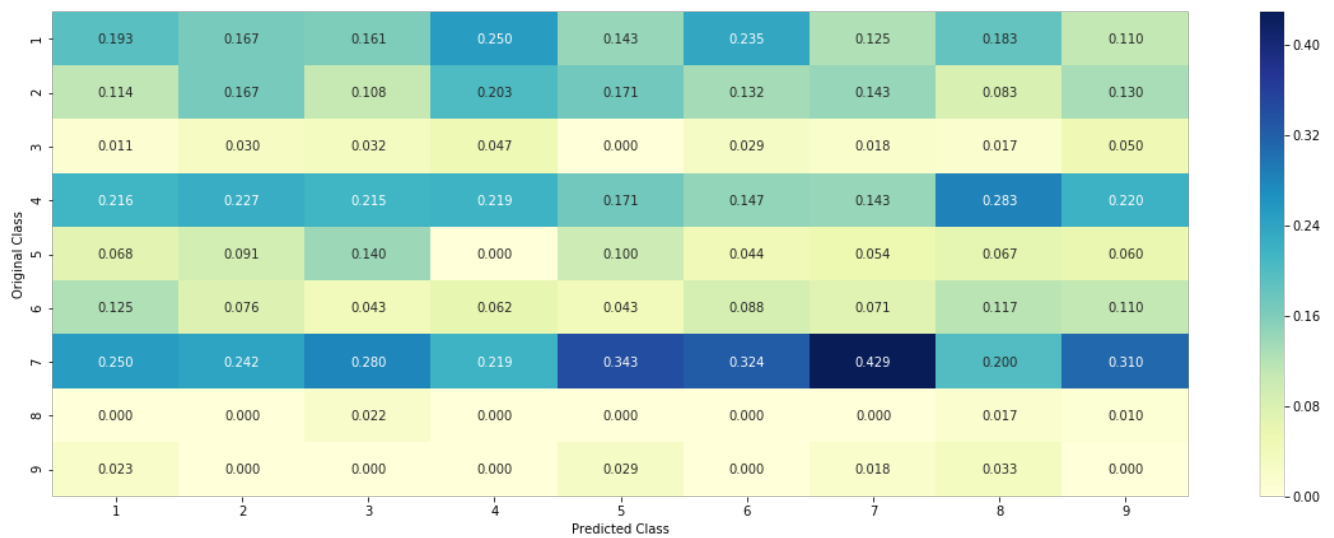
Log loss on Cross Validation Data using Random Model 2.45199307381341

Log loss on Test Data using Random Model 2.4933435235708292

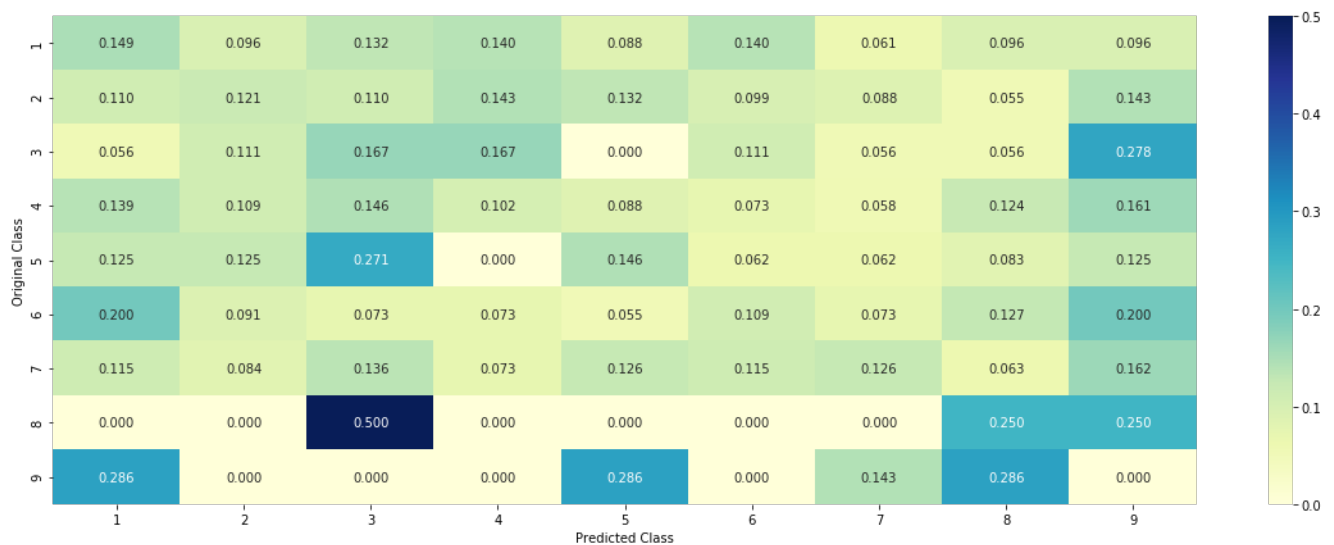
----- Confusion matrix -----



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



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



3.3 Univariate Analysis

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

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

3.2.1 Univariate Analysis on Gene Feature

Q1. Gene, What type of feature it is ?

Ans. Gene is a categorical variable

Q2. How many categories are there and How they are distributed?

In [15]:

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

Number of Unique Genes : 232

```
BRCA1      171
TP53       110
EGFR        93
BRCA2       83
PTEN        72
```

```
PIEN      13
KIT        65
BRAF       59
ERBB2      48
ALK        46
PDGFRA     43
Name: Gene, dtype: int64
```

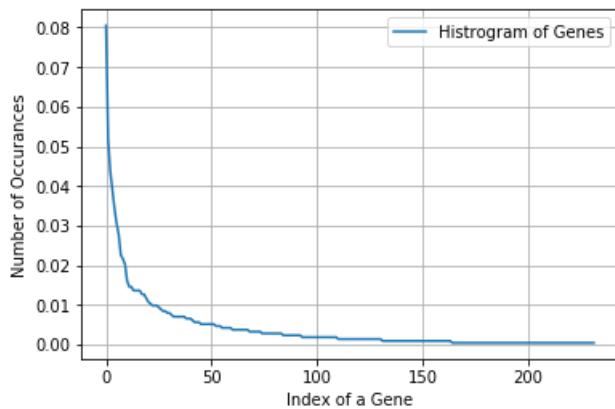
In [16]:

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

Ans: There are 232 different categories of genes in the train data, and they are distributed as follows

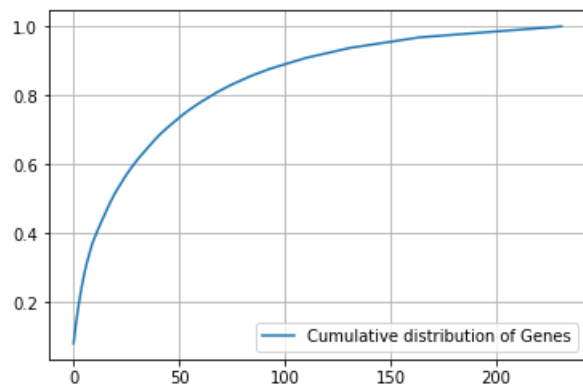
In [17]:

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



In [18]:

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



Q3. How to featurize this Gene feature ?

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

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical->

<https://www.appliedmlcourse.com/course/applied-ml-course-01/ml-lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

We will choose the appropriate featurization based on the ML model we use. For this problem of multi-class classification with categorical features, one-hot encoding is better for Logistic regression while response coding is better for Random Forests.

In [20]:

```
# tfidf encoding of Gene feature.
gene_tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,4))
train_gene_feature_tfidf = gene_tfidf_vectorizer.fit_transform(X_train['Gene'])
test_gene_feature_tfidf = gene_tfidf_vectorizer.transform(X_test['Gene'])
cv_gene_feature_tfidf = gene_tfidf_vectorizer.transform(X_cv['Gene'])
```

In [21]:

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

train_gene_feature_onehotCoding is converted feature using one-hot encoding method. The shape of gene feature: (2124, 232)

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 [22]:

```
alpha = [10 ** x for x in range(-5, 1)] # hyperparam for SGD classifier.

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

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

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

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

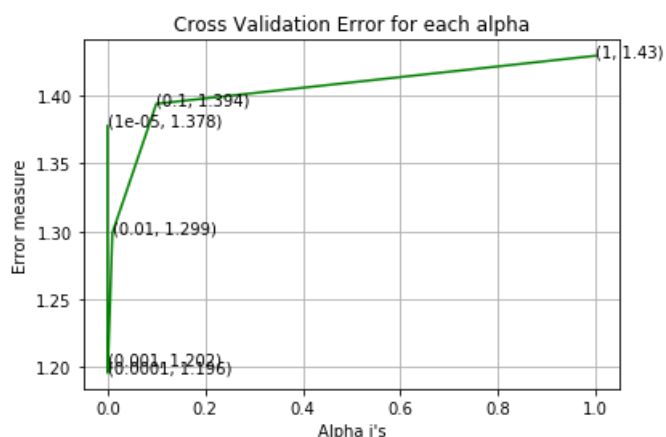
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

```
from sklearn import *
```

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

predict_y = sig_clf.predict_proba(train_gene_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_gene_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_gene_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))
```

For values of alpha = 1e-05 The log loss is: 1.3777902242941014
For values of alpha = 0.0001 The log loss is: 1.1955409419234948
For values of alpha = 0.001 The log loss is: 1.2020869572732933
For values of alpha = 0.01 The log loss is: 1.2988716156866777
For values of alpha = 0.1 The log loss is: 1.394158421476128
For values of alpha = 1 The log loss is: 1.4296025334763414



For values of best alpha = 0.0001 The train log loss is: 1.0382301362861248
For values of best alpha = 0.0001 The cross validation log loss is: 1.1955409419234948
For values of best alpha = 0.0001 The test log loss is: 1.232628761184949

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 [23]:

```
print("Q6. How many data points in Test and CV datasets are covered by the ", unique_genes.shape[0], " genes in train dataset?")

test_coverage=X_test[X_test['Gene'].isin(list(set(X_train['Gene'])))].shape[0]
cv_coverage=X_cv[X_cv['Gene'].isin(list(set(X_train['Gene'])))].shape[0]

print('Ans\n1. In test data', test_coverage, 'out of', X_test.shape[0], ":", (test_coverage/X_test.shape[0])*100)
print('2. In cross validation data', cv_coverage, 'out of', X_cv.shape[0], ":", (cv_coverage/X_cv.shape[0])*100)
```

Q6. How many data points in Test and CV datasets are covered by the 232 genes in train dataset?

Ans

1. In test data 638 out of 665 : 95.93984962406014
2. In cross validation data 513 out of 532 : 96.42857142857143

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 [24]:

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

```
Number of Unique Variations : 1929
Truncating_Mutations      60
Deletion                   48
Amplification              45
Fusions                    25
T58I                       3
G12V                       3
C618R                      2
P130S                      2
E330K                      2
Q22K                       2
Name: Variation, dtype: int64
```

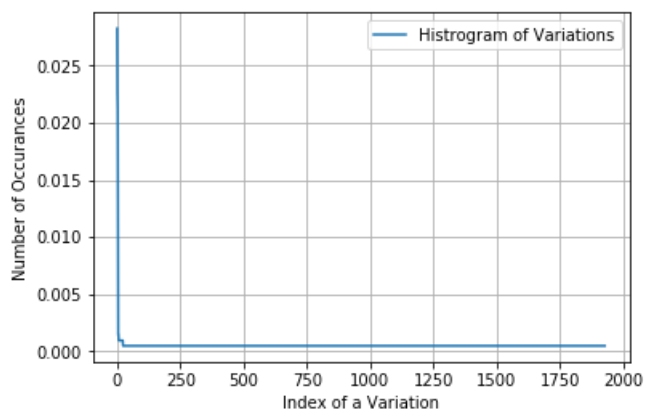
In [25]:

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

Ans: There are 1929 different categories of variations in the train data, and they are distributed as follows

In [26]:

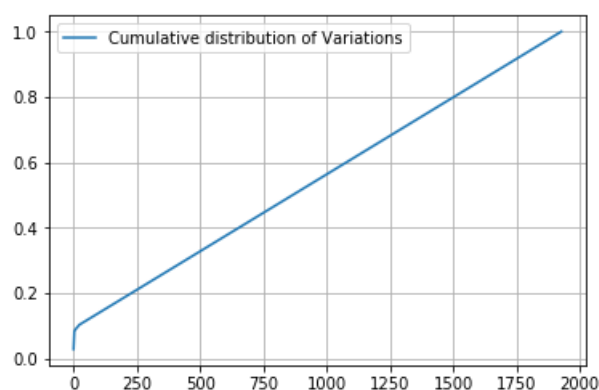
```
s = sum(unique_variations.values);
h = unique_variations.values/s;
plt.plot(h, label="Histogram of Variations")
plt.xlabel('Index of a Variation')
plt.ylabel('Number of Occurrences')
plt.legend()
plt.grid()
plt.show()
```



In [27]:

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

```
[0.02824859 0.05084746 0.0720339 ... 0.99905838 0.99952919 1. ]
```



Q9. How to featurize this Variation feature ?

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

<https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/handling-categorical-and-numerical-features/>

1. One hot Encoding
2. Response coding

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

In [29]:

```
# tfidf encoding of variation feature.
variation_tfidf_vectorizer = TfidfVectorizer(ngram_range=(1,4))
train_variation_feature_tfidf = variation_tfidf_vectorizer.fit_transform(X_train['Variation'])
test_variation_feature_tfidf = variation_tfidf_vectorizer.transform(X_test['Variation'])
cv_variation_feature_tfidf = variation_tfidf_vectorizer.transform(X_cv['Variation'])
```

In [30]:

```
print("train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding meth  
od. The shape of Variation feature:", train_variation_feature_tfidf.shape)
```

train_variation_feature_onehotEncoded is converted feature using the onne-hot encoding method. The shape of Variation feature: (2124, 2055)

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

Let's build a model just like the earlier!

In [31]:

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

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

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

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

```

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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(train_variation_feature_tfidf, y_train)
    predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)

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

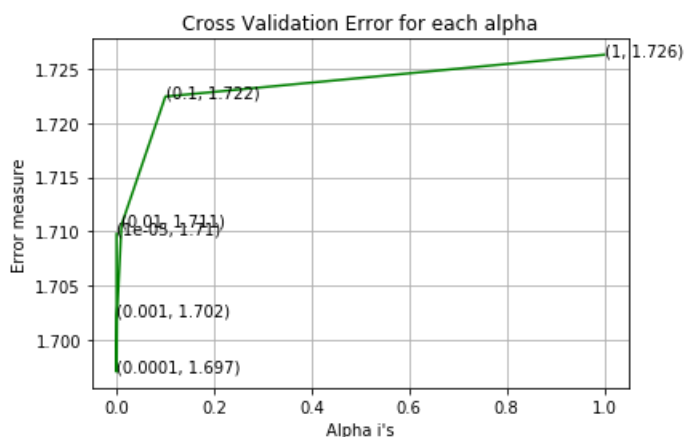
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

predict_y = sig_clf.predict_proba(train_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_variation_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.7097717824898069
 For values of alpha = 0.0001 The log loss is: 1.6970720464583409
 For values of alpha = 0.001 The log loss is: 1.7023351128062958
 For values of alpha = 0.01 The log loss is: 1.7106029551688564
 For values of alpha = 0.1 The log loss is: 1.722422410641387
 For values of alpha = 1 The log loss is: 1.7262726588310868



For values of best alpha = 0.0001 The train log loss is: 0.7531880584064388
 For values of best alpha = 0.0001 The cross validation log loss is: 1.6970720464583409
 For values of best alpha = 0.0001 The test log loss is: 1.7191119532157744

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 [32]:

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

Q12. How many data points are covered by total 1929 genes in test and cross validation data sets?

Ans

1. In test data 65 out of 665 : 9.774436090225564
2. In cross validation data 59 out of 532 : 11.090225563909774

3.2.3 Univariate Analysis on Text Feature

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

In [125]:

```
# tfidf encoding of text feature.
text_tfidf_vectorizer = TfidfVectorizer(max_features=4000,ngram_range=(1,1))
train_text_feature_tfidf = text_tfidf_vectorizer.fit_transform(X_train['TEXT'])
test_text_feature_tfidf = text_tfidf_vectorizer.transform(X_test['TEXT'])
cv_text_feature_tfidf = text_tfidf_vectorizer.transform(X_cv['TEXT'])
```

In [126]:

```
#normalizing tfidf features
train_text_feature_tfidf = normalize(train_text_feature_tfidf, axis=0)
cv_text_feature_tfidf = normalize(cv_text_feature_tfidf, axis=0)
test_text_feature_tfidf = normalize(test_text_feature_tfidf, axis=0)
```

In [50]:

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

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

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

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

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

    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
```

```

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

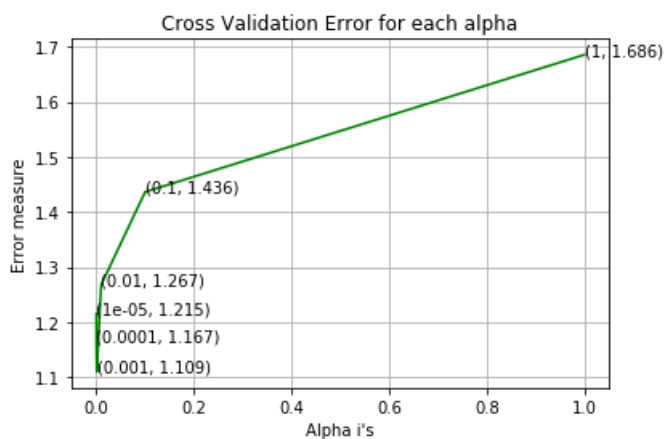
fig, ax = plt.subplots()
ax.plot(alpha, cv_log_error_array, c='g')
for i, txt in enumerate(np.round(cv_log_error_array, 3)):
    ax.annotate((alpha[i], np.round(txt, 3)), (alpha[i], cv_log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error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

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

predict_y = sig_clf.predict_proba(train_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_text_feature_tfidf)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))

```

For values of alpha = 1e-05 The log loss is: 1.2145502406561253
 For values of alpha = 0.0001 The log loss is: 1.1669258347984812
 For values of alpha = 0.001 The log loss is: 1.1093672622234114
 For values of alpha = 0.01 The log loss is: 1.2674720473999108
 For values of alpha = 0.1 The log loss is: 1.4363518884785482
 For values of alpha = 1 The log loss is: 1.6857539281972111



For values of best alpha = 0.001 The train log loss is: 0.6842472614801208
 For values of best alpha = 0.001 The cross validation log loss is: 1.1093672622234114
 For values of best alpha = 0.001 The test log loss is: 1.0992063070303926

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

Ans. Yes, it seems like!

4. Machine Learning Models

In [36]:

```

#Data preparation for ML models.

#Misc functions for ML models

```

```
# ALSO: FUNCTIONS FOR THE MODELS
```

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

    # for calculating log_loss we will provide the array of probabilities belongs to each class
    print("Log loss :", log_loss(test_y, sig_clf.predict_proba(test_x)))
    # calculating the number of data points that are misclassified
    global mis_per
    mis_per = (np.count_nonzero((pred_y - test_y)) / test_y.shape[0]) * 100
    print("Number of mis-classified points :", np.count_nonzero((pred_y - test_y)) / test_y.shape[0])
    plot_confusion_matrix(test_y, pred_y)
```

In [37]:

```
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 [38]:

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

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

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

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

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

Stacking the three types of features

In [127]:

```
# merging gene, variance and text features
```

```

# merging gene, variance and text features

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

#train_gene_var_onehotCoding =
hstack((train_gene_feature_onehotCoding,train_variation_feature_onehotCoding))
#test_gene_var_onehotCoding =
hstack((test_gene_feature_onehotCoding,test_variation_feature_onehotCoding))
#cv_gene_var_onehotCoding =
hstack((cv_gene_feature_onehotCoding,cv_variation_feature_onehotCoding))
#
#train_x_onehotCoding = hstack((train_gene_var_onehotCoding,
train_text_feature_onehotCoding)).tocsr()
#train_y = np.array(list(X_train['Class']))
#
#test_x_onehotCoding = hstack((test_gene_var_onehotCoding,
test_text_feature_onehotCoding)).tocsr()
#test_y = np.array(list(X_test['Class']))
#
#cv_x_onehotCoding = hstack((cv_gene_var_onehotCoding, cv_text_feature_onehotCoding)).tocsr()
#cv_y = np.array(list(X_cv['Class']))

train_gene_var_tfidf=hstack((train_gene_feature_tfidf,train_variation_feature_tfidf))
test_gene_var_tfidf=hstack((test_gene_feature_tfidf,test_variation_feature_tfidf))
cv_gene_var_tfidf=hstack((cv_gene_feature_tfidf,cv_variation_feature_tfidf))

train_x_tfidf=hstack((train_gene_var_tfidf,train_text_feature_tfidf))
test_x_tfidf=hstack((test_gene_var_tfidf,test_text_feature_tfidf))
cv_x_tfidf=hstack((cv_gene_var_tfidf,cv_text_feature_tfidf))

```

In [128]:

```

print(" tfidf encoding features :")
print("(number of data points * number of features) in train data = ", train_x_tfidf.shape)
print("(number of data points * number of features) in test data = ", test_x_tfidf.shape)
print("(number of data points * number of features) in cross validation data =", cv_x_tfidf.shape)

```

```

tfidf encoding features :
(number of data points * number of features) in train data =  (2124, 6287)
(number of data points * number of features) in test data =  (665, 6287)
(number of data points * number of features) in cross validation data = (532, 6287)

```

4.3. Logistic Regression

4.3.1. With Class balancing

4.3.1.1. Hyper paramter tuning

In [129]:

```

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

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

```

```

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

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

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

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

best_alpha = np.argmin(cv_log_error_array)
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
clf.fit(train_x_tfidf, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(train_x_tfidf, y_train)

predict_y = sig_clf.predict_proba(train_x_tfidf)
train_ll=log_loss(y_train, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(cv_x_tfidf)
cv_ll=log_loss(y_cv, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The cross validation log loss is:", log_lo
ss(y_cv, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(test_x_tfidf)
test_ll=log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, p
redict_y, labels=clf.classes_, eps=1e-15))

```

```

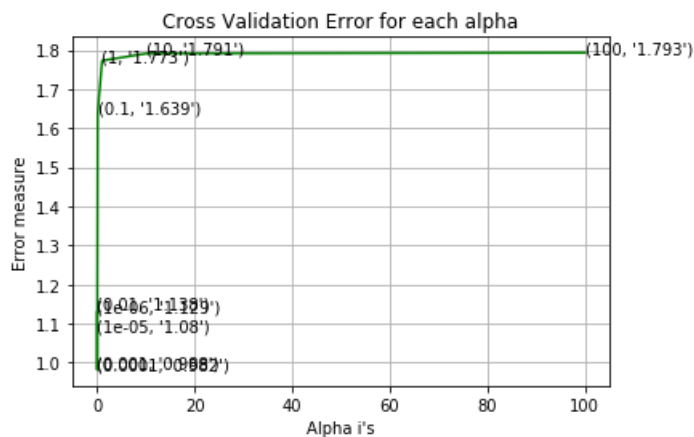
for alpha = 1e-06
Log Loss : 1.129094329872638
for alpha = 1e-05
Log Loss : 1.0797474704134136
for alpha = 0.0001
Log Loss : 0.9816992957940038
for alpha = 0.001
Log Loss : 0.9875243326588317
for alpha = 0.01
Log Loss : 1.1379930835730445

```

```

for alpha = 0.1
Log Loss : 1.6392115209584144
for alpha = 1
Log Loss : 1.7727759314804934
for alpha = 10
Log Loss : 1.790921494446961
for alpha = 100
Log Loss : 1.7930334091071638

```



```

For values of best alpha = 0.0001 The train log loss is: 0.4341576175370888
For values of best alpha = 0.0001 The cross validation log loss is: 0.9816992957940038
For values of best alpha = 0.0001 The test log loss is: 0.9860794684134891

```

4.3.1.2. Testing the model with best hyper paramters

In [92]:

```

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

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

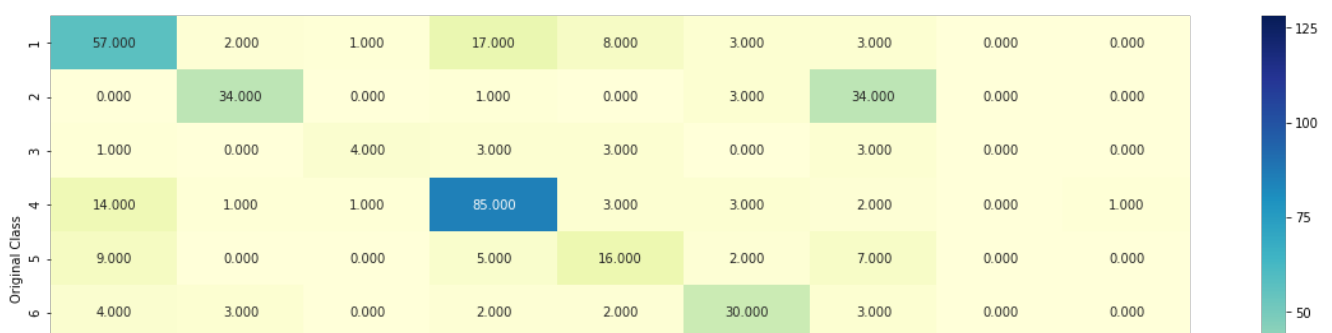
#-----
# video link: https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/geometric-in
tuition-1/
#-----
clf = SGDClassifier(class_weight='balanced', alpha=alpha[best_alpha], penalty='l2', loss='log', ran
dom_state=42)
predict_and_plot_confusion_matrix(train_x_tfidf, y_train, cv_x_tfidf, y_cv, clf)

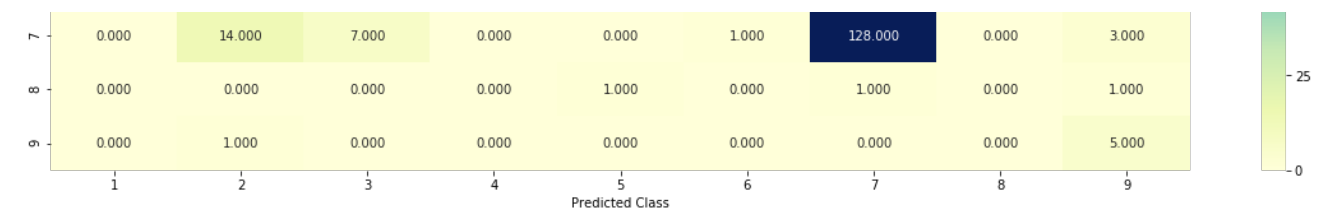
```

```

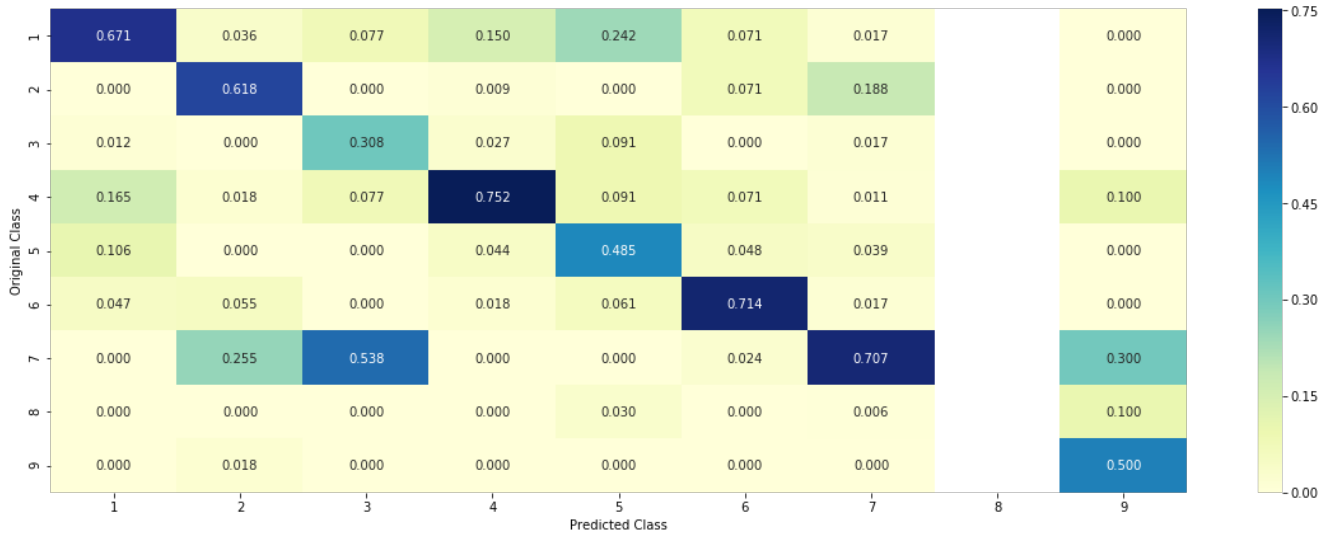
Log loss : 0.9949881000031798
Number of mis-classified points : 0.325187969924812
----- Confusion matrix -----

```

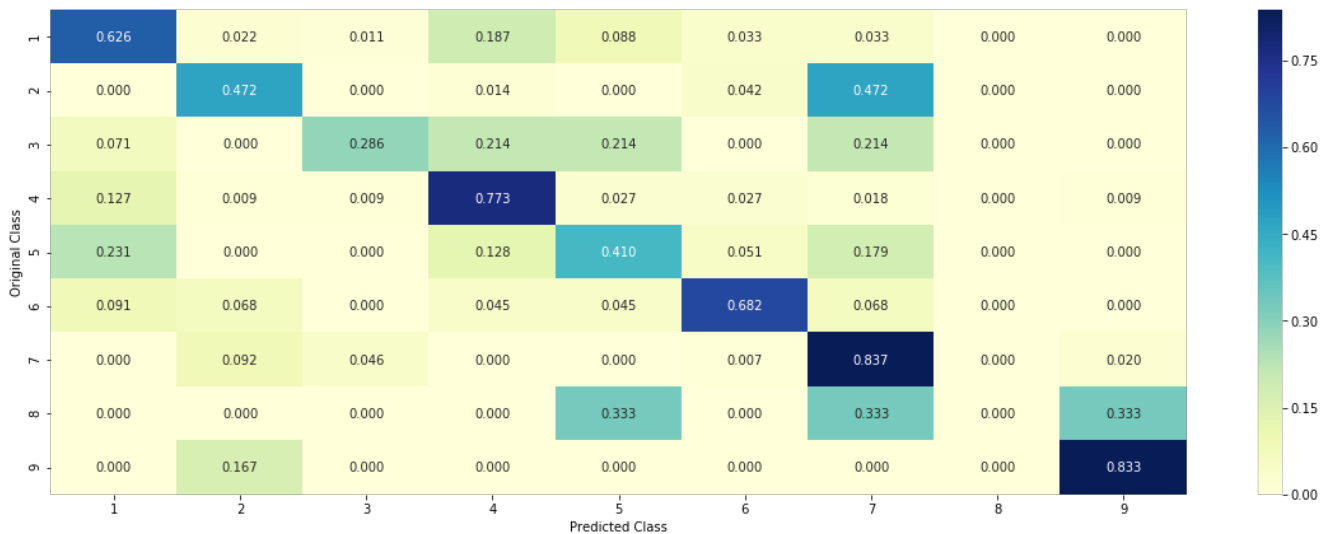




Precision matrix (Column Sum=1)



Recall matrix (Row sum=1)



I've run the model for multiple values of max no of features and n-grams range and recorded the below values. For max no of features of 5000 i was able to record the CV log loss below 1.0 and for max no of features of 4000 i was able to record both CV and test log loss below 1.0

In [1]:

```
from prettytable import PrettyTable

x = PrettyTable()

x.field_names = ["ML Model", "Max no of features", "n-gram range", "CV Log Loss", "Test Log loss"]
x.add_row(["Logistic Regression(TFIDF)", "4000", "(1,1)", "0.981", "0.986"])
x.add_row(["Logistic Regression(TFIDF)", "4000", "(1,2)", "0.980", "0.994"])
x.add_row(["Logistic Regression(TFIDF)", "4000", "(1,3)", "0.984", "0.998"])
x.add_row(["Logistic Regression(TFIDF)", "4000", "(1,4)", "0.983", "0.999"])
x.add_row(["Logistic Regression(TFIDF)", "5000", "(1,1)", "0.99", "1.01"])
```

```

x.add_row(["Logistic Regression(TFIDF) ", "5000", "(1,2)", "0.98", "1.01"])
x.add_row(["Logistic Regression(TFIDF) ", "5000", "(1,3)", "0.99", "1.02"])
x.add_row(["Logistic Regression(TFIDF) ", "5000", "(1,4)", "0.99", "1.03"])
x.add_row(["Logistic Regression(TFIDF) ", "10000", "(1,3)", "1.01", "1.17"])
x.add_row(["Logistic Regression(TFIDF) ", "10000", "(1,4)", "1.01", "1.11"])
x.add_row(["Logistic Regression(TFIDF) ", "15000", "(1,4)", "1.04", "1.16"])
x.add_row(["Logistic Regression(TFIDF) ", "15000", "(1,5)", "1.04", "1.11"])
x.add_row(["Logistic Regression(TFIDF) ", "20000", "(1,4)", "1.05", "1.13"])
x.add_row(["Logistic Regression(TFIDF) ", "20000", "(1,3)", "1.05", "1.10"])
print(x)

```

ML Model	Max no of features	n-gram range	CV Log Loss	Test Log loss
Logistic Regression(TFIDF)	4000	(1,1)	0.981	0.986
Logistic Regression(TFIDF)	4000	(1,2)	0.980	0.994
Logistic Regression(TFIDF)	4000	(1,3)	0.984	0.998
Logistic Regression(TFIDF)	4000	(1,4)	0.983	0.999
Logistic Regression(TFIDF)	5000	(1,1)	0.99	1.01
Logistic Regression(TFIDF)	5000	(1,2)	0.98	1.01
Logistic Regression(TFIDF)	5000	(1,3)	0.99	1.02
Logistic Regression(TFIDF)	5000	(1,4)	0.99	1.03
Logistic Regression(TFIDF)	10000	(1,3)	1.01	1.17
Logistic Regression(TFIDF)	10000	(1,4)	1.01	1.11
Logistic Regression(TFIDF)	15000	(1,4)	1.04	1.16
Logistic Regression(TFIDF)	15000	(1,5)	1.04	1.11
Logistic Regression(TFIDF)	20000	(1,4)	1.05	1.13
Logistic Regression(TFIDF)	20000	(1,3)	1.05	1.10

5. Assignments

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